Implementation of Deadlock Detection in a Simulated Network Environment

Sugath Warnakulasuriya and Timothy Mark Pinkston

CENG Technical Report 97-01

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4465

January 1997

Implementation of Deadlock Detection in a Simulated Network Environment

Sugath Warnakulasuriya and Timothy Mark Pinkston

January 1997

Abstract

We develop and present an approach to modeling resource allocations and dependencies within cut-through and wormhole interconnection networks. We also present a theoretical framework which allows the precise description of various types of message blocking behavior including deadlock. We show that a knot within a channel wait-for graph is a necessary and sufficient condition for deadlock. Our framework allows for distinguishing between messages involved in deadlock and those simply dependent on deadlock, and it shows that recovery from deadlock requires "removal" of messages in the knot. Finally, we describe an implementation of deadlock detection in a network simulator used for detailed deadlock characterization and give its time and space complexity.

1. Introduction

Interconnection networks that allow unrestricted use of channel resources are susceptible to deadlock. Deadlocks are caused by cyclic dependencies formed as consumers hold resources while waiting to acquire additional resources. In interconnection networks, messages are the consumers of resources and virtual channel buffers are the resources. Deadlock, if not properly resolved, results in the permanent blocking of messages being routed.

Previous studies have not provided insight into the frequency and characteristics of deadlock in interconnection networks, nor have they identified network parameters and the degrees to which they influence deadlock [1, 2, 3, 4]. A practical approach to better understanding deadlock is through detailed simulation. We have developed an interconnection network simulator capable of true deadlock detection. We use this simulator to perform detailed deadlock characterization.

This paper describes our approach for detecting deadlock. We first develop and present a theoretical framework which precisely defines deadlock within interconnection networks and prove the necessary and sufficient conditions for deadlock. We then describe our approach for detecting deadlock and present details of our implementation of deadlock detection. We begin by introduc-

ing our approach for modeling resource dependencies within a network in Section 2. Section 3 presents a formal model of deadlock. Section 4 describes our approach for detecting deadlock. Related work is discussed in Section 5, and we conclude with a summary of significant findings in Section 6.

2. Network Resource Model

We present here through example our approach to modeling resource dependencies, deadlock, and other types of message blocking behavior within interconnection networks. Our model supports maximally adaptive routing, and thus accommodates both restricted (deadlock avoidance-based [5, 6, 7]) and unrestricted (deadlock recovery-based [2, 3, 4]) routing algorithms. The model is formalized in Section 3.

Our resource model uses *channel wait-for graphs* (CWGs) to represent resource allocations and requests *existing within the network at a given point in time*. We describe CWGs through example here and precisely define them in Section 3. The CWGs consist of a set of vertices $vc_1 ... vc_n$, each representing an independently allocatable virtual channel resource in the network. The messages in the network at a given time are uniquely identified as $m_1 ... m_k$. We assume a fixed number of reusable virtual channels and a dynamically variable number of messages which can be in the network at a given time for a given network configuration. We describe how vertices in the CWG are connected to reflect resource allocations to and requests by messages in the following subsections.

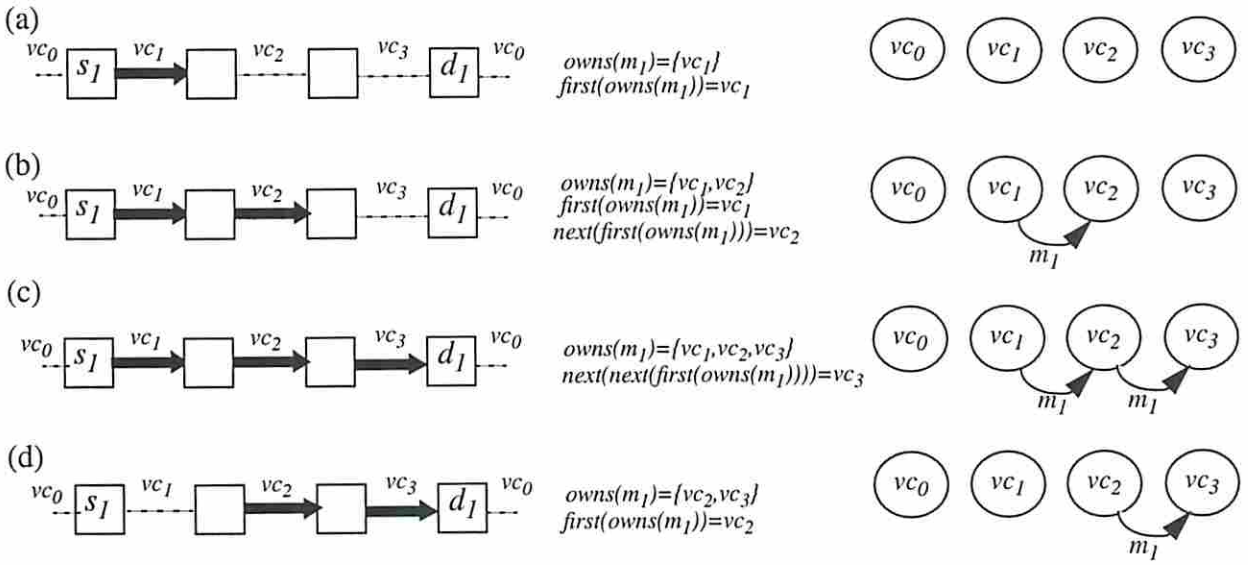

Figure 1- Network and CWG states for message m_1 being routed from source s_1 to destination d_1 in a unidirectional ring network with one VC per physical channel.

2.1 Messages and Blocked Messages

When a message progresses through the network, it acquires exclusive ownership of a new virtual channel (VC) prior to each hop. If the message header blocks, it awaits the exclusive use of one of possibly many alternative VCs in order to progress to the next hop. A blocked message can resume once a new VC is acquired. As the tail of a message moves through the network, it

releases (in the order of acquisition) previously acquired VCs which are no longer needed, and these VCs then become available for other messages to acquire.

We represent VCs owned by message m_i as the set *owns*(m_i) and the order in which the VCs were acquired using the functions *first* and *next*. As an example, Figure 1 shows the network and CWG states for a message m_1 being routed from its source node s_1 to its destination d_1 in a unidirectional ring network (*4*-ary *1*-cube) with one VC per physical channel. Figures 1 (a), (b), and (c) show the network and CWG states as the message header moves towards and reaches its destination (assuming a message length > 3 flits). The order in which this message acquired the VCs it owns at a given time can be expressed by *first*(*owns*(m_1)), *next*(*first*(*owns*(m_1))), *next*(*next*(*first*(*owns*(m_1)))), etc. Figure 1 (d) shows the state after the tail of the message has left the source node and has released the first VC it acquired.

The routing function determines the alternative VCs a message may use to reach the next node for a given current node and destination node pair. When none of the VCs supplied by the routing function are available (due to those VCs being owned by other messages or link failure), the message becomes blocked. For a blocked message m_i, the set *requests*(m_i) represents the set of alternative VCs it may use to resume (thus reflecting the set of VCs supplied by the routing function). As an example, Figure 2 shows four messages m_1...m_4 within a *4x2* unidirectional torus network with one VC per physical channel and which uses minimal adaptive routing. In this example, messages m_2 and m_4 have acquired all of the VCs needed to reach their respective destinations, while message m_1 and m_3 are blocked waiting for resources owned by the other messages (a listing of the *owns* and *requests* sets for each of the messages is included in Figure 2). Message m_1 has yet to exhaust its adaptivity and, therefore, is supplied with two alternative VCs by the routing function (as reflected by the set *requests*(m_1)). On the other hand, message m_1 has exhausted its adaptivity and is therefore supplied only a single VC by the routing function.

In the CWG illustrations, dashed arcs (also referred to as "request arcs") are used to represent the relationship between the last VCs owned by blocked messages and the set of VCs they "wait-for" in order to continue. The vertices from which these request arcs originate are referred to as "fanout vertices", as they are the only vertices in the CWGs which may have multiple outgoing arcs (all but the last owned VC of a message have a single outgoing arc). The number of possible outgoing arcs at the fanout vertices is determined by various factors including the number of physical and virtual channel resources in the network and the amount of routing freedom allowed in the use of these resources by the routing algorithm.

In order for a blocked message m_i to resume, it must acquire any one of the VCs in the set *requests*(m_i) after it is released by its previous owner. Continuing the example in Figure 2, when message m_4 releases vc_3 (after its tail flit has traversed this channel), the VC can be acquired by message m_3. The CWG state resulting from this along with the updated *owns* and *requests* are shown in Figure 3a. Subsequently, message m_2 may release vc_{13}, thus allowing message m_1 to continue. The CWG depicting this is shown in Figure 3b. In such cases where a blocked message resumes, its *requests* set becomes empty as any one of its members is acquired and becomes a member of the *owns* set. In essence, a message has dashed arcs (in the CWG illustrations) to *all* of

its alternative VCs when it is blocked, and these dashed arcs are replaced with a *single* solid arc to the newly acquired VC upon resumption. We discuss exceptions to this for supporting VC buffer sharing in Section 3.1.

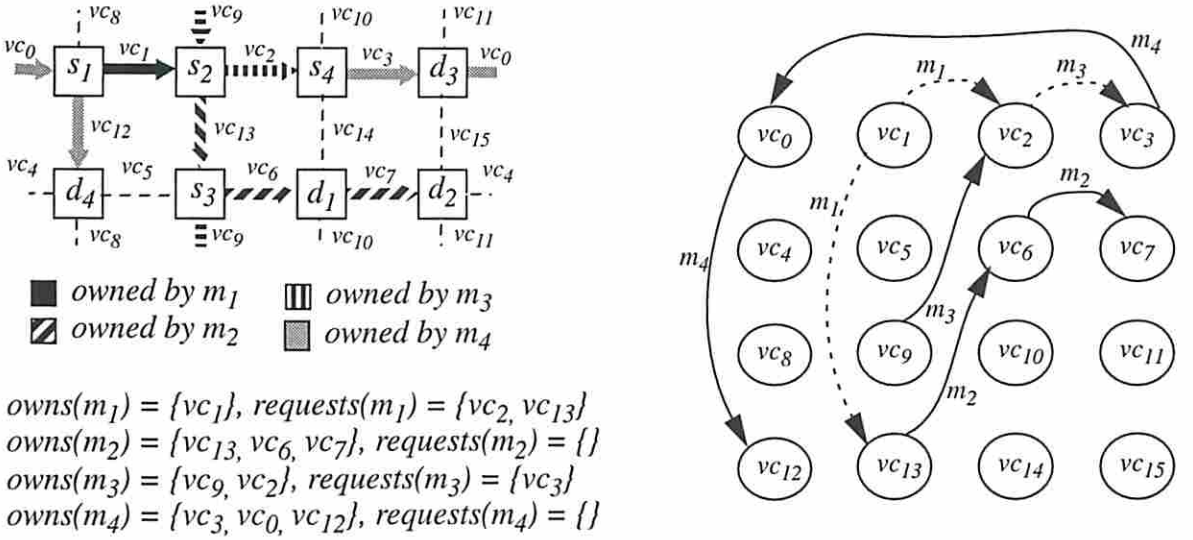

Figure 2 - Network and CWG state for messages m_1...m_4 being routed within a *2x4* unidirectional network.

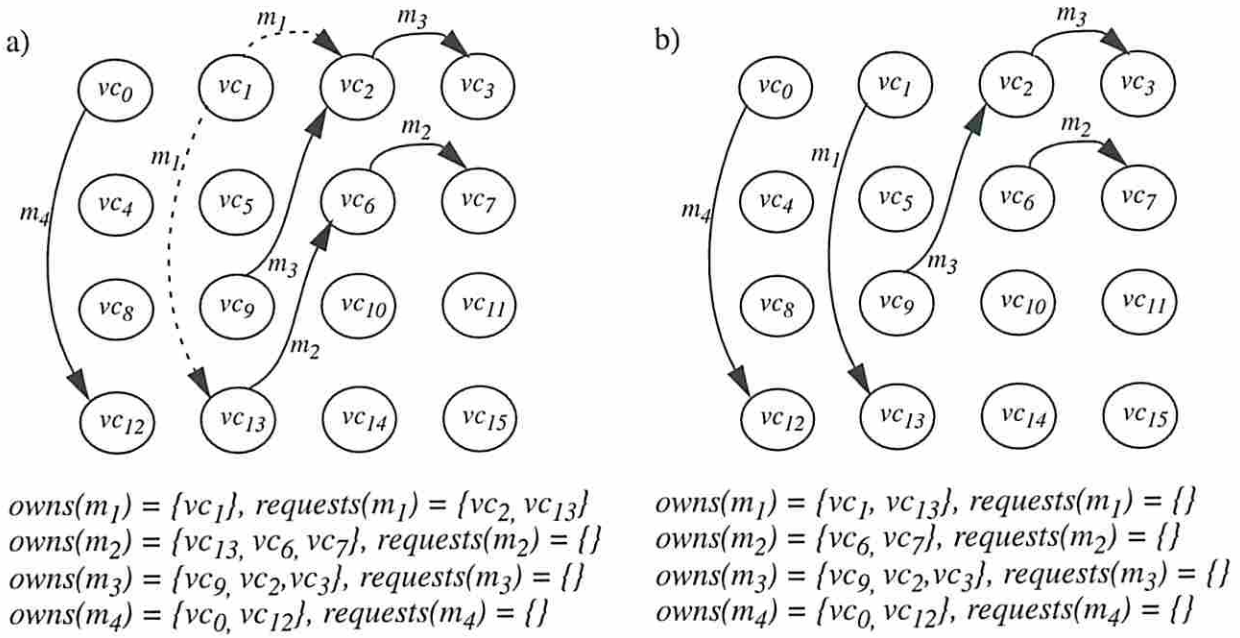

Figure 3 - a) CWG state when message m_3 acquires vc_3 and b) CWG state when message m_1 acquires vc_{13}.

In these illustrations, the VC labeling is done only to facilitate explanation and is not intended to convey information regarding the relative positions of VCs within the network. Also, to facilitate explanation, we label the outgoing arc(s) at each CWG vertex with the message which currently owns that VC. As described above, a path formed by a series of solid arcs with the

same label implies the temporal order in which the VCs represented by the vertices in the path were acquired and continues to be owned by a particular message. Given exclusive ownership of VC resources, there can be at most a single outgoing solid arc at any given node. At any vertex, the labels of incoming dashed arcs represent the group of messages that desire to use that VC at this instant in time. While we showed all of the vertices of the CWGs in Figures 1, 2, and 3, from now on we will show only those connected components of the CWGs useful for illustrative purposes.

2.2 Acyclic Paths, Cycles and Reachable Sets

Paths in CWGs are formed when two or more vertices are connected by a series of arcs. For example, in the CWG in Figure 2 a path exists from vc_{13} to vc_6, from vc_6 to vc_7, and from vc_{13} to vc_7 (through vc_6), etc. A path can be represented using an ordered list, and it may contain VCs owned by different messages, as in the path from vc_1 to vc_{12} in Figure 2. Here the path is represented by the ordered list (vc_1, vc_2, vc_3, vc_0, vc_{12}) and contains VCs owned by messages m_1, m_3 and m_4. When all of the vertices in a path are unique, an *acyclic path* is said to exist. All of the paths in Figures 1, 2 and 3 are acyclic.

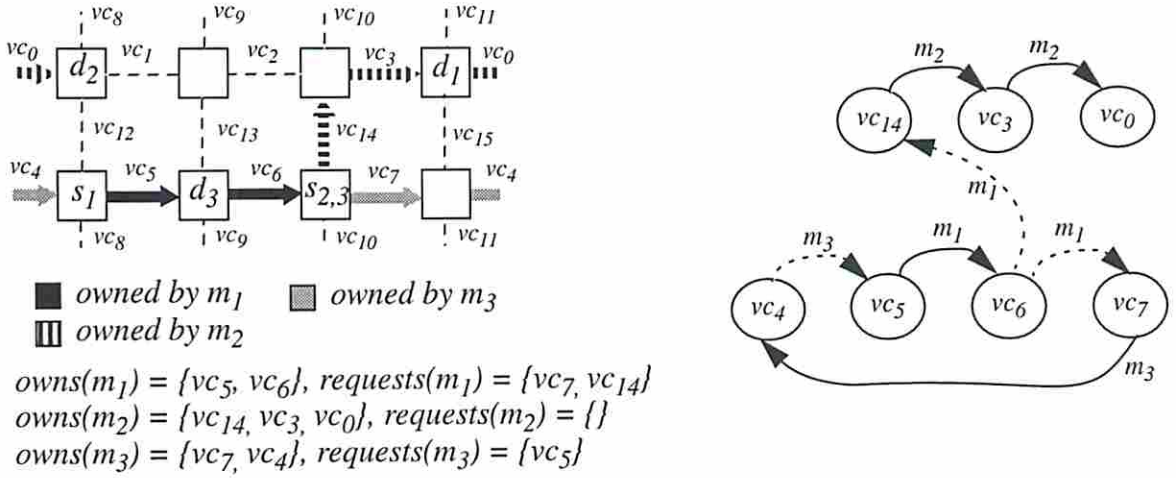

Figure 4 - Network and CWG state for messages m_1, m_2, and m_3 being routed within a *2x4* unidirectional network. There is a cycle in the CWG.

Figure 4 shows an example of a *cycle* within a CWG for a *2x4* unidirectional torus network with minimal adaptive routing. In general, a cycle in a CWG can be viewed as being composed of an acyclic path and an additional edge from the end of the path to the beginning of the path. The only cycle in the CWG depicted in Figure 4 consists of the set of vertices {vc_4, vc_5, vc_6, vc_7} and can be viewed as being composed of the acyclic path (vc_4, vc_5, vc_6, vc_7) and the edge (vc_7, vc_4). Alternatively, it can be viewed as being composed of the acyclic path (vc_5, vc_6, vc_7, vc_4) and the edge (vc_4, vc_5), among other variations. All of these variations represent a *unique cycle* in that they all refer to the same set of vertices and edges. Note that another unique cycle involving the set of vertices {vc_4, vc_5, vc_6, vc_7} can exist if, for instance, edges (vc_7, vc_6), (vc_6, vc_5), (vc_5, vc_4), and (vc_4, vc_7) also exist in the CWG. However, since there can be at most a single edge *in a given direction*

between any two vertices, we can refer to a unique cycle by an ordered list of vertices.

A *reachable set* of a vertex in a CWG is the set of vertices comprising all acyclic paths and cycles starting from that vertex. In Figure 4, the reachable set for vertex vc_{14} is $\{vc_3, vc_0\}$; for vc_3 is $\{vc_0\}$; for vc_0 is $\{\}$; and for vertices vc_4, vc_5, vc_6 and vc_7 is $\{vc_4, vc_5, vc_6, vc_7, vc_{14}, vc_3, vc_0\}$.

2.3 Deadlocks and Cyclic Non-Deadlocks

Figure 5 shows five messages being routed within a *4*-ary *2*-cube network with one VC per physical channel and which allows minimal adaptive routing. Here, message m_1 has acquired vc_0 and vc_1, message m_2 has acquired vc_2 and vc_3, message m_3 has acquired vc_4 and vc_5, message m_4 has acquired vc_6 and vc_7, and message m_5 has acquired vc_8, vc_9 and vc_{10}. While message m_5 has acquired all of the VCs needed to reach its destination, each of the other four messages are waiting to acquire an additional VC in order to reach their respective destinations. In the accompanying CWG, there is a cycle containing the set of vertices $\{vc_1, vc_3, vc_5, vc_7\}$. The set of vertices in this cycle has the property that the reachable set of each of its members is the set itself. A set of vertices which has this property is referred to as a *knot*. A network whose CWG contains a knot is said to contain a deadlock because none of the messages which own the vertices in the knot are able to advance. So we will use the terms knot and deadlock interchangeably. In a deadlocked state, For example, each of the messages m_1, m_2, m_3, and m_4 in Figure 5 will wait for one of the others in the group to release a required VC, thus waiting indefinitely assuming that this deadlock is not properly resolved. Messages not directly involved in a knot, but whose progress also depends upon the resolution of the deadlock are discussed in the next subsection.

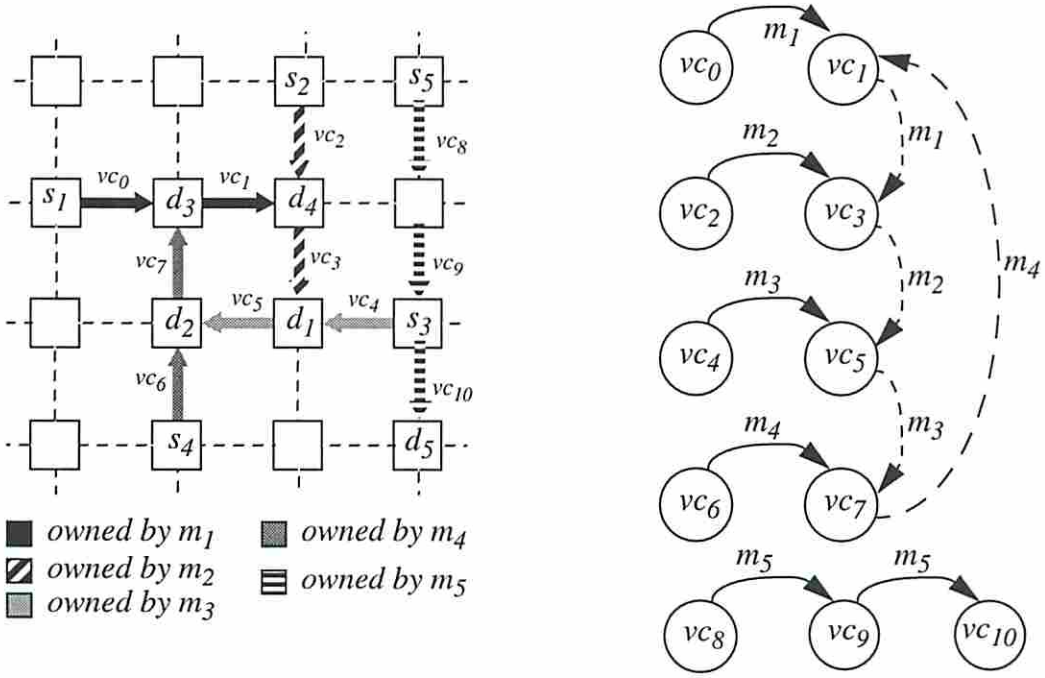

Figure 5 - Network and CWG state for messages m_1...m_5 being routed within a *4*-ary *2*-cube network with one VC per physical channel. There is a cycle in the CWG which constitutes a knot (single-cycle deadlock).

From the description of a knot, it is implied that a cycle is necessary for a knot since a cycle must exist in order for a vertex to be in its reachable set. However, a cycle is not sufficient for a knot or deadlock [8]. For example, the CWG in Figure 4 contains a cycle with vertices $\{vc_4, vc_5, vc_6, vc_7\}$, and the reachable set of all of its members is $\{vc_4, vc_5, vc_6, vc_7, vc_{14}, vc_3, vc_0\}$. However, notice that this set contains members which have different reachable sets than the set itself (i.e., vc_{14}, vc_3, and vc_0 have different reachable sets), thus violating the requirements for a knot. Such network configurations which contain one or more unique cycles, but no knot (deadlock) are said to contain what are referred to as *cyclic non-deadlocks*. Although cyclic non-deadlocks do not cause messages to block indefinitely, the "drainage dependencies" caused by large cyclic non-deadlocks can lead to substantial performance degradation [7, 9, 10]. Both deadlock avoidance and recovery based routing algorithms are susceptible to the formation of cyclic non-deadlocks.

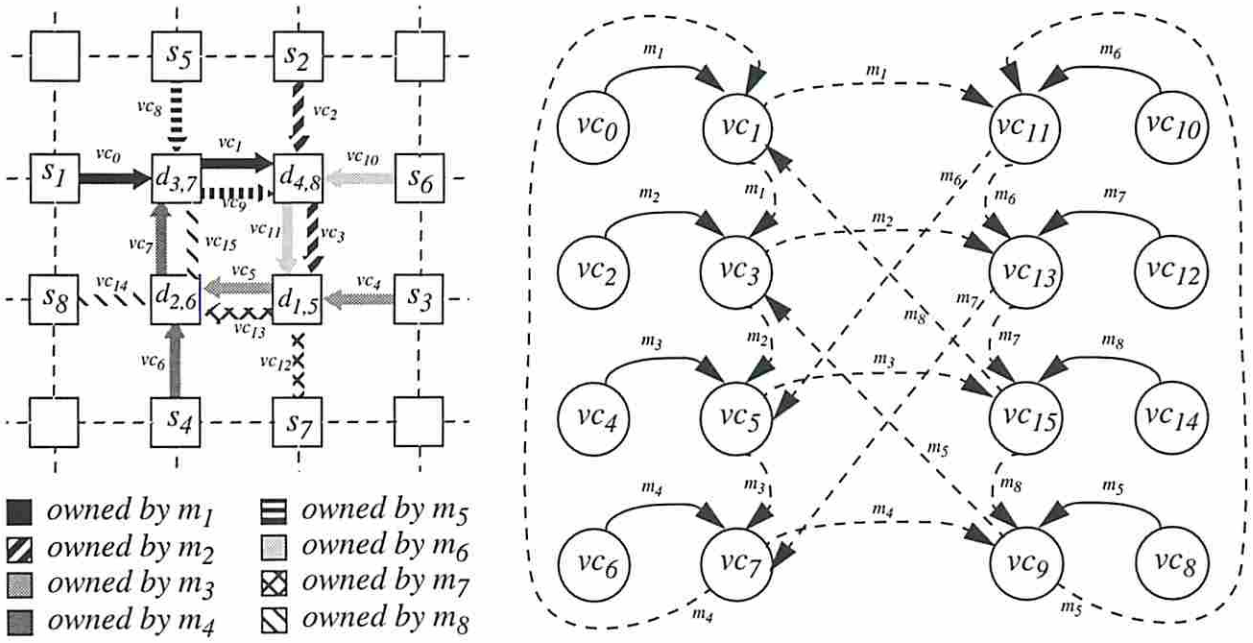

Figure 6 - Network and CWG state for messages m_1...m_8 being routed within a *4*-ary *2*-cube network with two VCs per physical channel. There are 24 unique cycles in the CWG which constitutes a knot (multi-cycle deadlock).

The deadlock depicted in Figure 5 is what is referred to as a "single-cycle deadlock" (the knot is composed of a single unique cycle). The number of unique cycles in a knot, referred to as *knot cycle density*, is a measure of deadlock complexity. All single-cycle deadlocks have a knot cycle density of one. The *deadlock set* represents the set of messages which own the VCs represented by the vertices in the knot (messages m_1, m_2, m_3, and m_4 in this example). Once a deadlock forms, it must be resolved by either removing [1, 2] or routing using some additional resources [3, 4] one of the messages in the deadlock set. The *resource set* of a deadlock, also used for characterization, represents the set of all resources owned by members of the deadlock set (the set $\{vc_0, vc_1, vc_2, vc_3, vc_4, vc_5, vc_6, vc_7\}$ in this example).

Figure 6 presents a more complex example of a knot---the reachable set for each of the nodes in the set $\{vc_1, vc_3, vc_5, vc_7, vc_9, vc_{11}, vc_{13}, vc_{15}\}$ is the set itself. There are 24 unique cycles formed by the vertices in the knot, which is referred to as a *multi-cycle deadlock*. The deadlock set for this knot is $\{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$ and its resource set is $\{vc_0...vc_{15}\}$.

2.4 Deadlock Dependent Messages

In addition to those messages in the knot, an unresolved deadlock may also force other messages not in the deadlock set to wait, indefinitely in some cases. Such messages are referred to as *deadlock dependent messages*. Figure 7 shows an example of a single cycle deadlock (knot $\{vc_0, vc_1, vc_2, vc_3\}$), with deadlock set $\{m_1, m_2\}$ and resource set $\{vc_0, vc_1, vc_2, vc_3\}$. Here, message m_3 and m_4, although not in the knot, must wait until the deadlock is resolved. Also, the progress of message m_6 is affected by blocking caused by the deadlock. These messages are distinguished from those properly in the deadlock since removing these deadlock dependent messages will not resolve the deadlock.

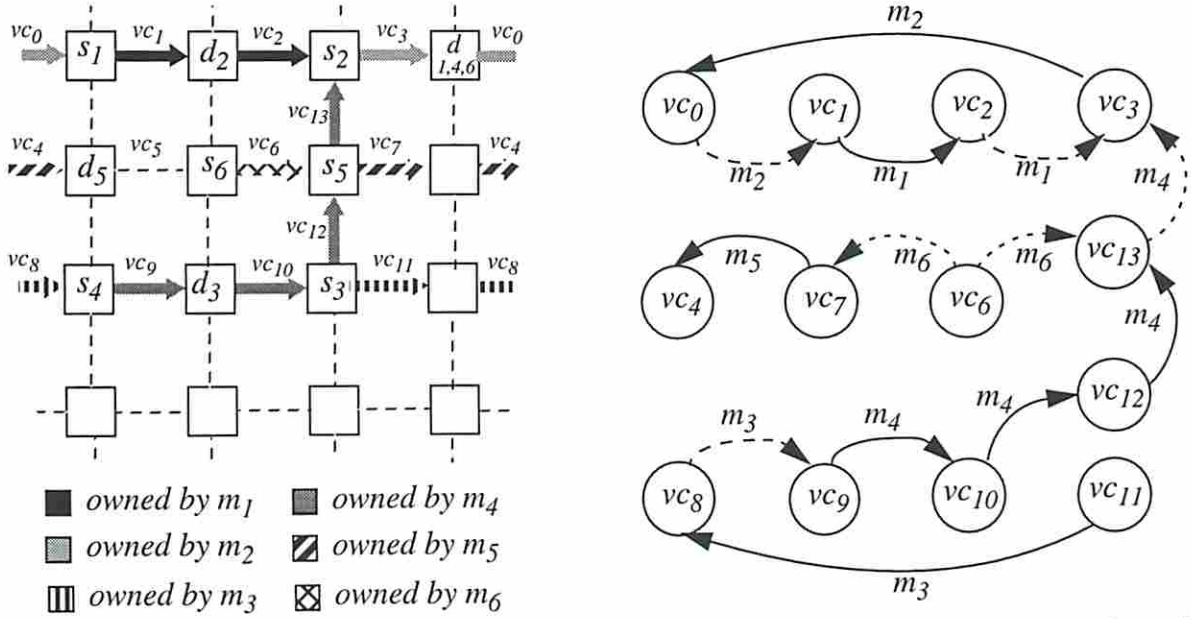

Figure 7 - Network and CWG state for messages m_1...m_4 being routed within a 4-ary 2-cube unidirectional network. There is a knot (deadlock), and message m_4 is fully directly deadlock dependent, message m_3 is fully indirectly deadlock dependent, and message m_6 is partially deadlock dependent.

Deadlock dependent messages can be further categorize into *directly* and *indirectly* deadlock dependent as well as *partially* and *fully* deadlock dependent. In the example shown in Figure 7, message m_4 is fully and directly deadlock dependent as each and every member of *requests*(m_4) is owned by a message in the deadlock set. Message m_3 is fully and indirectly deadlock dependent because at least some members of *requests*(m_3) are owned by other fully directly or fully indirectly deadlock dependent messages and all remaining members are owned by messages in the deadlock set. Both fully directly and fully indirectly deadlock dependent

messages can be referred to simply as "fully deadlock dependent messages." Fully deadlock dependent messages do not necessarily have to be dependent upon a single deadlock. Figure 8 presents an example of a *3D*-torus network with unidirectional channels and a single VC which contains two distinct deadlocks: knots $\{vc_0, vc_1, vc_2, vc_3\}$ and $\{vc_4, vc_5, vc_6, vc_7\}$. Here, message m_5 is fully directly deadlock dependent since all vertices in *requests*(m_5) are owned by members of the deadlock sets, although different deadlock sets. Although both deadlocks need to be resolved, the resolution of only one of the two will allow m_5 to resume. This example also points out that multiple deadlocks, each requiring an independent resolution, may exist simultaneously within a network.

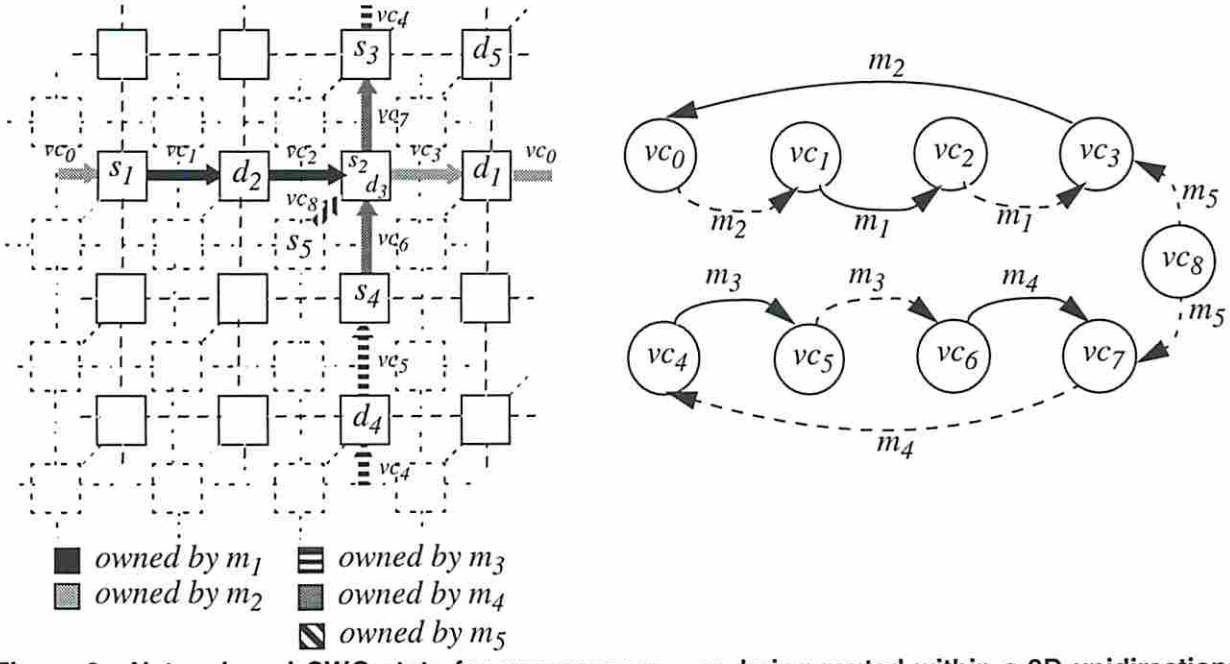

Figure 8 - Network and CWG state for messages m_1...m_5 being routed within a 3D unidirectional network. There are two independent knots (deadlocks) and message m_5 is fully directly dependent upon the two deadlocks.

Partially deadlock dependent messages are those messages whose *requests* sets contain VCs which are owned by non-blocked and/or blocked but non-fully deadlock dependent messages. These messages may resume even if the deadlock is not resolved. Message m_6 in Figure 7 is an example of a partially dependent message; it may resume when message m_5 relinquishes vc_7 despite the deadlock remaining unresolved.

Classifying dependent messages as such provides a basis for evaluating the accuracy of deadlock detection and recovery mechanisms. It also allows us to precisely describe and differentiate between previous notions of deadlock and ours. For instance, "deadlocked configurations" in [11] encompass not only messages in deadlock sets, but also all types of deadlock dependent messages and non-blocked messages as well. "Canonical deadlocked configurations" in [11] include both messages in deadlock sets and fully deadlock dependent messages but excludes partially deadlock dependent and non-blocked messages. Also, neither of these definitions of deadlocked configurations distinguish between a single and multiple

deadlocks, and more importantly, they do not distinguish between the causes and consequences of deadlock. In contrast, our model allows identification of each independent deadlock and distinguishes between causes (messages participating in knots) and consequences (indefinitely postponed dependent messages) of deadlock, thus enabling us to identify more fundamental components (knots) which require resolution for deadlock recovery.

In measuring the total impact of a deadlock, not only should the resources held by the deadlock set be taken into consideration, but also the resources held by deadlock dependent messages. We use the notion of *extended resource sets* when describing all of the resources indefinitely occupied by both members of the deadlock set and fully deadlock dependent messages. When a deadlock is allowed to persist, the number of deadlock dependent messages will grow to eventually encompass all of the messages within the network, and the extended resource set of the deadlock will grow to include most, if not all channel resources in the network. Therefore, the number of deadlock dependent messages and the size of extended resource sets are time-dependent, just as the canonical deadlocked configurations in [11].

2.5 Fault-Dependent Messages

While unresolved deadlocks can lead to permanent blocking of messages, there are non-deadlock related occurrences which can also cause this type of permanent blocking. Specifically, faults which disconnect the routing function can cause messages to block indefinitely and can eventually lead to all messages in a network becoming permanently blocked. Figure 9 shows four messages being routed within a *4x2* unidirectional network with one VC per physical channel and which allows minimal adaptive routing. In this example, the link for vc_2 has failed causing the routing function to be disconnected (i.e., at the node marked 'd_4', the routing function cannot supply a usable VC for message m_1 which is destined for node d_1). Despite there not being a knot in the CWG, messages m_1...m_4 will wait indefinitely or until the failed link is functioning once again.

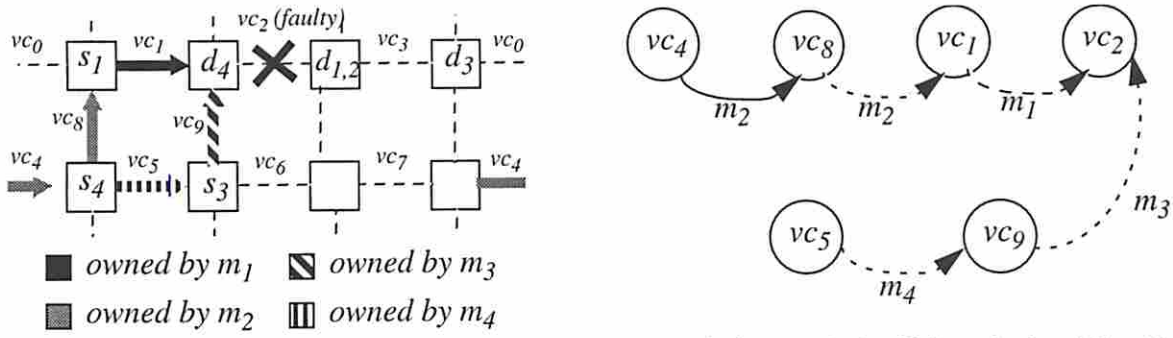

Figure 9 - Network and CWG state for messages m_1...m_4 being routed within a *2x4* unidirectional network containing a faulty link (vc_2). Messages m_1 and m_3 are fully directly fault dependent while messages m_2 and m_4 are fully indirectly fault dependent.

Allowing faults such as the one in this example to persist may result in the eventual blocking of all messages in the network---the same consequence as allowing a deadlock to remain unresolved. Despite this similarity in consequence, we do not consider this scenario to be deadlocked since 1) it does not meet the necessary conditions for deadlock as there are no cyclic resource dependencies and 2) it requires different means of resolution (i.e., repair of faulty link) than can be used for

deadlock resolution (i.e., removal of a deadlock set message). Note that both deadlock-avoidance and deadlock-recovery based routing algorithms are susceptible to this type of fault-related permanent message blocking. Nevertheless, we will assume networks with connected routing functions for the formal model of deadlock presented in Section 3.

Messages which are forced to wait (possibly indefinitely) due to faults are referred to as *fault-dependent messages*. Fault-dependent messages can be further categorized based on whether they are waiting only on faulty resources (*fully directly fault dependent*) or are waiting only on resources owned by other messages which are fully directly or indirectly fault dependent (*fully indirectly fault dependent*). In the example in Figure 9, messages m_1 and m_3 are fully directly fault dependent while messages m_2 and m_4 are fully indirectly fault dependent. As with partially deadlock dependent messages, *partially fault dependent messages* are those messages whose *requests* sets contain VCs which are owned by non-fault dependent messages. These messages may resume despite persistence of the fault. Various types of message blocking behavior and their properties are summarized in Table 1.

3 - Formal Framework

Here we formalize the intuitive model of deadlock and other message blocking behavior introduced in the previous section. Since we are interested in allowing deadlocks to form and not in preventing them, our formal framework has been generalized to support maximally flexible routing algorithms---those which allow unrestricted routing in any dimension and on any available VC at intermediate nodes. Therefore, more restrictive algorithms which also allow deadlocks (i.e., those that limit the number of misroutes, etc.) are included in our framework as well.

3.1 - The Formal Model of Deadlock

The following are some important assumptions we make in order to define deadlocks:

Assumption 1:
A node can generate messages destined for any other node at any rate, and the messages may be of any length greater than one flit.

Assumption 2:
A message arriving at its destination is eventually consumed.

Assumption 3:
Flits from two different messages may not be in the same queue at the same time. Therefore, a tail flit of a message must leave the queue before the header flit of another message enters the queue. An extension of which allows multiple messages to share a queue simultaneously is presented in Section 3.2.1.

properties of message m_i / *blocking behavior of message m_i*	*m_i indefinitely blocked*	*all members of requests(m_i) owned by msgs. in deadlock set [is faulty]*	*all members of requests(m_i) owned by fully deadlock [fault] dependent msgs.*	*only some members of requests(m_i) owned by fully deadlock [fault] dependent msgs.*	*m_i can occur in deadlock-avoidance based routing*	*m_i can occur in deadlock-recovery based routing*	***indefinite blocking in network resolved if m_i is removed***
in deadlock set	Yes	Yes	No	No	No	Yes	**Yes**
in cyclic non-deadlock	No	No	No	No	Yes	Yes	**n/a**
fully directly deadlock dependent	Yes	Yes	No	No	No	Yes	**No**
fully indirectly deadlock dependent	Yes	No	Yes	No	No	Yes	**No**
partially deadlock dependent	No	No	No	Yes	No	Yes	**No**
fully directly fault dependent	Yes	[Yes]	[No]	[No]	Yes	Yes	**No**
fully indirectly fault dependent	Yes	[No]	[Yes]	[No]	Yes	Yes	**No**
partially fault dependent	No	[No]	[No]	[Yes]	Yes	Yes	**No**

Table 1 - Summary of various types of message blocking behavior and their properties.

Assumption 4:
Once the queue of a particular virtual channel accepts the header flit of a message, it must accept all of the remaining flits of the message before accepting flits from any other message. This satisfies the "no preemption" condition necessary for deadlock.

Assumption 5:
A blocked message, while holding onto previously acquired VCs, remains waiting in the network until a VC becomes available. This satisfies the "resource wait-for" condition, and creates potential for the "circular wait" condition, both of which are also necessary for deadlock.

We use the following definitions to prove the necessary and sufficient conditions for deadlock within interconnection networks.

***Definition 1*:**
An *Interconnection Network I* is a strongly connected directed graph, $I = G(N, C)$, where the vertices N represent the set of router nodes and the edges C represent the set of channels that connect the routers. For a particular channel $c_i \in C$, $s_i, d_i \in N$ represent its source and destination nodes, respectively.

Definition 2:
A *routing function R* is of the form $R(N \times N) \rightarrow P(C)$ (where P denotes the Power Set). That is, for a given node $n_i \in N$, and a destination node $n_d \in N$, $R(n_i, n_d)$, $n_i \neq n_d$ provides a set of alternative channels $\{c_1, c_2,...,c_r\}$, $c_i \in C$, $1 \leq i \leq r$ through which a message may be routed to its next hop in route to n_d. We have used a routing function of the form $R(N \times N) \rightarrow P(C)$ instead of $R(C \times N) \rightarrow P(C)$ because it better suits our purpose here. Also, the selection function [7] is not relevant in the present context, and therefore, we will simply assume a random selection function.

Definition 3:
The set of *messages* currently being routed through network I is defined as M. We denote the source and destination nodes of message $m_i \in M$ using $source(m_i) \in N$ and $dest(m_i) \in N$, respectively.

Definition 4:
We define the function $status(c_i) = free \vee busy$ to indicate the availability of a channel $c_i \in C$ for allocation to a message.

Definition 5:
For each $m_i \in M$, we define a set $owns(m_i)$ representing the set of channels that message m_i has currently reserved (hence "owns") for its route through I. So for a given $owns(m_i)=\{c_i,..., c_n\}$, $n \geq 1$, $status(c_i) = busy$ for $i=1...n$. We also define the inverse function $ownedby(c_i)=m_i$. The

value of *ownedby*(c_i) is undefined when $status(c_i) = free$.

Definition 6:

For each set *owns*(m_i), $m_i \in M$, we define a function *first*(*owns*(m_i))=c_{first}, where c_{first} represents the least recently acquired of the channels currently owned by message m_i. We define the function *last*(*owns*(m_i))=c_{last}, where c_{last} represents the most recently acquired of the channels currently owned by m_i. To help establish the partial order in which message m_i has reserved the channels in *owns*(m_i), we define the function $next(C) \rightarrow C$, where *next*(c_j)=c_k implies that c_k was reserved by m_i immediately after c_j was reserved by m_i. The value of *next*(c_j) is undefined when c_j=*last*(*owns*(m_i)).

Definition 7:

For each $m_i \in M$, we define the set *requests*(m_i)={c_1, c_2,..., c_r} representing the set of channels requested by message m_i when it becomes blocked, any one of which allows m_i to proceed. When m_i is not blocked, |*requests*(m_i)|= 0. When m_i is blocked, *R*(d_k, *dest*(m_i))=*requests*(m_i) where *last*(*owns*(m_i))=c_k. Note that $dest(m_i) \neq d_j \ \forall \ c_j \in owns(m_i)$, which is to say that m_i does not already own the channel which is connected to its destination, thus implying that *R*(*dest*(m_i),*dest*(m_i))={}.

Definition 8:

A *Deadlock* in an interconnection network I is a condition in which there exists a set of messages $D=\{m_1,..., m_n\}$, $m_1 ... m_n \in M$, $n \geq 1$ which holds the relationship that $\forall \ m_i \in D$, (1) |*requests*(m_i)| > 0, (2) $\forall \ c_j \in requests(m_i)$, $ownedby(cj) = m_k$ for some $m_k \in D$, and (3) $\forall$ *owns*(m_i), $\exists \ c_l \in owns(m_i)$ such that $c_l \in requests(m_p)$ for some $m_p \in D$. This is to say that a deadlock is formed when a set of messages meets *all three* of the following conditions: (1) all of the messages in the set are blocked, (2) each channel waited upon by every message in the set is owned by messages also in the set, and (3) every message in the set owns at least one channel which is waited upon by a member of the set. By allowing n to be equal to one, and by not restricting m_i, m_k, and m_p to be distinct, we allow deadlocks involving a single message as well. D is the *deadlock set* for this deadlock. The *resource set* for this deadlock is $\bigcup owns(m_i) \ \forall \ m_i \in D$.

Definition 9:

A *fully directly deadlock dependent message* is one for which |*requests*(m_i)| > 0 and $\forall \ c_j \in requests(m_i)$, *ownedby*($c_j$)=$m_k$ where $m_i \in M$ and $m_k \in D$ for some deadlock D. This is to say that m_i is fully directly deadlock dependent if it is blocked and *all* of the channels it is waiting-for are owned by members of a deadlock set.

Definition 10:

A *fully indirectly deadlock dependent message* $m_i \in M$ is one for which (1) |*requests*(m_i)| > 0

and (2) there is at least one $c_j \in requests(m_i)$, $ownedby(c_j)=m_k$ where m_k is either fully directly or indirectly deadlock dependent and (3) for all remaining $c_j \in requests(m_i)$, $ownedby(c_j)=m_k$ where $m_k \in D$ for some deadlock D. This is to say that m_i is fully indirectly deadlock dependent if it is blocked, *at least one* of the channels it is waiting for is owned by a fully directly or indirectly deadlock dependent message, and *all* of the remaining channels it is waiting for (if any) are owned by members of a deadlock set.

Definition 11:
A *partially deadlock dependent message* $m_i \in M$ is one for which (1) $|requests(m_i)| > 0$ and (2) there is at least one $c_j \in requests(m_i)$, $ownedby(c_j)=m_k$ where $m_k \in D$ or m_k is either fully directly or indirectly deadlock dependent and (3) there is at least one $c_j \in requests(m_i)$, $ownedby(c_j)=m_k$ where $m_k \notin D$ for some deadlock D and m_k is neither fully directly nor indirectly deadlock dependent. This is to say that m_i is partially deadlock dependent if (1) it is blocked, (2) *at least one* of the channels it is waiting for is owned by a fully directly or indirectly deadlock dependent message or a member of the deadlock set, and (3) *at least one* of the channels it is waiting for is *not* owned by a fully directly or indirectly deadlock dependent message or a member of the deadlock set.

Definition 12:
The *extended resource set* of a deadlock D is the set $\bigcup owns(m_i)$ where $m_i \in D$ or m_i is either fully directly or indirectly deadlock dependent.

Definition 13:
A *Channel Wait-for Graph* is a directed graph $CWG=(V_c, E_c)$ representing a particular state of resource ownership and requests within I. The vertices V_c represent the channels C of I. In any given state of the CWG, $\exists$ an edge $(c_i, c_j) \in E_c$ if $\exists$ $m_k \in M$ such that either (1) $c_i, c_j \in owns(m_k)$ and $next(c_i)=c_j$, or (2) $c_i=last(owns(m_k))$ and $c_j \in reqeusts(m_k)$. Case (1) refers to the solid arcs and case (2) refers to the dashed arcs of the CWG illustrations in the previous section. Also, there does not exist any c_i such that $(c_i, c_i) \in E_c$, thus implying that no channel has the same network node as both its source and destination, and therefore a message is unable to request a channel immediately after acquiring that channel. By this we eliminate *1*-cycles in the CWG.

Definition 14:
An *Acyclic Path* in a CWG is an ordered list of unique vertices $(c_1, c_2,...,c_{n-1}, c_n)$, $c_1 \ldots c_n \in V_c$ such that there are edges $(c_1, c_2), (c_2, c_3), \ldots, (c_{n-1}, c_n) \in E_c$.

Definition 15:
A *Cycle* containing vertex c_i in a CWG is an ordered list of vertices $(c_i, c_{i+1},...,c_n, c_i)$,

$c_i \ldots c_n \in V_c$ such that there is an acyclic path from c_i to c_n and an edge $(c_n, c_i) \in E_c$.

Definition 16:
A *reachable set* for a vertex $c_i \in V_c$ is the set of vertices $R_s \in P(V_c)$ such that $c_j \in R_s$ if there is an acyclic path from c_i to c_j and $c_i \in R_s$ if there is a cycle containing c_i.

Definition 17:
A *knot* in the CWG is a non-empty set $K \in P(V_c)$ such that $\forall\ c_i \in K$, the reachable set of c_i is exactly K. The *cycle density* of this knot (referred to as *knot cycle density*) is $|Z_K|$ where Z_K is the set of all cycles involving vertices $c_i \in K$.

Definition 18:
We define the legal state transition of a CWG to reflect the *allocation* of a channel c_j, *status(c_j)=free* to message m_i as follows:

- if *|owns(m_i)|=0*, *owns(m_i)* becomes *{c_j}* where $c_j \in R(source(m_i), dest(m_i))$, and *status($c_j$)* becomes *busy.*
- otherwise, *owns(m_i)* becomes $owns(m_i) \cup \{c_j\}$ where $c_j \in R(d_k, dest(m_i))$ and *last(owns(m_i))=c_k*, E_c becomes $E_c \cup \{(last(owns(m_i)), c_j)\}$, and *status($c_j$)* becomes *busy.*

In the CWG illustrations of the previous section, this refers to simply adding a solid arc from the vertex representing the last channel owned by m_i to the vertex representing the newly acquired channel c_j.

Definition 19:
We define the legal state transition of a CWG to reflect the *request* for use of channels by a message m_i where *status(c_j)=busy* $\forall\ c_j \in R(d_k, dest(m_i))$, *last(owns($m_i$))=$c_k$* as follows:

- E_c becomes $E_c \cup \{(c_k, c_j)\}\ \forall\ c_j \in R(d_k, dest(m_i))$ and *requests(c_j)* becomes *R(d_k, dest(m_i))* where *last(owns(m_i))=c_k.*

In the CWG illustrations, this corresponds to adding dashed arcs originating from the vertex representing the last channel owned by message m_i to every vertex which represents an alternate VC that m_i may use to continue.

Definition 20:
We define the legal state transition of a CWG to reflect the *resumption* of a blocked message m_i (*|requests(m_i)|>0*) as follows:

- E_c becomes E_c - *{(last(owns(m_i)), c_j)}* $\forall\ c_j \in requests(m_i)$, *requests($m_i$)* becomes *{}*, and we then allocate a new channel c_j to m_i according to *Definition 18.*

In the CWG illustrations, this corresponds to removing all of the dashed arcs originating from the vertex representing the last channel owned by a message m_i and then adding a solid arc from this

vertex to the vertex representing the newly acquired channel c_j.

Definition 21:
We define the legal state transition of a CWG to reflect the *release* of channel c_j=*first(owns(m_i))* by a message m_i as follows:

- if |*owns(m_i)*|>*1*, E_c becomes E_c - *{(first(owns(m_i)), next(first(owns(m_i))))}*, *owns(m_i)* becomes *owns(m_i) - {first(owns(m_i))}*, and *status(c_j)* becomes *free*.
- otherwise, *owns(m_i)* becomes *{}* and *status(c_j)* becomes *free*.

This reflects the fact that a channel is released after its use by a tail flit of a message, hence channels are acquired and released in the same order.

We now propose and prove the following theorems:

Theorem 1: *A deadlock with a deadlock set D exists in an interconnection network I iff there exists a knot K in the channel wait-for graph of I.*

Outline of proof:
-> A deadlock in *I* implies that there is a group of one or more blocked messages *D* where each message $m_i \in D$ owns at least one channel waited upon by some message $m_j \in D$. It then follows that for each m_i, there is an edge $(last(owns(m_j)), c_l) \in E_c$ where $c_l \in owns(m_i)$ (this is a request arc to a vertex which represents a VC owned by m_i). So for each m_i, there is a subset of *owns(m_i)* which is "reachable" (contains a path) from each and every member of *owns(m_j)*. We will refer to this set as *reachable(m_i, m_j)*. For a given m_i, there can be different *reachable(m_i, m_j)* sets (one for each distinct m_j), and we identify the largest of such sets as *max_reachable(m_i)*. Note that for any |*reachable(m_i, m_j)*| > |*reachable(m_i, m_k)*|, $reachable(m_i, m_k) \subset reachable(m_i, m_j)$. Also, a deadlock implies that for each $m_i \in D$, every member of *requests(m_i)* is owned by some $m_j \in D$, suggesting that all members of *max_reachable(m_j)* are reachable from all members of *owns(m_i)* for some $m_i \in D$. By transitive properties of paths, we can show that every member of *max_reachable(m_i)* is reachable from every member of *max_reachable(m_j)* and vice-versa for all $m_i, m_j \in D$. Therefore, the set $\bigcup maxreachable(m_i)$ $\forall$ $m_i \in D$ meets the requirements for, and thus constitutes, a knot.
<- Using *Definitions 18-21*, assume that the network *CWG* has reached some state where it contains a knot *K*. Also assume that there is no deadlock, thus implying that there is some series of reductions of the *CWG* using the transitions in *Definitions 18-21* to eliminate edges $(c_i, c_j) \in E_c$ where $c_i \in K$. The existence of such reductions suggest that there exists a path from $c_i \in K$ to some node $c_k \in V_c$ such that message m_l=*ownedby(c_k)* is not blocked (|*requests(m_l)*|=*0*). It then follows that there is no edge in E_c which originates from *last(owns(m_l))*. However, c_k is reachable from c_i, but c_i is not reachable from c_k, a contradiction of the properties of a knot.
❐

***Corollary 1.1**: All vertices in a knot are owned by members of a single deadlock set.*

This arises from the fact that $\bigcup maxreachable(m_i)\ \forall\ m_i \in D$ is exactly the same as a knot K, thus helping us establish a one-to-one correspondence between the messages in the deadlock and the knot. So we can alternatively define a *deadlock set* to be a set of messages $\{m_1,..., m_n\}$, $n \geq 1$ where for each m_i in the set, $\exists$ a channel represented by a vertex $c_j \in K$ where *ownedby*$(c_j)=m_i$.

***Theorem 2**: A deadlock with deadlock set D in network I can be resolved if one or more of the messages in D is removed from the network.*

Outline of proof:
<-Given a deadlock D, assume that we select a message $m_j \in D$ to be removed from the network. Message m_j releases all the VCs $c_l \in owns(m_j)$ one-by-one (according to *Definition 19)* so that *status*(c_l) becomes *free* $\forall\ c_l \in owns(m_j)$. If $|D|=1$, then none of the messages in the deadlock set exist after m_j releases its resources, and therefore the deadlock seizes to exist (resolved). If $|D|>1$, then by definition of deadlock, $\exists\ D_r \subseteq D$ such that $(requests(m_i) \cap owns(m_j)) \neq \varnothing\ \forall\ m_i \in D_r$. This is to say there is at least one message owned by m_j that is needed by another message in D (condition (3) of *Definition 8*). Therefore, having m_j release its resources will allow at least one $m_i \in D_r$ to continue, thereby resolving the deadlock.
❐

***Theorem 3**: Removing fully directly, fully indirectly, and partially deadlock dependent messages from the network is not sufficient to resolve a deadlock.*

Outline of proof:
-> Assume that a deadlock D can be resolved by having deadlock dependent messages removed from the network. This implies that $\exists$ some message $m_i \in M - D$ such that $owns(m_i) \cap requests(m_j) \neq \varnothing$ for some $m_j \in D$ and that having m_i release its resources enables m_j to continue, thus resolving the deadlock. However, by definition of a deadlock (condition (2) of *Definition 8*), $ownedby(c_l) = m_k$, $m_k \in D\ \forall\ c_l \in requests(m_j)\ \forall\ m_j \in D$. That is to say that every channel requested by every member of the deadlock set is owned by a member of the deadlock set. Therefore, $owns(m_i) \cap requests(m_j) = \varnothing\ \forall\ m_i \in M - D$ and $\forall\ m_j \in D$, a contradiction.
❐

What we have demonstrated here is that removing one of the messages in the deadlock set is sufficient for deadlock resolution. We have also shown that removing deadlock dependent messages is not sufficient for deadlock resolution. When a deadlock remains unresolved, the

messages which meet the requirements of the first two conditions of *Definition 8*, but not the third condition are also indefinitely delayed. However, we impose the third condition so as to restrict the deadlock set to only those messages which can be removed from the network to resolve deadlock, thus excluding dependent messages.

Next we present possible extensions to our formal model.

3.2 - Extensions to the Model

Here we present two possible extensions to the basic model presented above.

3.2.1 - Sharing of Queues

Our earlier assumptions restrict flits of different messages from being in a single queue at the same time. Since our framework allows deadlocks to form, this restriction is not necessary. We present here an extension to the theory presented above which allows multiple messages to simultaneously share a queue. Figure 10 shows the routers and their VC queues for a portion of a unidirectional ring network with two VCs per physical channel along with the corresponding CWG. In the networks, flits h_i, d_i, and t_i correspond to the header, data, and tail flits of message m_i. This figure shows that with restrictions on buffer sharing, the leading flits of message m_3 cannot enter the queue for vc_4 despite there being room to accommodate some of these flits, thus limiting the maximal utilization of resources.

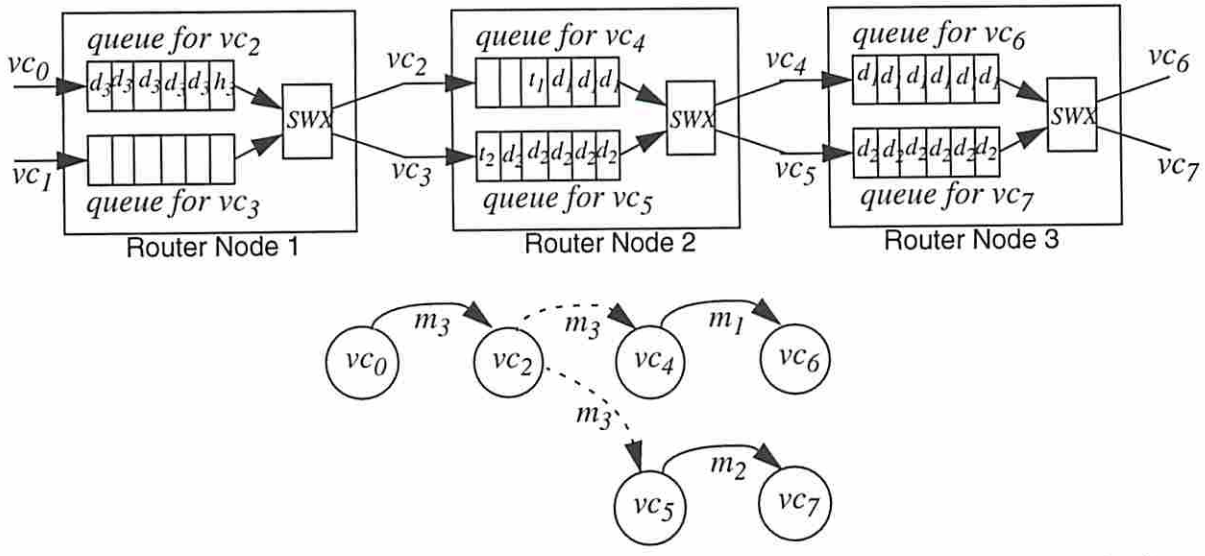

Figure 10 - Internal state of network routers and the corresponding CWG state when sharing of queues is not allowed.

Our model can be extended to allow queue sharing by replacing Assumption 3 with the following assumption:

- Flits of different messages may simultaneously occupy the same queue but cannot be interspersed. We designate the message whose flits are at the head of the queue as the

sole owner of the queue. Another message may "own" the queue only when the tail flit of the previous message has vacated the queue. Therefore, we allow physical sharing of resources, but maintain a logical model of "exclusive ownership", a necessary condition for deadlock.

When a message enters a queue which it does not own (given that the tail flit of another message still occupies this queue), it commits to using that VC and is no longer able to use any of the other alternative VCs supplied by the routing function. To reflect this in the CWG, our model requires that all of the request arcs (dashed arcs) be removed except the one request arc to the VC being committed to when the message was blocked immediately prior to entering the new queue or that a dashed arc be placed between the last channel owned and the channel committed to when the message was not blocked immediately prior to entering the new queue. This single dashed arc becomes a solid arc only when the message acquires ownership of the VC (when the tail flits of other messages in the queue have vacated the queue).

The following defines the legal transition for committing to a VC owned by another message:

Definition 22:
We define the legal state transition of a CWG to reflect the *committing* of a message m_i to a channel c_j, $ownedby(c_j) = m_k, m_k \neq m_i$ as follows:

- If m_i is blocked ($|requests(m_i)|>0$), E_c becomes E_c - $\{(last(owns(m_i)), c_l)\}$ $\forall$ $c_l \in requests(m_i), c_l \neq c_j$ and $requests(m_i)$ becomes $\{c_j\}$.
- Otherwise ($|requests(m_i)|=0$), E_c becomes $E_c \cup \{(last(owns(m_i)), c_j)\}$ and $requests(m_i)$ becomes $\{c_j\}$.

In the CWG illustrations, this corresponds to removing all of the dashed arcs originating from the vertex representing the last channel owned by a message m_i except the one dashed arc to the vertex representing the channel (c_j) to which the message commits when the message was previously blocked or simply adding a single dashed arc from the vertex representing the last channel owned by a message m_i to the vertex representing the channel c_j when the message was not blocked. The transition to reflect subsequent ownership of the channel c_j by message m_i follows *Definition 20.*

Figure 11 shows an example of allowing queues to be shared. Here, the leading flits of message m_3 have entered the queue for vc_4, thus committing to the use of vc_4. By "committing early" to using vc_4, m_3 is no longer able to use vc_5 (as reflected by the lack of a dashed arc to vc_5 in the CWG) even though the queue for vc_5 may become available for ownership prior to vc_4. Empirical results suggest that reducing "fanout" as done here by committing to VCs early increases the probability of deadlock formation [10].

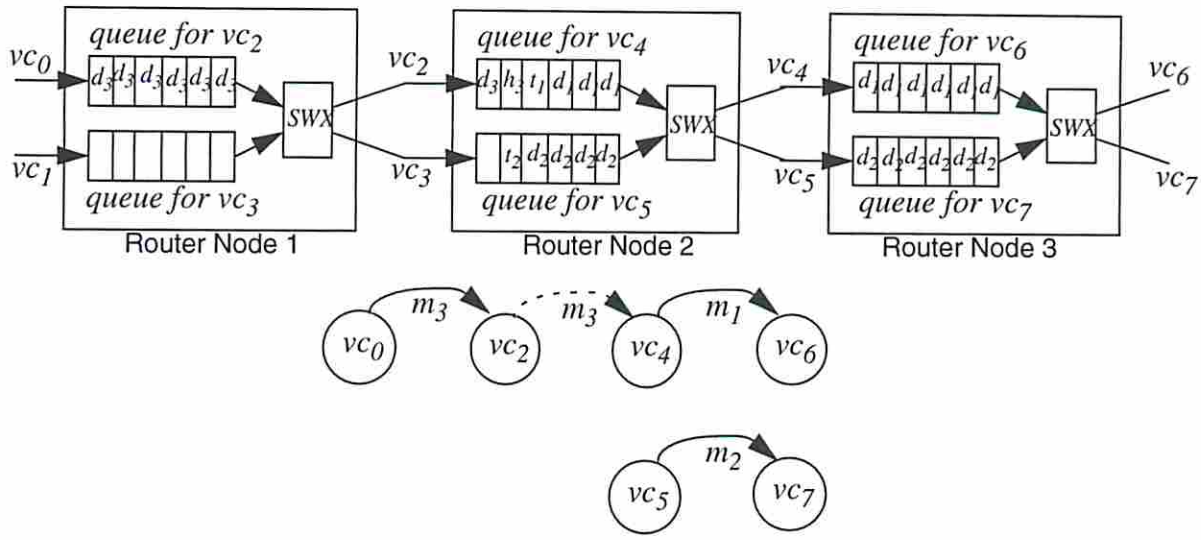

Figure 11 - Internal state of network routers and the corresponding CWG state when sharing of queues is allowed.

3.2.2 - Inclusion of Injection and Reception Channels

The resource dependency model using CWGs we have presented takes into account only those messages which have "entered" the network, and therefore have acquired at least a single resource (i.e., $|owns(m_i)| > 0 \; \forall \; m_i \in M$). So it does not take into account those messages which are waiting in the injection queues (from the processor) to acquire their first VCs. When a deadlock occurs, messages waiting in the injection queues may be impacted similar to other dependent messages. To properly account for this, our model can be expanded to include messages in the injection queues as well.

Our model can be easily expanded by including all of the injection channels in the set of vertices V_c in the *CWG*. Ownership rules for the injection channels remain the same as those for the VCs. The representation of wait-for relationships between messages in the injection channels and VCs owned by other messages is similar as well. An example is presented in Figure 12, which shows messages m_4 and m_5 in injection channels ic_1 and ic_2 (respectively) waiting for channels owned by messages m_1 and m_2. If vertices $vc_4 ... vc_7$ in the CWG belong to a knot, then messages m_4 and m_5 would be considered deadlock dependent messages. So extending our model to include injection channels may be useful for more detailed deadlock characterization. Dependence on reception channels (to processor) can also be modeled in a similar fashion. Note that messages in injection and reception channels cannot participate in a deadlock set unless injection and reception channels are dependent upon each other (i.e., share buffer resources). However, we assume that a message reaching its destination gets consumed eventually. Therefore, removing messages in injection channels will not help resolve deadlock.

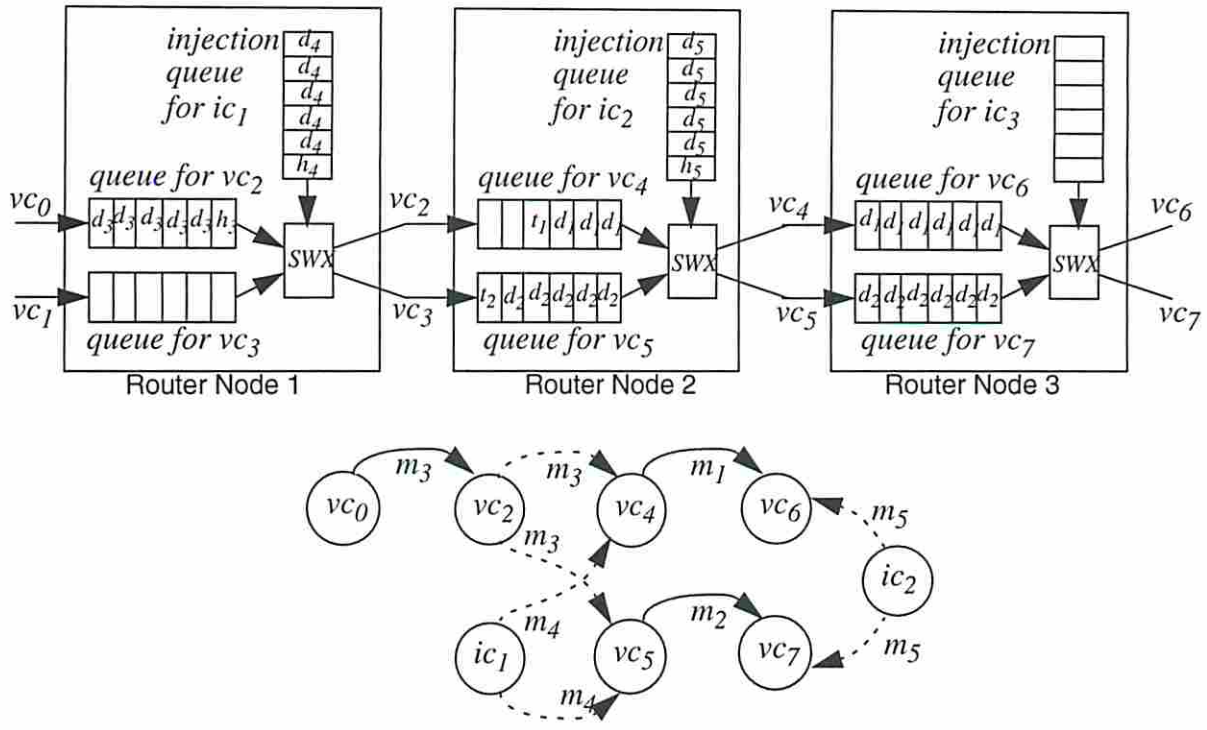

Figure 12 - Internal state of network routers and the corresponding CWG state when including injection channels in the resource model.

4 - Implementation of Deadlock Detection

Our approach for deadlock detection is based on the formal framework presented in Section 3. Deadlock detection has been implemented in a flit-level interconnection network simulator called *FlexSim* (an extension of *FlitSim 2.0*). Our implementation of deadlock detection involves maintaining a CWG, detecting cycles within the CWG, and identifying groups of messages and cycles which form knots. Each of these areas is discussed in the following subsections.

4.1 - Building and Maintaining the Channel Wait-for Graph

We dynamically build and maintain a CWG reflecting the resource allocations and requests of an on-going network simulation. The CWG is implemented using an array of linked lists. Each array element represents a message (m_i) within the network, and each linked list represents the set of VCs owned by a particular message (*owns*(m_i)), with additional links to desired VCs for blocked messages (*requests*(m_i)). Figure 13 provides an overview of data structures used to maintain the CWG, cycle list, and deadlock list. The state of the data structures in this figure corresponds to the example in Figure 7.

When a VC is allocated to a message, the "VCNode" data structure for the VC is appended to the linked list for that message and the VCNode is annotated with its new owner. Similarly, when a VC is released, the VCNode is removed from the beginning of the linked list and the ownership information is removed accordingly. When a message blocks, special request links are placed

from the end of the linked list to the VCNodes representing each of the alternate VCs the message may use to continue routing. When a message resumes, these request links are removed and replaced with a link to the VCNode representing the newly acquired VC. These CWG maintenance operations are formally described in *Definitions 17-20* presented in the previous section. The direct links from the elements of the array of messages to the last VCNodes of each linked list are used as an optimization for efficient traversal of the linked lists. The use of the cycle list and the deadlock list data structures are discussed in the subsequent sections.

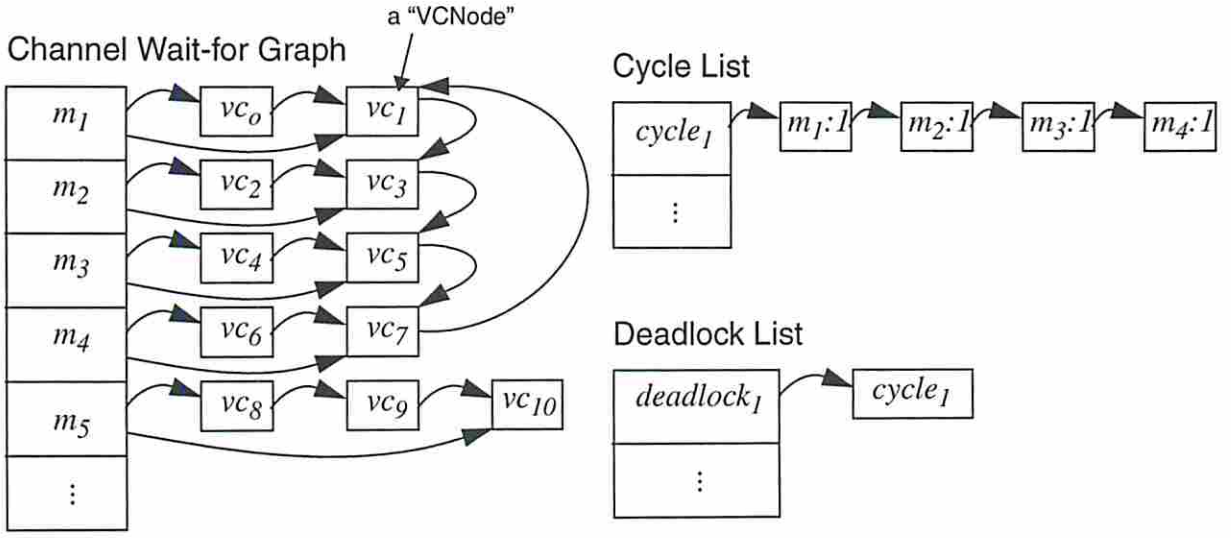

Figure 13 - The CWG, Cycle List, and Deadlock List data structures. The state of the data structures reflects the example in Figure 7.

4.2 - Detecting Cycles

We detect all cycles within a CWG in order to facilitate the detection of deadlocks. Cycles in the CWG are detected by performing depth-first search of the graph with backtracking (starting with each linked list in linear order). Each node visited is marked as such, and therefore visiting a node which has previously been marked as having been visited indicates the presence of a cycle. Annotations indicating visitation are removed upon backtracking. To avoid detection of each unique cycle more than once, the partial order of message IDs is used to prune the search (i.e., restrict visiting a VCnode owned by a message with a lower message ID). For example, the VCNodes in the CWG in Figure 13 are visited in the following order where '*' indicates cycle detection, *b* indicates a backtracking step, and *p* indicates a pruning measure: $[m_1]$, vc_0, vc_1, vc_3, vc_5, vc_7, vc_1, *, *b*, *b*, *b*, *b*, *b*, *b*, $[m_2]$, vc_2, vc_3, vc_5, vc_7, *p*, *b*, *b*, *b*, *b*, $[m_3]$, vc_4, vc_5, vc_7, *p*, *b*, *b*, *b*, $[m_4]$, vc_6, vc_7, *p*, *b*, *b*, $[m_5]$, vc_8, vc_9, vc_{10}.

When a unique cycle is detected, it is placed in the cycle list as shown in Figure 13. A list of messages involved and their corresponding "branches" are sufficient to uniquely identify a cycle (i.e., m_1:1 in the figure indicates the first branch of message m_1, representing the first of possibly many VCs the message is waiting for). When previously detected cycles are again detected during a different run of the algorithm, their entries in the cycle list are updated to indicate the time of their most recent detection. Our cycle detection algorithm guarantees detection of each unique cycle in

the CWG during each invocation. Therefore, those cycles which were not detected during the most recent invocation of the algorithm are removed from the cycle list. Since a very large number of cycles can exist within a congested network, we use advanced indexing functions and search techniques to compare cycles in determining their uniqueness. Formally, the cycle detection algorithm generates a set $Z=\{\}$ *or* $\{Z_1, Z_2, ..., Z_n\}$, $n \geq 1$, where each Z_i represents a cycle which is uniquely represented by $\{(m_1, idx_1), (m_2, idx_2), ..., (m_k, idx_k)\}$, $k \geq 1$, and where each (m_i, idx_i) represents a message and a "branch index" as described above.

4.3 - Detecting Deadlocks

Once the cycle detection algorithm identifies all cycles within the CWG, the deadlock detection algorithm is invoked to identify groups of these cycles which form a knot (as we have shown a knot to be necessary and sufficient for a deadlock in *Theorem 1*). A number of heuristics are used to make this possible. First, a group of blocked messages whose request arcs are all involved in cycles is identified using information provided by the cycle detection algorithm (i.e., messages m_1, m_2, m_3, and m_4 in Figure 13). Each message in this group is examined to see if all of the channels they are waiting for are owned by messages also in this group (as required by condition (2) of *Definition 8*). The subset of messages which meet this condition are further distinguished based on whether they own at least one message requested by a message also in the group (as required by condition (3) of *Definition 8*). Note that these conditions hold for messages m_1, m_2, m_3, and m_4 in Figure 13. Cycles are then examined to identify a group of them which only includes messages from this group. A group of cycles which meets this condition forms a knot, and is therefore identified as a deadlock and placed in the deadlock list as shown in Figure 13. Detected deadlocks are "broken" by removing a single message in the deadlock set in a flit-by-flit fashion so as to simulate an optimal recovery procedure.

We formally describe the deadlock detection algorithm as follows:

- Find a set of messages $M_P \subseteq M$ such that $\forall\ m_i \in M_P$, $|requests(m_i)| > 0$ and $(c_j, idx) \in Z_i$ for some cycle $Z_i \in Z$ and some branch index $idx\ \forall\ c_j \in requests(m_i)$.
- Find a set of messages $M_D \subseteq M_P$ such that $\forall\ m_i \in M_D$, $ownedby(c_j)=m_k$ where $m_k \in M_D\ \forall\ c_j \in requests(m_i)$.
- Find a set of cycles $Z_D \in P(Z)$ such that $ownedby(c_j) \in M_D\ \forall\ (c_j, idx) \in Z_i$ for some index $idx\ \forall\ Z_i \in Z_D$.
- The set $\bigcup c_j\ \forall\ (c_j, idx) \in Z_i$ for some index $idx\ \forall\ Z_i \in Z_D$ constitutes a knot and, therefore, a deadlock.

4.4 - Time and Space Complexity

The operations to build and maintain the CWG are invoked automatically by the simulator on an as-needed basis, and each of the operations requires constant time. The deadlock and cycle detection times are invoked based on user specified time intervals. For a simulation run of length T_{sim}

simulation cycles (sim-cycles), we allow the cycle detection algorithm to be run once every T_{cd} sim-cycles, $T_{cd} \leq T_{sim}$, and the deadlock detection algorithm to be run once every $(T_{cd} \times n) \leq T_{sim}, n \geq 1$ sim-cycles.

We use the following variables to express the computational complexity of the algorithms:

M_b - Average number of blocked messages in the network.
R_f - Routing freedom (average number of routing options per blocked message).
M_l - Average length of a message (number of distinct network nodes a message spans).
C - Number of unique cycles found in the current invocation of the algorithm.
V_c - Average number of vertices in a cycle.
M_c - Average number of messages in a cycle
C_a - Average number of active cycles in the network.
I_m - Average size of set of cycles with the same index (based on distribution generated by a hash function)
D - Number of unique deadlocks found during the current invocation of the algorithm.

The computational complexity of the cycle detection algorithm (cd_{time}) can be expressed as follows:

$$cd_{time} \approx \underbrace{\left(M_b \times \left(\frac{R_f}{2}\right)^{M_l}\right)}_{(a)} + \underbrace{\left\{C \times \left\{V_c + \left(\frac{M_c}{2}\right)^2 + \left(\frac{M_c \times C_a}{4}\right)^2 + \left(\frac{I_m \times M_c}{4}\right)\right\}\right\}}_{(b)}$$

In the above expression, the component (a) corresponds to the time required for full traversal of the CWG. The M_l parameter is limited by the average inter-node distance of a network, thus allowing us to easily traverse the CWGs of large multidimensional networks (i.e, *16*-ary *2*-cubes, *8*-ary *3*-cubes, etc.). The component (b) corresponds to the time required for "processing" (determine uniqueness, record characteristics, etc.) of each cycle found within the CWG. Currently, we are able to detect all cycles formed in networks which provide a moderate degree of routing freedom (i.e., true fully adaptive routing [3, 4] in 2D and 3D networks with as many as 4 VCs per physical channel) for up to saturation loads. Given the theoretical worst case for C and C_a of $(R_f)^{Mb}$, simulations can become impractical as a very large number of messages with high fanout may block within some networks at loads beyond network saturation.

The computational complexity of the deadlock detection algorithm (dd_{time}) can be expressed as follows:

$$dd_{time} \approx M_b + \{D \times C \times M_c\}$$

The M_b component of this expression corresponds to the examination and selection of a set of blocked messages which are candidates for participation in deadlock while the $\{D \times C \times M_c\}$ component corresponds to the validation of the properties of a set of messages in a deadlock. The time required for this algorithm has been greatly optimized by the use of heuristics based on information provided by the cycle detection algorithm. Since this deadlock detection algorithm relies on all cycles being detected, it is of limited use for detecting deadlock in deep saturation for some networks. We are currently implementing an alternative deadlock detection scheme which does not require that all cycles be identified and which therefore can be used to detect deadlock behavior during network conditions beyond saturation.

We use the following variables to express the space complexity of our implementation:

$^{*}Max_{vc}$ - The maximum number of VCs in the network.

$^{*}Max_{m}$ - The maximum number of messages allowed in the network.

$^{*}Max_{fr}$ - The maximum number of routing options (freedom) allowed for any message.

$^{**}Max_{cy}$ - The maximum number of cycles expected in the CWG at any given time.

$^{**}Max_{dl}$ - The maximum number of deadlocks expected in the network at any given time.

C_a - Average number of active cycles in the network.

M_c - Average number of messages in a cycle

D_a - Average number of deadlocks in the network.

C_d - Average number of cycles in a deadlock.

M_d - Average number of messages in a deadlock set.

$c_{1...}c_8$ - Small constants representing the sizes of various data structures.

The variables marked * are automatically determined based on simulation parameters while the variables marked ** can be specified by the user. The space complexity of our implementation can be expressed in terms of the space needed to maintain the channel wait-for graph, cycle list, and deadlock list as follows:

$$cwg_{size} \approx (Max_{vc} \times c_1) + (Max_m \times c_2) + (Max_{fr} \times c_3)$$

$$cyclelist_{size} \approx (Max_{cy} \times c_4) + (C_a \times M_c \times C_5)$$

$$deadlocklist_{size} \approx (Max_{dl} \times c_6) + (D_a \times C_d \times c_7) + (D_a \times M_d \times c_8)$$

5 - Related Work

Static channel dependency and wait-for graphs which represent connections allowed by routing algorithms have been presented in previous work [7, 8, 12]. In contrast, the channel wait-for graphs presented in this work are dynamic and represent the state of resource allocations and requests existing within a network at a given point in time. While the dependency and wait-for

graphs in previous work are intended for developing deadlock avoidance based routing algorithms, our channel wait-for graphs are used to precisely define deadlocks and related blocking behavior, to model resource dependencies within networks which allow unrestricted routing, and to implement deadlock detection in a network simulator.

Previous definitions of deadlock within interconnection networks include "deadlocked configurations" and "canonical deadlocked configurations" [7, 8, 11]. These notions of deadlock encompass not only messages which are directly involved in deadlock, but also other types of messages which are affected by deadlock. Furthermore, these definitions do not distinguish between single and multiple instances of deadlock which may exist simultaneously. In contrast, our definition of deadlock allows identification of every instance of deadlock, and distinguishes between the causal components of deadlock and those messages which are affected by deadlock. Therefore, our definition enables us to precisely define optimal deadlock resolution.

A summary of work characterizing deadlocks as knots in generalized resource graphs intended to describe deadlocks in operating systems is presented in [13]. The resource model presented here is a specialized version of this work, intended to precisely define deadlocks within interconnection networks.

5 - Concluding Remarks

We have presented here an approach to modeling resource allocations and dependencies within an interconnection network. We have formally defined various types of message blocking behavior, and have shown knots in channel wait-for graphs to be necessary and sufficient for deadlock. The framework we have provided distinguishes between messages involved in deadlock and those simply dependent on deadlock, thus providing a specification for deadlock recovery-based algorithms for properly resolving deadlock. We have also presented an approach to implementing deadlock detection for the purpose of simulation. Results of deadlock characterization performed using our implementation of deadlock detection is presented in other work [9, 10].

References

[1] D. Reeves, E.Gehringer, and A. Chandiramani. "Adaptive Routing and Deadlock Recovery: A Simulation Study". In Proceedings of the 4th Conference on Hypercube Concurrent Computers and Applications, 1989.

[2] Jae H. Kim, Ziqiang Liu, and Andrew A. Chien, "Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing", In Proceedings of the 21st International Symposium on Computer Architecture, IEEE Computer Society, pages 289-300, April 1994.

[3] Anjan K.V. and Timothy M. Pinkston, "An Efficient, Fully Adaptive Deadlock Recovery Scheme: Disha", In Proceedings of the 22nd International Symposium on Computer Architecture, IEEE Computer Society, pages 201-210, June 1995.

[4] Anjan K.V., Timothy M. Pinkston, and Jose Duato, "Generalized Theory for Deadlock-Free Adaptive Wormhole Routing and its application to Disha Concurrent", In Proceedings of

the 10th International Parallel Processing Symposium, IEEE Computer Society, pages 815-821, April 1996.

[5] L. Ni and C. Glass, "The Turn Model for Adaptive Routing", In Proceedings of the 19th International Symposium on Computer Architecture, IEEE Computer Society, pages 278-287, May 1992.

[6] Andrew A. Chien and Jae H. Kim, "Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors", In Proceedings of the 19th International Symposium on Computer Architecture, IEEE Computer Society, pages 268-277, May 1992.

[7] Jose Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks", IEEE Transactions on Parallel Distributed Systems, Vol. 4, No. 12, pages 1320-1331, December, 1993.

[8] Jose Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks", IEEE Transactions on Parallel Distributed Systems, Vol. 6, No. 10, pages 1055-1067, 1995.

[9] Sugath Warnakulasuriya and Timothy Mark Pinkston. "Characterization of Deadlocks in Interconnection Networks", To appear in Proceedings of the 11th International Parallel Processing Symposium, IEEE Computer Society, April 1997.

[10] Timothy Mark Pinkston and Sugath Warnakulasuriya. "On Deadlocks in Interconnection Networks", To appear in Proceedings of the 24th International Symposium on Computer Architecture, IEEE Computer Society, June 1997.

[11] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni, "Interconnection Networks: An Engineering Approach", To be published, 1997.

[12] Loren Schwiebert, D.N. Jayasimha, "A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing", Journal of Parallel and Distributed Computing, 32, 103-117 (1996).

[13] Mamoru Maekawa, Arthur E. Oldehoft, and Rodney R. Oldehoft, "Operating Systems: Advanced Concepts", Benjamin Cummings Publishing Company, 1987.