13

Deadlock Characterization and Resolution in Interconnection Networks

Timothy Mark Pinkston
University of Southern California, Los Angeles, CA, USA.

This chapter describes three important classes of deadlocking behavior that can occur in interconnection networks used in computer systems and discusses various techniques that can be used to handle them. Network attributes and phenomena that influence the formation of network deadlocks are identified. Ways in which a network's susceptibility to deadlock can be reduced are presented. Advantages and disadvantages of various proposed approaches are highlighted, and practical usage of certain techniques in experimental and commercial systems is also given.

13.1 INTRODUCTION

High-performance interconnection networks comprise the communication backbone in digital systems at several system levels. At the higher system levels, local-area networks (LANs) [1] are used in clusters of PCs, networks of workstations and other distributed processing systems which serve as cost/performance-effective alternatives to tightly-coupled massively parallel processing systems. System-area networks (SANs) [2] are used for interconnecting processors, memories, and I/O devices in systems with the primary goal of increasing reliability in the presence of link/router failures (often at the expense of duplicating physical resources). Storage-area networks (STANs) [3] are used to increase performance and reliability of large disk arrays by offering access to stored data by processors through multiple paths, thus providing continued service in the presence of processor failure. Internet protocol router fabric (IPRF) networks [4] are used within IP routers to handle IP traffic at high (multigigabit) sustained line rates. Server I/O (SIO) and interprocessor communication (IPC) networks [5,6] are used to overcome many of the scalability limitations of multichip bus-based systems, allowing high-speed interconnections between memory controllers and I/O devices, direct access to disk from LAN adapters, and concurrent communication between processors, memories and I/O devices in multiprocessors. Likewise, at lower levels, networks-on-chip (NOCs) [7–9] are used to overcome many of the performance limitations of bus-based systems at the chip level.

Parallel computing and communication systems built from the above networks require high-performance communication services with high reliability, availability and dependability—collectively, high robustness. The performance of the interconnection network is measured, in part, by packet delivery time from source to destination (i.e., latency) and by the number of packets delivered per unit time (i.e., throughput). In essence, a high-performance network allows the maximum number of packets to make forward progress to their destinations in minimal time, preferably along shortest paths to preserve network bandwidth. Likewise, the reliability, availability and dependability of a network equally impact the overall "goodness" quality of a system. These attributes are measured, in part, by the network's ability to remain up and running at near normal levels even when events occur which change its configuration, possibly due to changes in users' needs and/or system state. Such reconfiguration events may include, for example, hot-swapping of components, failure or addition of links/nodes, activation or deactivation of hosts/routers, etc., in a LAN environment depicted in Figure 13.1. Irrespective of the system, all of the above mentioned attributes have significant importance with the emergence of bandwidth-hungry applications such as high-definition video/audio-on-demand processing, distributed on-line transaction processing, database/decision support systems, grid and Internet applications. Such applications impose a great demand on the communication subsystem not only to be of high performance but also to be highly robust.

In the past, research concentrated on improving topological aspects of interconnection networks, but in recent years, research efforts have focused on improving the router as the primary means of increasing network performance. This includes efforts in such diverse areas as improving router switching, scheduling, injection limitation, flow control, and the routing algorithm. Among these, some of the more significant contributions have arisen from the notion of virtual cut-through switching [10], congestion control [11,12], virtual channel flow control [13,14], virtual output queuing [15] and adaptive routing [16–18]. Figure 13.2 shows a simple router model which allows the implementation of many of these techniques. Virtual cut-through switching allows pipelined transmission of packets across multiple routers and links without occupying multiple routers/links when a

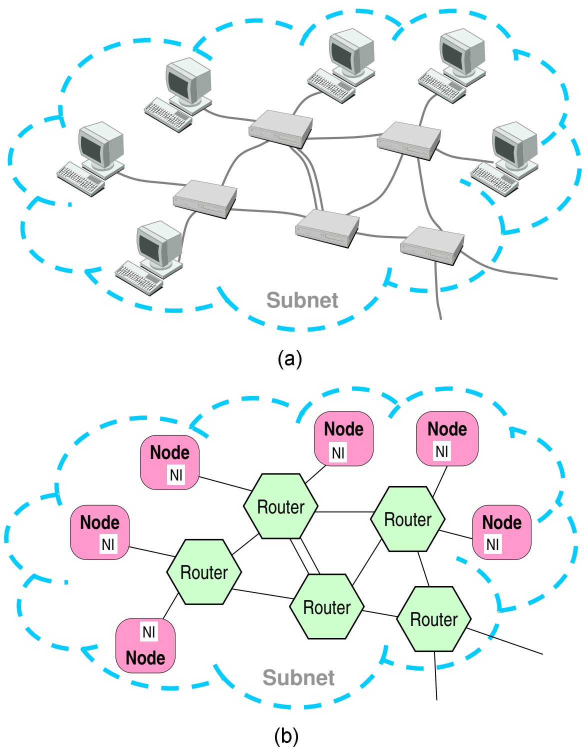


Figure 13.1. An illustration of a switched-LAN subnetwork: (a) its actual components, and (b) a high-level conceptual model of the network components, where NI is the network interface.

packet blocks, which is the case with wormhole switching [19]. Congestion control limits the number of packets injected into the network when the network load is considered to be too high (e.g., nearing saturation). This reduces the chances of the network becoming overly congested and saturated. Virtual channels and virtual output queuing mitigate head-of-line blocking of packets temporarily stored in channels (i.e., edge and/or central queues) during transmission. They provide logically independent multiple communication paths to packets multiplexed across each network link and/or the router crossbar. Adaptive routing increases the degree of flexibility in routing allowed by packets as they traverse the network to their destinations. This allows packets the option of choosing between multiple paths in the network according to prevailing blockage conditions.

Because network resources are finite and, ultimately, are contended for, structural hazards on those resources are inevitable which delay or prevent packet

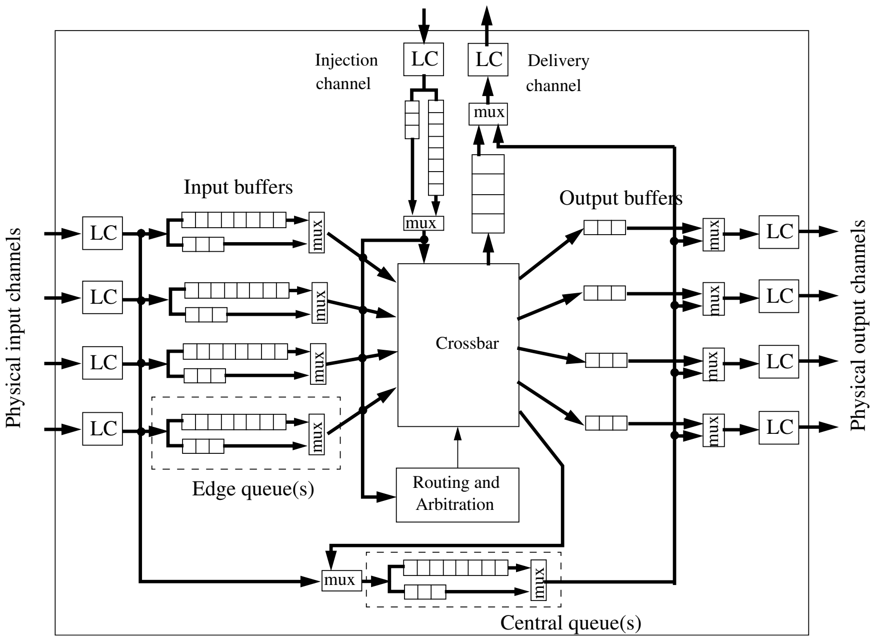


Figure 13.2. A simple model of a generic network router. Each physical channel may have one or more virtual channels associated with it, implemented as edge queue(s). Central queue(s) may also accept flits and be assigned physical channel bandwidth. Flow over the channels is controlled by link controllers (LCs), and multiplexed access (mux) to the internal crossbar ports and other shared resources of the router is determined by the routing and arbitration unit. (From Ref. 18 ©2003 IEEE.)

transmission in the network. This occurs even in networks that feature advanced router architectures. Such hazards cause packets to block which, eventually, can lead to network congestion and, possibly, *deadlock*. One of the more critical problems to be addressed in order to achieve high network performance and robustness is that of efficiently handling deadlock anomalies. Deadlock occurs when there is a circular hold-and-wait dependency relation on network resources by in-flight packets such that progress in routing those packets is indefinitely inhibited. That is, packets would block in the network forever unless some action to resolve the deadlock situation were taken. This phenomenon can result in the entire network (and system) coming to a complete stand-still, consequently degrading system reliability, availability, and dependability considerably. Thus, it is vitally important to guard against deadlock in such a way as not to impose overly restrictive measures that under-utilize network resources.

Deadlocks in interconnection networks are classified into three basic categories, depending on the circumstances under which they form. *Routing-induced deadlocks* are those caused by interactions and dependencies created within the network—between network endpoints—by the routing function which prevents packets from reaching their destinations. The routing function supplies the possible paths packets are allowed to take in the network to reach their destinations from their current locations. *Message-induced deadlocks* (also called protocol-induced deadlocks) are those caused by interactions and dependencies created at the network endpoints among different message types (i.e., requests, replies, etc.), which prevent packets from sinking upon arrival at their destinations. *Reconfiguration-induced deadlocks* are those caused by the interactions and dependencies created through time (dynamically) in a network that undergoes reconfiguration, which prevents packets from reaching their destinations due to being routed under the influence of multiple active routing functions. This can occur even if each of those routing functions is independently deadlock-free under static conditions. Note that we exclude from our deadlock categorization indefinite blocking situations which appear to be deadlocked but really are not, such as those arising from network disconnectivity, i.e., fault-induced indefinite blocking [20]. Such blocking situations are excluded from those termed as deadlock since, in fact, no cyclic hold-and-wait dependency relation on network resources exists.

Given this brief introduction, the remainder of this chapter is organized as follows. The next section describes ways in which deadlocks in interconnection networks can be depicted. This is followed by a section that presents the basic approaches for handling interconnection network deadlocks. While each of these approaches are applicable to all three forms of deadlock, specific examples for each class of deadlock are given. This chapter ends with a few concluding remarks, some bibliographic notes, acknowledgements, and references.

13.2 DEPICTING DEADLOCKS IN INTERCONNECTION NETWORKS

There are a number of different ways in which a network's deadlocking properties can be depicted. Two of the more common ways are to use channel dependency graphs (CDGs) or channel waiting (or wait-for) graphs (CWG). Both are directed graphs in which the vertices represent the channels (either physical or virtual) of the interconnection network. However, the arcs or edges between two channels (c_i, c_j) in CDGs denote a possible channel dependency from c_i to c_j allowed by the routing function whereas in CWGs they denote the next channel c_j reserved and being waited for by a packet in c_i. Thus, CDGs are static and depict all possible channel dependencies allowed by a routing function (whether currently used or not); CWGs are dynamic and represent actual resource allocations and requests existing in a network at a given instance in time. It follows that the CWG provides a critical subset of allowed dependencies represented by a network's CDG.

Channel dependency graphs and channel wait-for graphs can be used to depict the deadlocking properties of a network simply by noting the existence and make-up of cycles that may be present. If no cycles are present, there can be no deadlock as cycles are a necessary condition for deadlock, but they are not sufficient for deadlock to occur. If cycles are present in either of the graphs, the potential for deadlock exists, but whether or not deadlock actually exists for a given configuration of packets must be determined by examining the "reach" of dependencies involved in the cyclic relation. No deadlock exists as long as the reach of dependencies extends beyond the scope of cyclically related resources.

More formally, the *reachable set* of a vertex in a CWG is the set of vertices comprising all paths starting from that vertex. If the reachable set is larger than the set of vertices involved in the cycle, a way of escape for packets involved in the cyclic dependency is provided. If, however, the reachable set of all vertices in the cycle is the set itself, there would be no way of escape. A set of vertices that has this property is referred to as a *deadlock set*, and it forms a *knot* [21,22]. A knot comprises the set of resources involved in cyclic dependency for which there is no alternative reachable resource that can be used to escape from deadlock. As a result, once all resources comprising the knot become full, deadlock abounds. Below, the circumstances under which the three classes of deadlock form and how deadlocks may be depicted using CDGs and CWGs are presented.

13.2.1 Routing-Induced Deadlocks

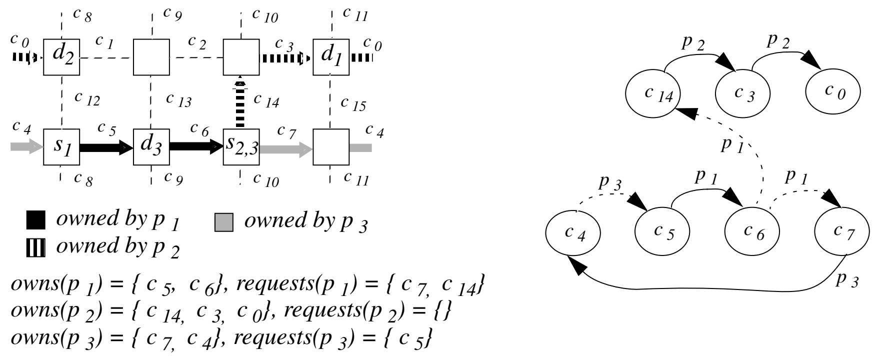
As noted to earlier, the routing function is responsible for supplying at least one output channel or delivery channel to a packet arriving on a given input channel or injection channel at each router. The aggregation of these routing-induced channel dependence relations for all possible packet configurations of a network is captured by the network's CDG. Accordingly, each channel used by a packet has a dependence relation on the next channel(s) supplied by the routing function, creating a chain or path of dependencies captured by the CWG. As such, routing-induced dependencies take into account only those dependencies on channel and queue resources shown in Figure 13.2; specifically, injection and delivery channels, edge queues, and/or central queues. Interactions occurring at network endpoints are excluded from this set, meaning that packets are assumed always to sink upon reaching their destinations. If knotted cycles appear along a fully occupied set of these resources, *routing-induced deadlock* is said to form.

The likelihood of routing-induced deadlock occurring is largely influenced by three key interrelated factors: the *routing freedom* of the routing algorithm, the number of *blocked packets* within the network, and the number of *resource dependency cycles* as depicted in the channel wait-for graph [22]. Routing freedom corresponds to the number of routing options available to a packet being routed at a given node within the network. Routing freedom is reflected in the CWG by the fan-out of vertices, which can be location dependent for packets en route in the network: smaller fan-out than that which is maximally allowed by the routing function may be the result if adaptivity is exhausted. Blocked packets are those packets in the network that cannot acquire any of the alternative channels required to make progress at a given point in time due to those channels being fully occupied by other packets. Correlated packet blocking can result in wait-for dependency cycles on resources, possibly leading to deadlock.

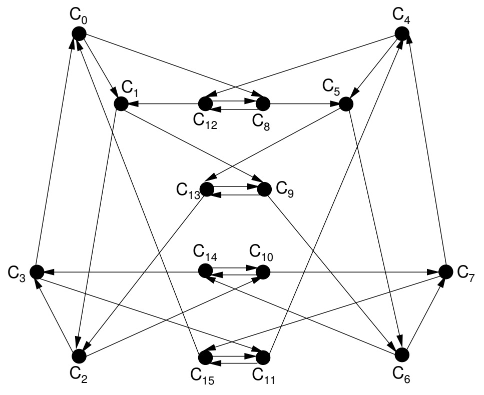
On the one hand, routing freedom can considerably increase the potential number of cycles that can form once packets block, i.e., at network saturation. This is due to the compounding effects of routing freedom with the number of network hops experienced by packets and the number of blocked packets in the network. For example, given an average distance of h hops each allowing a routing freedom of F channel options at each router, the theoretical upper bound on routing freedom of a packet from source to destination is F^h (i.e., routing a packet diagonally over an n-dimensional space), whereas blocking opportunity is only h. This upper bound on routing freedom may not be reached if F decreases as hops are taken (i.e., routing a packet minimally over the periphery of an n-dimensional space). Given a total of B blocked packets each with a routing freedom of F, the theoretical upper bound on the number of unique cycles which can form is F^B. This upper bound may not be reached due to limitations on possible topological connections allowed. Nevertheless, as the number of blocked packets increases due to an exhaustion of routing freedom and/or the network reaching saturation, the potential for cycle formation increases considerably, which can increase the probability of deadlock.

On the other hand, routing freedom has an opposite and more influential effect on deadlock probability than it does on the creation of cycles. As routing freedom is increased, the number of blocked packets decreases substantially. More importantly, the degree of correlation required among blocked packets to form a knot also increases substantially. This greatly decreases the likelihood of the occurrence of deadlock. Given enough routing freedom, this correlation factor offsets the opposing effect on deadlock probability caused by the potential increase in the number of cycles. Networks with minimal routing freedom may not offset the opposing effects as there may exist a one-to-one correspondence between cycles and deadlocks, e.g., single-cycle deadlocks. However, networks with greater routing freedom may offset the opposing effects as a large number of cycles can exist without deadlock formation, e.g., cyclic non-deadlocks.

Consider, for example, the 2×4 torus network shown in the left-hand side of Figure 13.3. If the network's routing function supplies all possible channels along minimal paths to packets' destinations from their current positions (i.e., adaptive minimal routing is used), the CDG shown on the right-hand side of the figure is formed. Clearly, cycles exist in this graph, indicating that deadlock may be possible for some configurations of packets routing in the network. The CWG (shown in the middle) for packet configuration p_1, p_2, and p_3 routing in this network depict the critical subset of allowed dependencies "alive" in the network, assuming wormhole switching. The channels currently occupied (possibly only partially)


owned by p 1
owned by p 3
owned by p 2
owns(p 1) = { c 5, c 6}, requests(p 1) = { c 7, c 14}
owns(p 2) = { c 14, c 3, c 0}, requests(p 2) = {}
owns(p 3) = { c 7, c 4}, requests(p 3) = { c 5}
d2
d1
s1
d3
s2,3
p 2
p 1
p 3

(a)



(b)

Figure 13.3. (a) The network graph and channel wait-for graph, and (b) the channel dependency graph. These are for packets routed adaptively within a 2×4 torus network (the links on the periphery are wrap-around links). Although cycles exist in the CWG and CDG, this packet configuration is not deadlocked.

by packet p_i are represented by the set $owns(p_i)$ shown as solid arcs and channels supplied by the routing function for the packet to continue routing is represented by the set $requests(p_i)$ shown as dotted arcs. The added routing freedom given by adaptive minimal routing is reflected in the request set for some blocked packets having a cardinality greater than one (i.e., $|requests(p_1)| = 2$). As shown, a cycle for this packet configuration exists consisting of the set of vertices $\{c_4, c_5, c_6, c_7\}$. However, there is no deadlock as the reach of dependencies extends beyond the set of resources involved in this cycle of dependency.

This cyclic non-deadlock situation is easily verified by examining the reachable set for each vertex. The reachable set for vertex c_{14} is $\{c_3, c_0\}$; for c_3 is $\{c_0\}$;

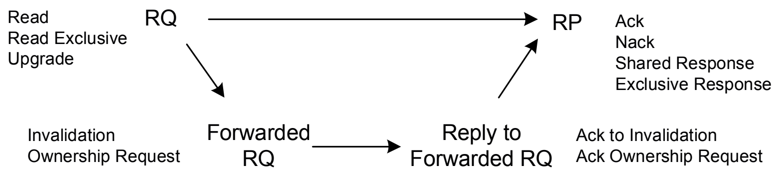


Figure 13.4. Ordering among message types allowed by a typical cache coherence protocol, where RQ and RP are request and reply message types, respectively, and arrows indicate a message dependency from one message type to another.

for c_0 is $\{\}$; and for vertices c_4, c_5, c_6 and c_7 which are involved in the cycle, the reachable set is $\{c_4, c_5, c_6, c_7, c_{14}, c_3, c_0\}$. Clearly, the reachable set is larger than the set of vertices involved in the cycle, thus providing a way of escape for packets involved in the cyclic dependency. If, however, the destination for p_1 were in the bottom row instead of the top row of the same column, the reachable set of all vertices in the cycle would be identical to the set itself, i.e., $\{c_4, c_5, c_6, c_7\}$, as there would be no escape through channel c_{14} for packet p_1. Given packet p_1's current position relative to the new destination, routing freedom is decreased as the routing function supplies only channel c_7. In essence, all packets involved in the cycle have exhausted their adaptivity. This resource set, therefore, comprises a deadlock set on which a knot forms—which, in this case, is composed of only a single cycle. Adaptive routing functions typically have knots composed of a large number of cycles in most cases, i.e., multicycle deadlocks. No matter the case, routing-induced deadlock occurs once all channels comprising the deadlock set become fully occupied. The occurrence of such deadlocks, however, have been shown to be very rare and may never occur in practice when multiple virtual channels are used with maximum routing freedom [22,23].

13.2.2 Message-Induced Deadlocks

The exchange of various types of messages is pervasive in computer systems in which interconnection networks are employed. Many message types—as defined by the communication protocol of the system—may be used to complete data-interchange transactions. For instance, cache coherence protocols may permit data transactions to be composed of certain combinations of *request*,*forwarded-request*, *reply-to-forwarded-request*, and *reply* message types, as shown in Figure 13.4. At any given end-node in the system, there can be a coupling between the two message types: the generation of one message type, e.g., the reply message generated by the destination, is directly coupled to the reception of another message type, e.g., the request message received by the destination. As the coupling between message types is transferred to network resources due to the finiteness of resources along the

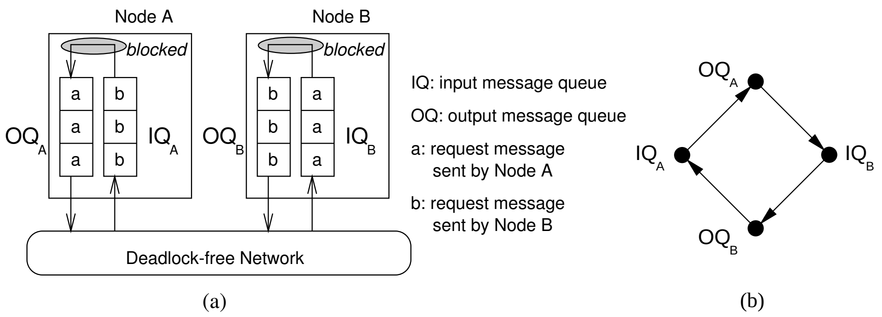


Figure 13.5. (a) A simple example of message-induced deadlock occurring between two nodes connected by a network free of routing-induced deadlock. (b) The corresponding dependency graph for resources at network endpoints, where dependencies form a cyclic (and knotted) wait-for relationship.

message path inside each node (at network endpoints), additional dependencies on network resources are created, referred to as *message or protocol dependencies*.

A distinct class of message dependency is created for each pair of message types for which a direct coupling exists and is transferred to network resources. Each combination may present different kinds of message dependencies and a corresponding *message dependency chain*, which is a series of message types used to complete a transaction on network resources. Since message dependencies may prevent messages from sinking at their destinations, they must be added to the complete set of resource dependencies. If knotted cycles form along a set of resources when resources at the network endpoints are also taken into account, a type of deadlock called *message-induced or protocol-induced deadlock* forms once all resources comprising the knot become full.

A message dependency chain represents an ordered list of message dependencies allowed by the communication protocol. We define the partial order relation "$\prec$" between two message types m_1 and m_2 by the following: $m_1 \prec m_2$ if and only if m_2 can be generated by a node receiving m_1 for some data transaction. Message type m_2 is said to be *subordinate* to m_1, and all message types subordinate to m_2 are also subordinate to m_1. The final message type at the end of the message dependency chain is said to be a *terminating* message type. The number of message types allowed within a message dependency chain is referred to as the *chain length*. For example, if the system defines only two message types, *request* and *reply*, for all data transactions and the message types establish the dependency relation *request* $\prec$ *reply*, then the chain length is two.

Consider the following example of queue resource sharing at network endpoints by messages of two different types (*request* $\prec$ *reply*), shown in Figure 13.5. Depicted in the figure is a simple message-induced deadlock represented by a

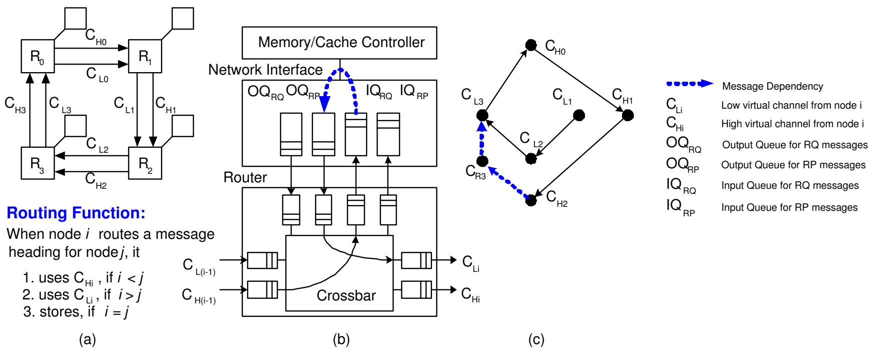


Figure 13.6. (a) A four node system interconnected by a unidirectional ring network using two virtual channels to avoid routing deadlocks, as described in Section 13.3.1. (b) Message dependencies occurring inside network interfaces. (c) The corresponding resource dependency graph consisting of network channels (i.e., C_{Hj} and C_{Lj}) and queue resources at Node $R3$ (i.e., C_{R3}). (From Ref. 61 ©2003 IEEE.)

resource dependency graph in which two nodes, Node A and Node B, are each sending request messages to one another and expecting to receive reply messages over a network free from routing-induced deadlock. Each node's network interface has an output message queue and an input message queue, which are used to buffer messages of any type being injected into or received from the network. If no buffer space is available in the output queue, no message in the input queue that generates a subordinate message type is serviced. Otherwise, the message generated by the serviced message would cause overflow in the output queue or indefinitely stall the network interface from servicing other messages. These are situations which are avoided to ensure correct execution. If the arrival rate of request messages exceeds the consumption rate, a backlog starts to form at the input message queue IQ_A at Node A. After a while, the backlog propagates backward in the direction of message injection at the output message queue OQ_B at Node B. The backlog eventually reaches the input message queue IQ_B at Node B, and, further, to the output message queue OQ_A at Node A. At this point, a deadlock forms as no buffer space can be freed for reply messages needed by both nodes to continue execution.

Consider next how the sharing of channel resources *between* network endpoints (i.e., within the network) among messages of two different types (*request* $\prec$ *reply*) can cause message-induced deadlocks. Shown in Figure 13.6 is a simple four node system interconnected by a unidirectional ring network consisting of two virtual channels used in such a way as to avoid routing-induced deadlock between network endpoints (refer to Section 13.3.1 to understand the routing function given in Figure 13.6(a)). A processor-memory node is connected to each router via the node's network interface which transmits and receives messages to/from the network through its output and input queues, respectively. Figure 13.6(b) depicts the message dependency that can occur at the network endpoint. The channel dependency graph for the routing algorithm is shown as solid arcs in Figure 13.6(c) and is cycle-free between endpoints. However, when node $R1$ sends a request message to $R3$ which responds back to the request by sending a reply message to $R1$, a message dependency from C_{H2} to C_{L3} exists in the network (shown as dotted arcs in the figure for node $R3$ only) through the network interface channel represented by C_{R3} at $R3$. This completes the cycle in the channel dependency graph, making message-induced deadlocks possible.

13.2.3 Reconfiguration-Induced Deadlocks

When a change in network topology or routing arises through time, it may be necessary to reconfigure the routing function in order to remap and/or reconnect routing paths between nodes in the system. Reconfiguring a network's routing function can cause additional dependencies among network resources both during and after the reconfiguration process that are not independently allowed by either the old or new routing functions. The paths of channels occupied by some undelivered packets routed with the old routing function (i.e., their *configuration*) could be illegal under the new routing function. As a result, two adjacent units of such packets (called *flits*[1]) could be stored in two different channels—one allowed only by the old routing function and the other allowed only by the new routing function. This can create a set of residual dependencies, referred to as *ghost dependencies* [24], that must be taken into account in the total set of resource dependencies when determining the network's deadlock properties. Ghost dependencies can interact with dependencies allowed by the new routing function to close dependency cycles on resources used to escape from deadlock, causing *reconfiguration-induced deadlock*.

Figure 13.7 illustrates how ghost dependencies brought on by undelivered packets in a wormhole network undergoing reconfiguration can cause reconfiguration-induced deadlock. The arrows in the figures indicate the up directions assigned to the links for up*/down* routing [25], a routing technique that is free from routing-induced deadlocks (see Section 13.3.1). Up*/down* routing allows packets to follow any path leading to their destination which is comprised only of zero or more up links followed by zero or more down links. This avoids cycles from forming, which makes the routing algorithm deadlock-free in the case of no reconfiguration. In the example shown, the network undergoes reconfiguration in response to the old root node being removed. Link directions are altered for some links in order for a new root node to be established for the new up*/down* routing function. Reconfiguration-induced deadlock may be caused by the ghost

[1]A *flit* is the smallest unit of a packet on which flow control is performed. The flit size can be less than the packet size when wormhole switching is used; it is equal to the packet size when virtual cut-through switching is used.

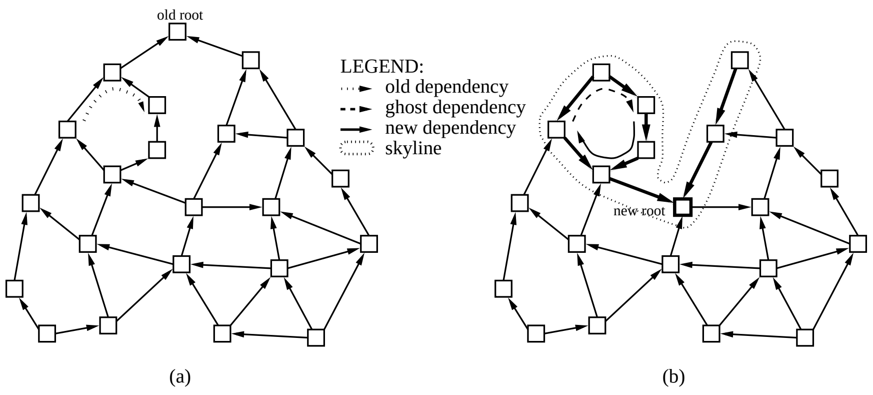


Figure 13.7. (a) Reconfiguration of a network that uses Up*/Down* Routing which is free from routing-induced deadlock (see Section 13.3.1). In it, packets route toward their destinations over paths consisting of zero or more "up" links followed by zero or more "down" links. The "up" direction for each link in the figure is indicated by the direction of the arrowheads (and is also implied by the relative vertical positions of the nodes). In (b), the old root node is removed, which triggers a new root node to be discovered within the skyline of the network (enclosed by dotted lines), where the thicker lines indicate links which reverse their directions after reconfiguration completes. Ghost dependencies (dashed arc) carried from the old routing function can form a dependency cycle with new dependencies (solid arc) from the new routing function, thus creating the potential for deadlock.

dependency between channels, shown as a dashed arc. This dependency is not among the normal dependencies allowed by the new routing function, yet it persists even after reconfiguration is completed if a packet that was routed using the old routing function remains in the network and holds resources that were supplied by the old routing function. Its existence closes the channel dependency cycle as shown, preventing escape from deadlock once all affected resources become fully occupied.

13.3 APPROACHES FOR HANDLING DEADLOCKS

Approaches for handling deadlock in interconnection networks are mainly based either on applying restrictions to avoid them (i.e., avoidance-based) or on lifting restrictions but supplying nominal resources to detect and resolve them (i.e., recovery-based). A third approach is based on reserving all needed resources prior to transmission. This way, deadlock is altogether prevented as no blockage of any kind is encountered. However, since this strategy is typically used only in legacy circuit switched networks, it will not be discussed further in this chapter as we focus on techniques for packet switched networks.

From an implementation point of view, the primary distinction between avoidance and recovery approaches lies in the tradeoff made between increasing the routing freedom and reducing the potential for deadlock formation. Routing freedom can be increased by adding physical channels, i.e., increasing the number of alternative paths *available across* different directions and dimensions of the network by using bidirectional and high-degree (rich) topologies. It can also be increased by adding more virtual channels per physical channel, i.e., increasing the number of logical routing options *available within* each dimension/link. Increasing the adaptivity of the routing function also increases routing freedom as the number of routing options *allowed within and across* each dimension is increased. However, as previously discussed in Section 13.2.1, increasing the routing freedom increases the potential number of cycles and, likewise, affects the probability of knot formation. Hence, the advantages of techniques based either on deadlock avoidance or deadlock recovery depend on how infrequently deadlocks might occur if routing freedom is maximized and how efficiently packets can be routed through the network if routing freedom is restricted such that deadlocks are completely avoided.

Let us first consider deadlock avoidance approaches. The simplest way of avoiding deadlock is to disallow the appearance of cycles in the network's CDG [19]. As cyclic resource dependency is necessary for deadlock, the lack of cycles precludes deadlock. We can think of this as deadlock avoidance in the *strict sense* since the phenomena that precipitate deadlock, i.e., cyclic dependencies, cannot exist anywhere in the network. An alternative way of avoiding deadlock is to allow cycles in the network's CDG but to enforce routing restrictions only on a subset of network resources used to escape deadlock such that all dependencies on those resources (as given by an *extended* CDG) are acyclic [16]. This is the situation in which the "reach" of all possible cyclic dependencies are guaranteed at all times to extend outside the scope of cyclically related resources. We can think of this as deadlock avoidance in the *wide sense* since cyclic dependencies which precipitate deadlock at certain points in the network are prevented from escalating into knotted dependencies due to the constant availability of escape resources network-wide. Yet a third alternative for avoiding deadlock in the *weak sense* is to allow cycles in the network's CDG and extended CDG, and even to allow knots to form, but to require at least some subset of escape resources (i.e., those in the extended CDG) to be large enough such that they never become fully occupied [26]. This guarantees that packets along knotted resources eventually are able to make forward progress, albeit slowly.

Deadlock recovery approaches, on the other hand, require correct detection and resolution of all potential deadlocks that may occur in a network. Precisely detecting the occurrence of a potential deadlock situation in interconnection networks requires an excessive amount of distributed resources controlled under a complex management mechanism. To reduce costs, less accurate heuristic techniques that detect all true potential deadlock situations but, consequently, also some occasional false ones are typically used, such as time-out or flow-control based mechanisms [27]. Once detected, deadlock can be resolved either in the wide sense or in the weak sense.

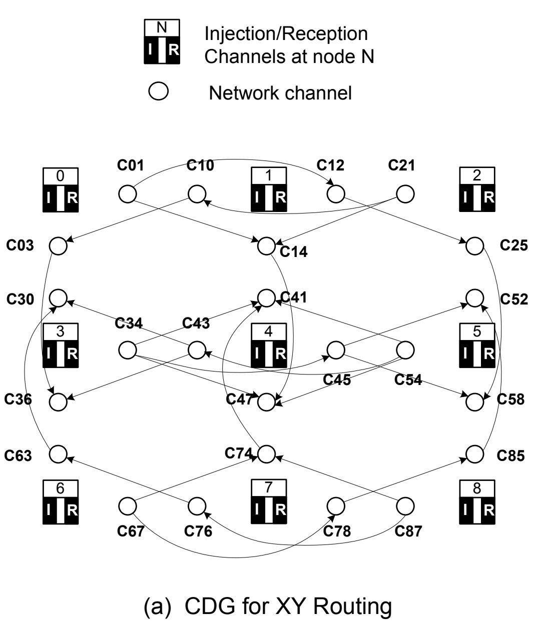
The simplest way of resolving potential deadlock is to remove from the network one or more packets in the deadlock set [20], i.e., by killing and later re-injecting it for subsequent delivery after some random delay. We can think of this as *regressive* deadlock recovery since the abort-and-retry process, in effect, makes packets regress back to their source in order to resolve deadlock. Alternatively, deadlock can be resolved by ensuring that at least one packet in the deadlock set no longer waits only for resources occupied by other packets in the aggregation of all deadlock sets in the network [20]. This can be done either *deflectively* or *progressively*, depending upon whether recovering packets make progress toward their destination or are simply deflected out of the potential deadlock situation obliviously with regard to their ultimate destination.

Below, some well-known as well as some recently proposed approaches for handling the three types of deadlocks that can occur in interconnection networks are described. As mechanisms for handling each is discussed separately, it is assumed that other mechanisms are in place to sufficiently handle the other two types of deadlocks, i.e., using the avoidance or recovery techniques discussed in the other sections given below.

13.3.1 Handling Routing-Induced Deadlocks

A. Routing-Induced Deadlock Avoidance in the Strict Sense

Prior to the past decade, routing-induced deadlock was handled primarily by avoiding it in the strict sense. Cyclic-wait situations were completely avoided by imposing severe restrictions on the order in which all network channel queues were allocated to packets. Techniques for accomplishing this can be classified as being *path-based*, *queue-based*, or some *hybrid* combination of the two. Queue-based techniques may be subdivided into *channel-based* schemes and *buffer-based* schemes. The idea behind path-based schemes is to avoid cyclic-wait dependencies on resources by restricting the possible paths packets can take from source to destination, as supplied by the routing function. By prohibiting certain turns in routing while maintaining network connectivity, it is possible to eliminate some critical resource dependencies that might otherwise close dependency cycles. This effectively leads to some total ordering being imposed on the use of network resources. Similarly, queue-based schemes accomplish the same by restricting the possible queues (virtual channels for channel-based schemes or queue buffers for buffer-based schemes) used by packets to their destinations, as supplied by the routing function.



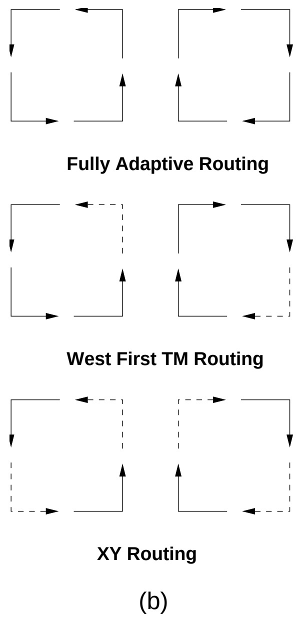


Figure 13.8. (a) A 3×3 bidirectional mesh network with its corresponding acyclic CDG shown for XY dimension-order routing (i.e., channel Cij is the channel from node i to node j). (b) Inter-dimensional turns allowed for routing functions with higher (top) to lower (bottom) routing freedom, where dashed arcs indicate turns disallowed by the routing function.

A number of path-based schemes have been proposed and many implemented in multiprocessor, network-based, and supercomputer systems. Dimension-Order Routing (DOR), used in the Cray T3D [28] and SGI Origin 2000 [29] for example, constrains packets to *deterministically route* minimally to their destinations in network dimension order. Figure 13.8(a) shows the network and channel dependency graph for XY dimension-order routing on a 3×3 mesh, where packets must reach their X dimension coordinates first before routing in the Y dimension. DOR is similar except that routing freedom along the two directions of each dimension is increased. Turn Model (TM) Routing [30], used in the Rapid Reconfigurable Router (RCube) [31], prohibits the minimum number of turns needed to prevent inter-dimensional cycles in an n-dimensional mesh in order to further increase routing freedom. This is $n(n-1)$ or a quarter of the possible turns instead of half the turns for DOR, as shown in Figure 13.8(b). West-first Turn Model routing, for example, requires all packets to be routed non-adaptively west first before being routed adaptively in other directions. Although the Turn Model increases routing freedom, its applicability is limited primarily to mesh networks. Up*/Down* Routing, used in Autonet [25] and Myrinet [1] for example, is generally applicable to arbitrary network topologies. As shown in Figure 13.7, each link is assigned a direction, and packets route toward their destinations over paths consisting of zero or more "up" links followed by zero or more "down" links. This forces an ordering on network resources such that turn dependencies from "down" links to "up" links are prohibited, thus disallowing cycles to be completed in the network's CDG.

Path-based schemes have the advantage of not requiring virtual channels to strictly avoid deadlock. However, their routing flexibility (i.e., routing freedom) is sacrificed and, more importantly, these schemes may not always provide a complete deadlock-free solution for networks with wrap-around links, like rings and tori, unless some network links (i.e., the wrap-around links) go unused. This is demonstrated for the ring network shown in Figure 13.9(a), where its corresponding cyclic CDG is shown in Figure 13.9(b). Queue-based schemes can effectively deal with network topologies that inherently have cycles since they decouple the allocation of queue resources (virtual channels or buffers) from the use of physical links by packets. Buffer-based schemes, such as structured buffer pools [32,33] used in legacy packet switched networks, place packets into buffer classes associated with the number of hops left to reach their destinations. This requires $D + 1$ distinct buffer classes[2], where D is the network diameter which is the maximum number of hops (or distance) between any two nodes along a minimal path in the system.

Channel-based schemes which use virtual channels such as [19,34,35] allow the number of required queue resources to be independent of the network diameter. As shown in Figure 13.9(c) and (d) (and alluded to earlier in Figure 13.6) for a four node unidirectional ring network, each physical channel can be associated with two virtual channels—a high channel and a low channel—and an ordering on the use of those virtual channels can be imposed explicitly by the routing function. For example, in the figure, physical channel c_0 from router 0 to router 1 is associated with virtual channels c_{H0} (high) and c_{L0} (low). The routing function supplies the high channel c_{Hi} to a packet at router i destined to node j if $i < j$; or it supplies the low channel c_{Li} if $i > j$; otherwise, it delivers the packet to the node associated with router i. Thus, deadlock is strictly avoided with the use of just two virtual channels per physical channel independent of network diameter, but only one of those channels is supplied to packets at any given point in the network. That is, there is an upper bound on routing freedom and network capacity of only 50% utilization of virtual channel resources.

Pure channel-based schemes that do not restrict the allowable paths packets can take in routing to their destinations, i.e., those that allow *fully adaptive routing*

[2]This number can be halved using a *negative hop* scheme in which adjacent nodes are partitioned into distinct positive and negative subsets (i.e., via graph coloring), and packets are placed into a buffer of class j if it has traveled exactly j negative hops.

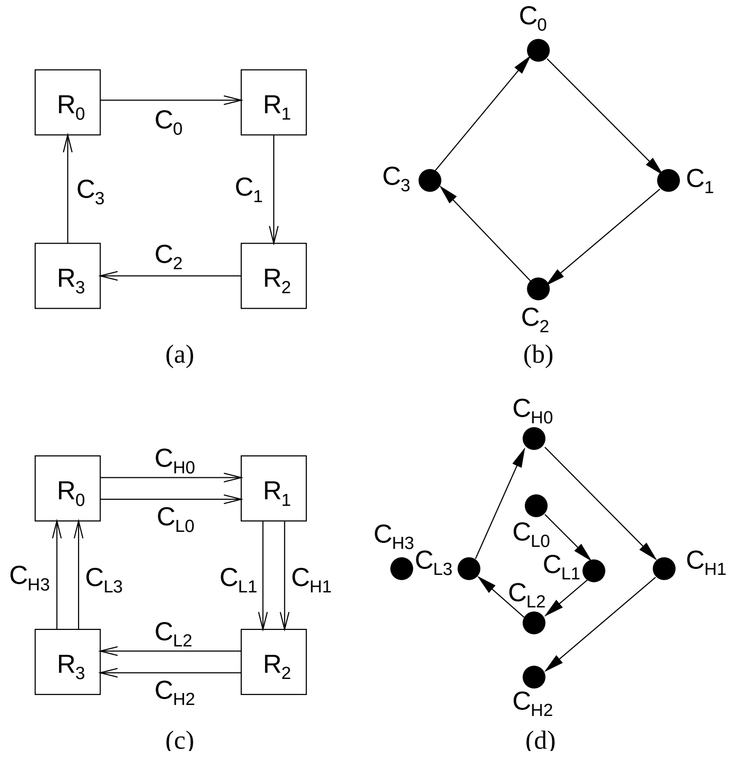


Figure 13.9. (a) A four node unidirectional ring network and (b) its corresponding cyclic CDG for unrestricted routing. (c) The same four node unidirectional ring network with two virtual channels per physical channel and (d) its corresponding acyclic CDG for a routing function which enforces an ordering on the use of the virtual channels.

illustrated at the top of Figure 13.8(b), such as Linder and Harden's scheme [34], consequently require an exponential growth[3] in the number of virtual channels needed to prevent cycles in the CDG. This growth can increase the implementation complexity of the router considerably. Even with such a large number of virtual channels, these schemes allow only a small subset of those resources to be usable by each packet as virtual channels are grouped into ordered levels and classes. This has motivated the development of hybrid schemes that are based partially on both path and channel restrictions. Hybrid path/channel schemes are prevalent in systems which strictly avoid reconfiguration-induced deadlock, including the

[3]This growth is based on network dimensionality, i.e., $2^{n-1}(n+1)$ virtual channels per physical channel are required, where n is the number of network dimensions.

SGI Spider [36], and Intel Cavallino [37] to name a few. As an example, Planar Adaptive Routing (PAR) [35] is a *partially adaptive routing* scheme that strictly avoids deadlock using a combination of both path and channel restrictions. This scheme requires three virtual channels per physical channel in n-dimensional mesh networks and six in torus networks, of which only one can be used in each of the allowed dimensions by packets routing in the network. Deadlock is avoided in the strict sense by restricting the possibility of routing adaptively to, at most, two dimensions at a time and by ordering the traversal of adaptive planes taken by packets. Although routing freedom is increased over a purely deterministic path-based scheme like DOR, it is still very much limited as some idle channels along minimal paths in the $n-2$ other dimensions of the network (i.e., those not in the current adaptive plane) are automatically excluded from use by blocked packets since they lie outside of the adaptive plane.

B. Routing-Induced Deadlock Avoidance in the Wide Sense

Routing freedom can be increased considerably while still guarding against routing-induced deadlock using routing schemes based on avoiding deadlock in the wide sense. This has been the technique of choice for most systems implemented within the past decade and continues to be popular since network resources are used much more efficiently. The most well-known and widely used scheme is Duato's Protocol [16]. It is a hybrid path/channel-based scheme that has seen widespread use in research and commercial systems, including the MIT Reliable Router [38], Cray T3E [39], Alpha 21364 [40], and IBM BlueGene/L supercomputer [41]. The idea behind this scheme is to allow unrestricted fully adaptive routing on the majority of network virtual channels while providing a way of escape from deadlock on a connected minimal subset of virtual channels.

Virtual channels are divided into two classes or sets: one susceptible to cyclic dependencies, i.e., those on which fully adaptive routing is permitted, and the other free from cyclic dependencies, i.e., those on which escape routing (usually deterministic) is performed. The escape channel set can be made deadlock-free by using any of the strict deadlock avoidance techniques described previously on them. Deadlock is avoided in the wide sense as no restrictions are placed on the adaptive channels. This can lead to dependency cycles forming on them. When packets block (possibly cyclically) on adaptive channels, they may use escape channels which are always supplied by the routing function. A packet can either stay on the set of escape channels until it reaches its destination or it can come back to adaptive channels, depending on which switching technique is used (i.e., wormhole or virtual cut-through, respectively) [17]. The existence of a coherent and connected escape path which has no cycles in its extended channel dependency graph is sufficient to avoid deadlock in the wide sense.

A number of schemes similar to Duato's Protocol based on avoiding deadlock in the wide sense have also been proposed in the past decade, but with slight differences. The more interesting are the *-channels [42], Schwiebert and

Jayasimha's algorithm [43], and the Dynamic Routing Algorithm by Dally and Aoki [44]. As an example, deadlocks are avoided in the wide sense by the Dynamic Routing Algorithm by not allowing cycles in the packet wait-for graph instead of the channel wait-for graph. In this hybrid path/channel-based scheme, each packet keeps track of the number of network dimension reversals it makes. Similar to Duato's Protocol, a packet is routed adaptively on any channel in the adaptive class until blocked with all supplied output channels being used but, in this case, by packets with equal or lower values of dimension reversals. Upon this condition, the packet is forced onto the deadlock-free deterministic class of channels and must remain there until routed to its destination. A packet's dimension reversals relative to other packets at a given router ultimately places an upper bound on adaptivity, and the deterministic channel class ultimately becomes a performance bottleneck.

Hybrid path/channel-based routing schemes which avoid deadlock in the wide sense may have the advantage of requiring only a few additional resources (in the form of virtual channels) over that required by pure path-based schemes which avoid deadlock in the strict sense, but they still have some inefficiencies. Although all virtual channels in the adaptive set are supplied to packets, typically only one of the virtual channels composing the escape set is supplied to packets at each router. Restrictions on escape virtual channels consequently cap the routing freedom of such schemes below the upper bound provided by *true fully adaptive routing*, where full adaptivity is maintained across all physical link dimensions of a network topology as well as across all virtual channels within a given link dimension, without restriction. For example, if three virtual channels were implemented in a torus network, these schemes could use only one virtual channel for fully adaptive routing and only one is supplied for escape routing whereas all three could be used by a true fully adaptive routing scheme. Likewise, if only two virtual channels were implemented, fully adaptive routing would not be permitted on either channel by these schemes, but it would be permitted on both with a true fully adaptive scheme. Consequently, it may be very inefficient to allow fully adaptive routing on only a subset of the total number of virtual channels, particularly if knotted dependencies which can lead to deadlock occur very infrequently, as is shown to be the case typically [22,23].

C. Routing-Induced Deadlock Avoidance in the Weak Sense

Routing freedom can be increased further by using routing schemes which avoid deadlock in the weak sense by never allowing knotted resources to become fully occupied. These are typically hybrid path/queue-based schemes which operate on the principles of deflection and/or injection limitation. To better understand these principles, consider closely what takes place when packets make forward movement in the network. Each time a packet moves forward, an empty buffer in a queue associated with the next router port along the path is consumed by the head of the packet. Likewise, an occupied buffer associated with a prior router port is released by the tail of the packet. Assuming that the unit of an empty buffer space is defined as a *bubble* [26], each forward movement of a packet in one direction is equivalent to the backward propagation of a bubble in the opposite direction. Thus, the movement of packets in a network can be characterized simply by considering the availability of bubbles in resources needed by packets and how those bubbles flow through those network resources: the more bubbles flowing within needed network resources, the greater the number of packets that can make forward progress.

All of the techniques mentioned in Section 13.1 in some way affect the availability and/or flow of bubbles in the network. Virtual cut-through switching defines the granularity of bubbles to be the size of a packet. Network throttling or injection limitation regulates the number of bubbles that flow out of the network (or network dimensions) to keep the number of bubbles available within the network (or network queues) above some threshold. Virtual channel flow control confines bubbles to flow within logical networks so that bubble movement within different logical networks can be independent of one another, subject to the channel scheduling algorithm. Likewise, virtual output queuing confines bubbles to flow within separate network dimensions. Techniques for avoiding routing-induced deadlock in the strict or wide sense as discussed in Sections A. and B. restrict bubble flow such that all bubbles in the network always flow through some defined subset (or entire set) of network resources in some total or partial order. Given the above facilities to manipulate bubbles and their displacement, the real challenge essentially becomes how to apply the fewest restrictions on bubble flow so as to always maintain deadlock freedom, thus avoiding deadlock in the weak sense.

Increasing the routing freedom relaxes restrictions on packet movement, which allows bubbles to flow more freely among network resources, but some restrictions must still be enforced to ensure deadlock freedom. The idea behind techniques designed to avoid deadlock in the weak sense is to allow complete freedom of forward packet movement without regard for avoiding cyclic or knotted dependencies. The resulting flexibility in routing options provided by true fully adaptive routing on a multitude of network resources (i.e., virtual channels, virtual output queues, etc.) reduces the probability that correlated resource dependencies form in the first place, as discussed in Section 13.2.1.

In the rare case that knotted dependencies do form on a set of queues, deadlock can be avoided by *limiting injection* into that set by resources external to them once some threshold on their occupancy is reached. This concept is illustrated in Figure 13.10 for the case of a threshold equal to one buffer space less than the maximum aggregate queue capacity along the cycle (Figure 13.10(a)). Such a cycle could arise from inter-dimensional turns or from routing along a ring (or wrap-around links in a torus network). Once this state is detected as being reached, packet $E1$ is denied access to Queue 4 as shown in Figure 13.10(b); otherwise, knotted resources would become fully occupied as shown in Figure 13.10(c). While intuitive and straightforward to conceive of in theory, verifying the existence of a

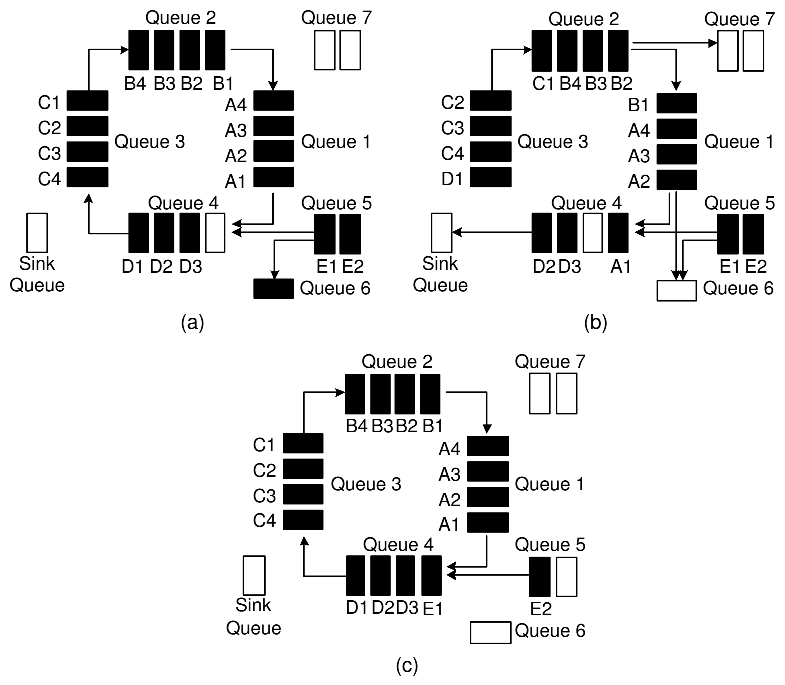


Figure 13.10. Importance of controlling bubble flow in one of possibly many logical (virtual channel) networks with adaptive routing. Only queue resources and dependencies on them by packets are shown. (a) Packets A1 and E1 compete for a bubble in Queue 4, the only one in the knotted dependency cycle consisting of Queues 1–4. Bubbles in other queues are not available; that is, they are not supplied by the routing function for A1, B1, and D1, shown as arcs. When available, a bubble in Queue 6 can be used by packet E1. (b) If the bubble is allocated to A1, deadlock is prevented as the knot keeps the bubble. Assuming the bubble traverses the cycle, new head packets establish a different dependency pattern, making other bubbles available, i.e., packet D2 sinks; A2 and B2 have additional routing options; and E1 can use the bubble in Queue 6. (c) If allocated to E1, the bubble exits the cycle, causing knotted resources to become fully occupied and deadlock to ensue.

single bubble within all possible knots may be difficult to implement in practice. One possible mechanism for identifying cycles nearing full occupancy is described in Section D..

An example of a scheme based partially on this technique is the hybrid path/channel-based Adaptive Bubble Routing scheme [26]. Injection limitation is used to implement the escape routing function in the weak sense *within each* *dimension* of a torus network, thus requiring only one virtual channel for escape routing instead of two. However, escape routing across dimensions is still implemented using dimension order routing, thus disallowing the formation of inter-dimensional knots. Because of this, this scheme places some restrictions on routing freedom, which keeps it from being true fully adaptive. This scheme can, therefore, be said to be hybrid in another respect: it avoids deadlock in the wide sense for inter-dimensional escape routes and avoids deadlock in the weak sense for intra-dimensional escape routes. Although packet injection and flow into each network dimensional ring is allowed, in theory, if as few as only two free buffers exist *along the ring*, gathering the global status information needed to enforce this condition may be costly to implement in practice (the pinging mechanism described in Section D. may facilitate this). As a simplification, the decision to inject packets into dimensional rings could be made locally by requiring that at least two free buffers exist in the appropriate escape queue *at each router* in the network, as is implemented in the IBM BlueGene/L supercomputer [41].

An alternative way of avoiding routing-induced deadlock in the weak sense is to allow the full network capacity always to be supplied to packets via controlled deflection of packets out of cyclic or knotted dependencies, consequently through misrouting. That is, there is always at least one packet in each knot that is supplied some nonminimal path to reach its destination as long as the input degree (# of ports) of each router is equal to the output degree. This is done through *deflection rerouting*. Since injection and delivery ports are included in the input/output degree of a router, nonminimal paths could include delivery into attached nodes that are not the ultimate destination of a packet, as is done in the In-Transit Buffer scheme [45] and Hot Potato Routing [46]. This would be the case for packet $D1$ in Figure 13.10(c) if it were deflected into the sink queue. Alternatively, packet $B1$ could be deflected into Queue 7, which also happens not to be along a minimal path but does have several bubbles (empty queue buffers) available to route packets. A technique similar to this is used in the hybrid path/queue-based true fully adaptive Chaotic Routing scheme [47] implemented in the ChaosLAN [48]. The availability of bubbles is regulated using a packet exchange protocol, which allows a packet from router i to be deflected to router j as long as router i has queue space to receive a packet from router j. This ensures that bubbles are always available for deflecting packets between neighboring routers.

With both techniques, some bubble(s) are always available to any set of network resources on which a path of cyclic or knotted dependencies can form, thus averting routing-induced deadlock. All blocked packets along resource dependency chains or cycles are able to shift forward by at least one buffer position within queues by consuming bubbles. This operation may not immediately remove packets from cyclic or knotted dependencies, but it does reduce the probability of sustaining those dependency relations. This is because new routing candidates arising from the shift may supply bubbles to those packets involved. That is, after the shift, some packets reaching the head of the queues which now become eligible for routing may then be able to use alternative resources or sink at their destination, as illustrated in Figure 13.10(b). This decreases the coupling among packets involved in dependency relations on congested resources. In effect, this provides a way for packets to disperse out of those areas, which is the main advantages of maximizing routing freedom.

Some of the disadvantages of these techniques, however, are that they are applicable only to virtual cut-through networks, not wormhole networks. Also, limiting injection into some resources prevents the network's full capacity from being utilized, and it could cause some packets never to be granted access to needed resources—a situation commonly referred to as *starvation*. For example, in Figure 13.10(a), packet $E1$ might remain indefinitely in the network waiting endlessly on Queue 4 and Queue 6 resources if packets in Queue 1 are always granted bubbles which may appear in those queues and other packets are continuously injected into the cycle at other queues. Moreover, nonminimal routing can wastefully consume scarce network bandwidth on each deflection, causing the network to saturate earlier. Unless controlled, deflection routing might also result in packets continuing to bounce around in the network indefinitely, never reaching their destinations—a situation commonly referred to as *livelock*. Many of these problems can be mitigated with some of the recovery-based routing approaches discussed below.

D. Routing-Induced Deadlock Recovery

Deadlock recovery routing approaches aim to optimize routing performance in the absence of deadlock. This is achieved by allowing unrestricted, true fully-adaptive routing on all physical and virtual channels and efficiently handling impending deadlock if it occurs using minimal resources. This is in contrast to deadlock avoidance routing schemes which typically devote a greater number of physical or virtual channel resources for avoiding improbable deadlock situations. In avoidance-based schemes, a set of escape resources are always supplied by the routing function whereas nominal resources (if any) are supplied in recovery-based schemes and only when potential deadlock situations are detected. Another distinguishing factor is that physical resources (i.e., link bandwidth) may be deallocated from normal packets in some progressive recovery-based schemes whereas no such deallocation occurs in any of the avoidance based schemes. In general, the viability of recovering from routing-induced deadlock depends critically on the recovery overhead, which is determined by the frequency with which deadlock is detected and, as importantly, the costs associated with resolving deadlock.

Regressive recovery routing schemes kill at least one packet detected as participating in potential deadlock so as to create a bubble(s) along knotted resources, as described in Section C. earlier. For example, in Figure 13.10(c), if packet $E1$ were killed, the unfilled cycle shown in Figure 13.10(a) (with $E1$ removed from Queue 5 and $E2$ put in its place) would come about, which resolves the deadlock. Aggressively killing more than just one packet in the cycle (i.e., all the packets at the end of Queues 1–4) reduces the probability of subsequent potential deadlocks occurring that involve those packets remaining in the cycle. In order for the source node that generated a killed packet to know it must re-inject the killed packet, some form of a "tether" must trace the packet's progress back to its source. This can be done in wormhole networks by using packet padding flits as is done in the Compressionless Router [49] or, in general, by using a combination of end-to-end acknowledgements and timeouts implemented by control signals of a higher network protocol layer. The additional latency, bandwidth overhead and control complexity needed to implement the killing and re-injection process increase cost and reduce network performance.

Proposed deflective recovery routing schemes operate on the same principle as deflective avoidance-based schemes described in the previous section once potential deadlock is detected, but with one major distinction: bubbles are not guaranteed always to be available at all neighboring routers. Instead, bubble movement in the network can be made to be stochastic (i.e., random) due to bubbles being subjected to causal motion: bubbles are allocated randomly to normal packets when multiple packets compete to consume them at each router. In addition to this, any bubble in the vicinity of a router participating in the knotted set of resources gets "sucked in" since potentially deadlocked packets that are to be deflected have priority over normal packets. With this, bubbles are guaranteed to propagate randomly to needed areas in the network and be used by packets in need of deflection eventually (in finite time) to resolve deadlock. This kind of guarantee on the arrival of bubbles to potentially deadlocked packets can be thought of as the dual case of the probabilistic guarantee on the arrival of packets to their destinations provided by the Chaos Router for resolving livelock. Hole-based routing [50] is one proposed deflective recovery routing scheme which follows this approach. Software-based Recovery [51] is a deflective recovery scheme that is less dependent on stochastic bubble movement within the network and more dependent on the deterministic availability of bubbles at network endpoints (e.g., like the In-Transit Buffer scheme). Although deflective recovery routing lifts certain requirements on packet exchanges between neighboring routers, it suffers from the other disadvantages common to deflective avoidance-based schemes mentioned previously.

In contrast to regressive and deflective recovery, progressive deadlock recovery is based on the idea of providing access to a connected, deadlock-free recovery path through which an eligible packet is progressively routed out of the knotted cyclic dependency to resolve deadlock. The deadlock-free recovery path need not be implemented using physical paths or virtual channels (edge queues) devoted specifically for this purpose. Instead, a special *deadlock buffer* central to each router and accessible from all neighboring routers can be used as a "floating" internal channel shared by all physical dimensions of a router to accomplish the same. Such a deadlock buffer is shown in Figure 13.2 as the central queue. The size of the buffer depends on the switching technique used, i.e., flit-sized for

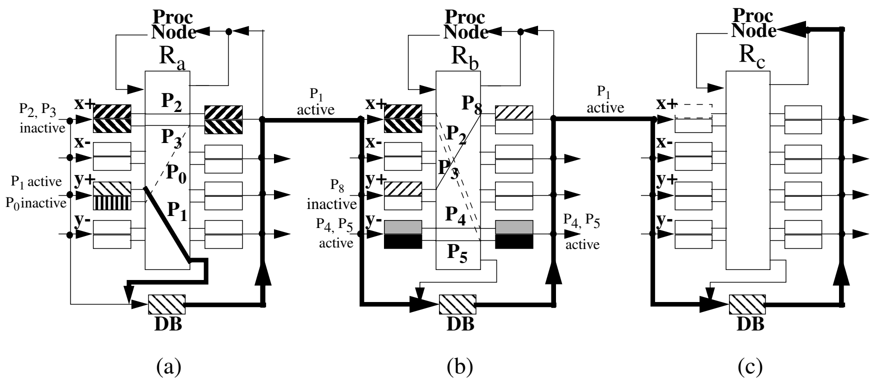
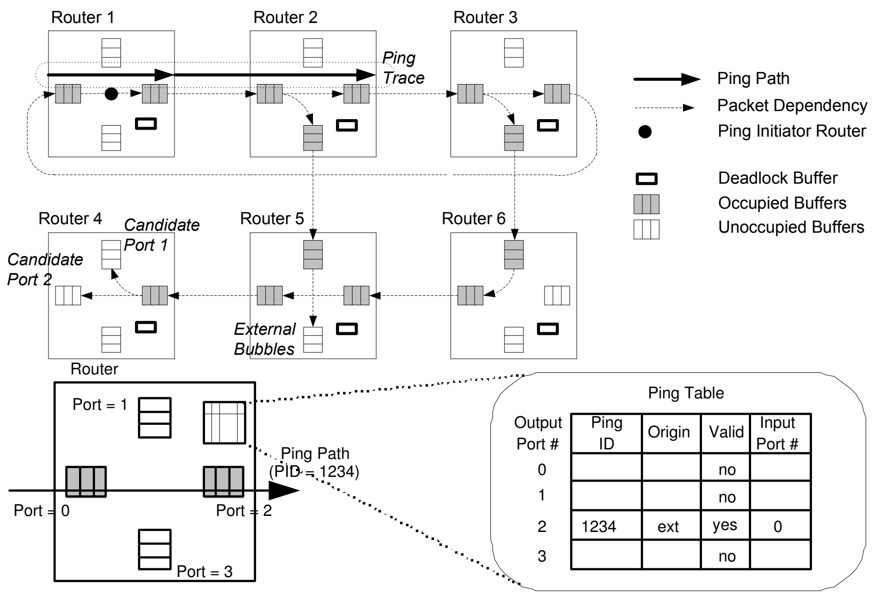


Figure 13.11. Dynamic recovery path formation and progressive deadlock resolution in *Disha*. Packet P_1 at router R_a recovers by routing through a recovery lane consisting of deadlock buffers (DBs) at R_a, R_b, and finally R_c which is its destination node. P_1 preempts physical channel bandwidth from P_8, suspending normal packet routing until deadlock is recovered. (From Ref. 56 ©2003 IEEE.)

wormhole switching and packet-sized for virtual cut-through switching. Deadlock buffers can be used to progressively resolve routing-induced deadlock either in the wide sense or in the weak sense, as explained below.

A well-known progressive recovery technique is *Disha* [52] used in the *WARRP* router [53]. In forming an escape recovery path system wide, deadlock buffers are used to route eligible packets minimally to their destinations. Packets in deadlock buffers preempt packets in normal resources so that network bandwidth is dynamically allocated for resolving deadlock if it occurs on rare occasions. As the tail of a recovering packet passes, network bandwidth is reallocated to the preempted packets. The recovery path is made deadlock-free by enforcing mutually exclusive access to it (i.e., *Disha Sequential* [52]) or by structuring routing on it to enforce some total ordering on resources such that cyclic dependencies are prohibited (i.e., *Disha Concurrent* [54]). One way of enforcing mutual exclusion is to use a circulating token which visits all routers in a predetermined cyclic path and is captured by a router participating in a potential deadlock [55]. The router progressively routes a recovering packet over a preempted output physical channel while having the corresponding control line set to indicate that this packet should be directed to the deadlock buffer at the next router in sequence, as illustrated in Figure 13.11. A new token is released by the destination node once the packet header is received. As there exists at least one packet in each deadlock that can fairly gain access to and route progressively on the recovery path which is connected and deadlock-free, this routing scheme safely recovers



Output Port #	Ping ID	Origin	Valid	Input Port #
0			no	
1			no	
2	1234	ext	yes	0
3			no	

Figure 13.12. Illustration of packet dependency and ping movement for detecting congestion (top). Possible organization of router ping table for recording path reservation information (bottom).

from all deadlock [56]. Thus, we can think of the Disha scheme as using deadlock buffers to progressively resolve routing-induced deadlock in the wide sense.

The *Ping and Bubble* scheme is a more recently proposed progressive recovery technique [57]. The idea behind this scheme is to correctly trace all cyclic dependencies on network resources *in real time* and to dynamically supply a bubble to those resources and force it to traverse the entire cycle. Small control packets called "pings" are used to globally probe only those resources suspected of participating in potential deadlock once detected as possibly occurring. This works as follows. A ping is generated by a router for a given input port when certain deadlock precipitating conditions are detected, e.g., due to its buffer being occupied for longer than some threshold amount of time or some other criteria. It is then sent through one of the output ports requested by the packet at the head of that input port queue in order to probe the network for possible cyclic dependency. When a ping arrives at a neighboring router port, it is further propagated if the deadlock precipitating conditions are met there as well, and that router port queue is added to a path or *trace* of such dependent resources. This is illustrated in Figure 13.12 for an example scenario. In addition, a reservation is made in a ping table at that router so that any arriving bubble will be allocated to the pinged port. The ping table allows for precise control over bubble movement so that bubbles remain within resources along the dependency cycle and are coerced to propagate backward along the trace path. Continuing, if the ping returns to the same router port which initiated it, a cycle of dependency is detected on which a set of resources have reserved the allocation of an arriving bubble.

Along with the pinging mechanism, the special deadlock buffer is used to temporarily "shelter" a culprit packet at the head of the input port queue identified as closing a dependency cycle. This removes the packet from the deadlock cycle, creating an "escape" bubble in that queue which starts to propagate backward along the ping trace. As at most one router (the one with the highest ranking ping) detects each cycle and all non-overlapping cycles are detected, a bubble is guaranteed to be made available to resources along the cycle. Also, since bubble movement is totally controlled by the ping table entries in other routers along the detected cyclic trace path, the escape bubble generated by sheltering the culprit packet is guaranteed to arrive back at that router in finite time and can be consumed by the packet in the special deadlock buffer. This frees the deadlock buffer for future use. Once the returned bubble is allocated to the sheltered packet, all other packets along the dependency cycle will have already been shifted progressively forward by at least one position, possibly breaking the cycle. In the unlikely event that the same cycle recurs due to no change in the dependency relation among new routing candidates at the head of the queues, the procedure repeats. As packets are routed along minimal paths, at least one packet will eventually reach its destination in finite time even if the dependency relation persists until then. Thus, we can think of the Ping and Bubble scheme as using deadlock buffers to progressively resolve routing-induced deadlock in the weak sense.

Operation of the Ping and Bubble scheme is illustrated using the simplistic example given in Figure 13.10(c). Assuming that a potential deadlock is detected by packets at the head of Queue 1 and Queue 5. A ping is generated by the router for either Queue 1 or Queue 5 (assume both are associated with the same router) and transmitted to Queue 4 in the neighboring router. Subsequently, the ping is transmitted to Queue 3, then Queue 2, and returns back to Queue 1 following the path supplied by the routing function for the packets at the head of those fully occupied queues. No matter whether the ping was generated by Queue 1 or Queue 5, Queue 1 is the one identified as closing a cycle of resource dependencies, like the cyclic path shown in Figure 13.12. Alternatively, other pings could simultaneously have been generated by the other queues in the cycle. If not squashed by an outranking ping generated by a different queue that alone would detect the cycle, those pings would follow the same cyclic path and return to their initiating router—identifying one of those other queues as closing the dependency cycle.

Preliminary results show that substantial performance gains are possible. As shown in Figure 13.13, the proposed Ping and Bubble scheme can sustain near maximum throughput even under heavily loaded network conditions, yielding twice the throughput that can be sustained by Duato's Protocol and Disha. All

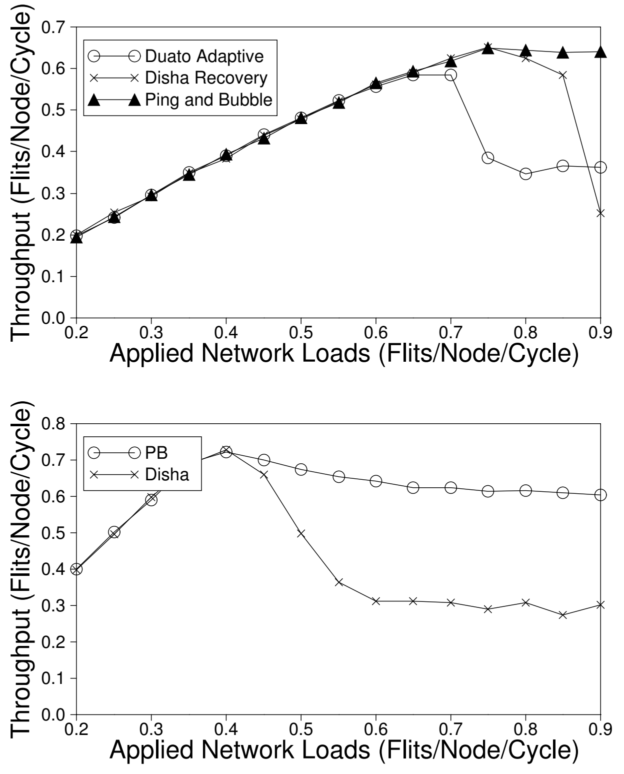


Figure 13.13. Network throughput for an 8×8 bidirectional torus with 4 virtual channels for the proposed Ping and Bubble scheme compared to Disha Sequential and Duato's Protocol (top) and Ping and Bubble (PB) compared to Disha for a network with an increased bristling factor of 2, i.e., two injecting nodes attached to each router in the network (bottom).

the while, routing-induced deadlock is effectively resolved. In addition to the increased routing freedom it has over Duato's Protocol, the main reason for the Ping and Bubble scheme's improved performance is that escape bubbles are made more accessible to packets needing them most under saturated network conditions (i.e., packets involved in cyclic dependencies). With Duato's Protocol and Disha, the escape and recovery paths can become a performance bottleneck; there is no such bottleneck with the Ping and Bubble scheme.

13.3.2 Handling Message-Induced Deadlocks

A. Message-Induced Deadlock Avoidance Techniques

Message-induced deadlock can be avoided in the weak sense by providing enough buffer space in each node's network interface queues to hold at least as many messages as can be supplied, as is done in [58,59], for example. Even though knotted dependencies on a set of resources may exist, deadlock never actually occurs as the set never becomes fully occupied. This guarantees that packets can sink at their destinations eventually. Although simple to implement, this technique is not very scalable since the size of the network interface queues grows as $O(P \times M)$ messages, where P is the number of processor nodes and M is the number of outstanding messages allowed by each node in the system.

At the opposite extreme, message-induced deadlock can be avoided in the strict sense by providing logically independent communication networks for each message type (implemented as either physical or virtual channels) and restricting the use of those networks based on message type, as is done in [28,37,39,40], for example. The partial ordering on message dependencies defined by the communication protocol is transferred to the logical networks such that the usage of network resources is acyclic. This restriction on routing freedom guarantees that no deadlock can form due to message dependencies. Figures 13.14 illustrates this for the four node ring system shown previously in Figure 13.6 and 13.9(c), but this time with four virtual channels per physical channel. With two logical networks each consisting of two virtual channels to avoid routing-induced deadlocks (see Section A.), this network is able to avoid message-induced deadlocks for message dependency chains of length two, i.e., *request* $\prec$ *reply*, but no greater.

For message-induced deadlock handling techniques based on enforcing routing restrictions, the size of network resources does not influence deadlock properties, making such techniques more scalable. Because of this, these are the more commonly used, but they still suffer the cost disadvantage of requiring at least as many logical networks as the length of the message dependency chain. Such partitioning of network resources not only increases network cost but also decreases potential resource utilization and overall performance, particularly when message dependencies are abundant and resources (i.e., virtual channels) are scarce. For example, the Alpha 21364 processor/router chip [40] used in AlphaServer systems requires seven logically separated networks to avoid message-induced deadlocks on seven message types. For six of these, two virtual channels are required to escape from routing-induced deadlocks in a torus network and an additional one is used for adaptive routing. Therefore, of the 19 total virtual channels implemented, at most only two can be used at any given time by any given message type. This is a severe limitation on routing freedom.

In general, routing freedom at any given point in the network is limited to $(1 + (C/L - E_r))$ of the C virtual channels on a link, where L is the message dependency chain length, E_r is the minimum number of virtual channels required

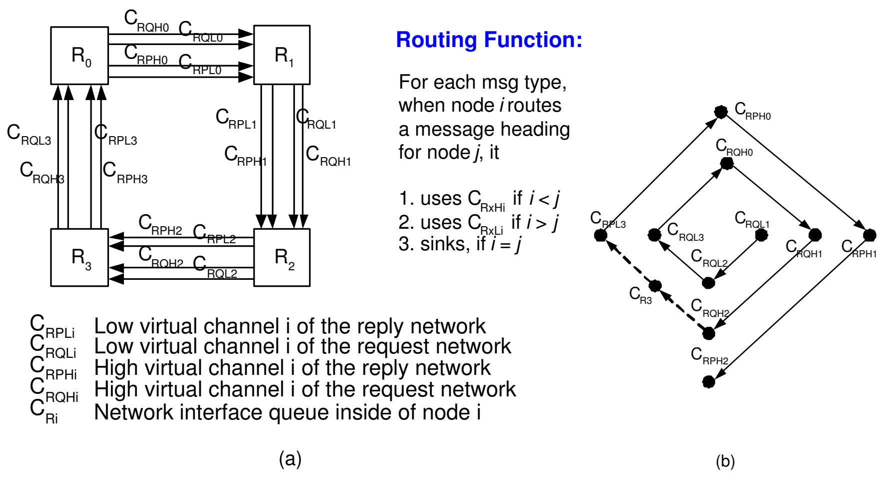

Routing Function:
For each msg type, when node i routes a message heading for node j, it
1. uses C_RxHi if i < j
2. uses C_RxLi if i > j
3. sinks, if i = j
C_RPLi Low virtual channel i of the reply network
C_RQLi Low virtual channel i of the request network
C_RPHi High virtual channel i of the reply network
C_RQHi High virtual channel i of the request network
C_Ri Network interface queue inside of node i
R0
R1
R2
R3
(a)
(b)

Figure 13.14. (a) Separation of request and reply networks avoids cyclic dependencies in the channel dependency graph shown in (b), but it reduces channel utilization. Shown in (b) is the case in which R1 is the requester and R3 is the responder. (From Ref. 61 ©2003 IEEE.)

to escape from routing-induced deadlock for a given network, $E_m = L \times E_r$ is the minimum number of virtual channels required to escape from message-induced deadlock for a given network, and $C \geq E_m$. Routing freedom can be increased if message-induced deadlock is, instead, avoided in the wide sense by allowing all channels other than the minimum number required to escape from message-induced deadlock to be shared amongst all message types, as proposed in [60]. That is, cycles among message-dependent resources are allowed as long as it is always possible to reach some set of escape resources on which no cyclic message dependencies can exist. With this technique, the upper limit on virtual channel availability is increased to $(1 + (C - E_m))$. Nevertheless, restrictions enforced on escape channels due to only one channel being supplied by the routing function out of the E_m channels acts to limit the overall potential performance.

B. Message-Induced Deadlock Recovery Techniques

Evidently, the main disadvantage of avoiding deadlock by disallowing cyclic dependencies on escape resources is the number of partitioned logical networks required. It is possible to reduce the number of partitions by allowing different message types to use the same logical network and removing message(s) from cyclic dependencies only when a potential deadlock situation is detected. Since a detection mechanism and recovery action are required to resolve the potential deadlock situation, this technique for handling message-induced deadlock is said to be based on deadlock recovery. Detected deadlocks can be resolved by killing and later re-injecting packets in network interface queues (i.e., *regressive recovery*), deflecting packets out of resources involved in cyclic dependency by converting them from a nonterminating message type to a terminating message type (i.e., *deflective recovery*), or by progressively routing packets using resources along a path that is guaranteed to sink (i.e., *progressive recovery*). The actions taken by regressive and deflective recovery increase the number of messages needed to complete each data transaction, whereas progressive recovery does not; all packets make progress toward their destinations and never degeneratively regress or deflect.

The differences between the above techniques are illustrated using an example cache coherence protocol that could be used in a multiprocessor system which permits generic message dependency chains shown in Figure 13.15(b). In this protocol, an original request message ($m_1 = ORQ$) arriving at some home node is forwarded to the owner or sharers as a forwarded-request message ($m_3 = FRQ$) before being responded to by a terminating reply message ($m_4 = TRP$) if the home node is unable to fulfill the request. Thus, the length of the message dependency chain for data transactions can be two ($ORQ \prec TRP$) or three ($ORQ \prec FRQ \prec TRP$), depending upon where the requested data is located. Message-induced deadlocks can be avoided in the strict sense by partitioning resources into three separate logical networks, one for each of these message types. However, to reduce the number of logical networks to two, both ORQ and FRQ messages may be allowed to use the same *request network* and TRP messages can use a separate *reply network*. Since message-induced deadlocks can now potentially form on the request network, they must be resolved once detected.

Consider first how resolution might be done with a deflective recovery scheme such as the one used in the SGI Origin 2000 multiprocessor [29], shown in Figure 13.15(a). If a potential deadlock situation is detected at a home node, the node takes ORQ messages that would generate FRQ messages from the head of the input request queue and deflects back to the requesters "backoff" reply message types ($m_2 = BRP$). These messages contain owner or sharer information that allows the requester to generate FRQ message(s) directly to the intended target(s) without further intervention from the home node. They are additional messages needed to carry out the data transaction. That is, with this protocol, the $ORQ \prec FRQ \prec TRP$ message dependency chain is converted into a $ORQ \prec BRP \prec FRQ \prec TRP$ chain when potential message-induced deadlock is detected. Although the message dependency chain length is increased during recovery, the number of logical networks implemented need not increase; it remains at two. In order for this to happen, the system allows BRP messages to use the same reply network as TRP messages and avoids message-induced deadlock in the weak sense on the reply network by dynamically reserving (preallocating) sufficient space in the incoming reply queue of the requester in order to be able to sink responses for all outstanding ORQ messages. This reservation is made before ORQ messages enter the network.

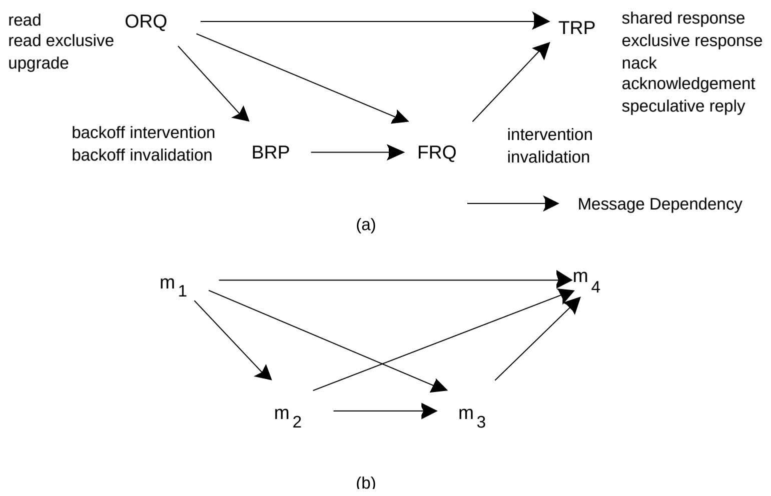


Figure 13.15. (a) The ordering among message types (shown in uppercase) and message subtypes (shown in lowercase) in the Origin 2000. Note that BRP occurs only if a message-induced deadlock is detected, otherwise the maximum chain length is three. (b) The ordering among message types (m_i) for a generic cache coherence protocol with four message types. (From Ref. 61 ©2003 IEEE.)

By relaxing restrictions in the way in which network resources are used, resources can be utilized more efficiently and provide increased network performance. However, as potential deadlock situations mainly occur when the system nears a saturated state, resolving potential deadlock situations by increasing the number of messages required to complete data transactions only exacerbates the problem. Progressive recovery techniques resolve potential deadlock situations more efficiently by using the same number of messages as are required to avoid deadlock. One such recently proposed technique is derived from the *Disha* technique described in Section D..

The proposed technique, referred to as *mDisha* [61], extends the notion of *Disha-Sequential* recovery paths existing only between network endpoints to one that includes network endpoints as well, as shown in Figure 13.16. Hence, the circulating token must also visit all network interfaces attached to each router node, and a deadlock buffer (referred to as a deadlock message buffer or DMB) must also be provided in each network interface. The size of the DMB is determined by the minimum unit of information on which end-to-end error detection/protection (i.e., ECC, checksum) is performed. Typically, this is at the packet level, requiring these deadlock buffers to be at least packet-sized. This is also the minimum size of the network interface input and output queues; however, larger input/output queues would typically be used to increase performance. All network resources

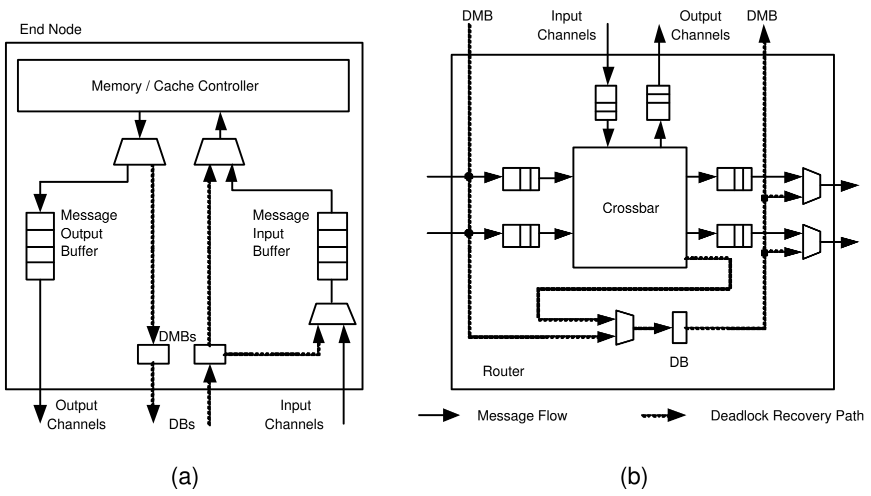


Figure 13.16. Network interface and router architecture for *mDisha* progressive recovery of message-induced deadlock. (From Ref. 61 ©2003 IEEE.)

can be completely shared, independent of the message types. Relaxing resource allocation and routing restrictions in this way maximizes the utilization of resources while allowing for all potential message-induced deadlocks to be recovered once detected as long as the same token is reused to recover all subordinate messages along a given message dependency chain until one of them sinks. The progressive recovery process is guaranteed to terminate since each message dependency chain is acyclic and has a terminating message type. However, like its *Disha-Sequential* predecessor, limitations of this scheme are its single point-of-failure due to the token mechanism and the sequential nature of recovery, which should not be a problem since the frequency of message-induced deadlocks typically is low. However, it is necessary to have a reliable token management mechanism, i.e., one that can be transmitted as an in-band control packet multiplexed with data packets over network channels. This way, the path taken by the token can be logical and, thus, reconfigurable for increased reliability.

13.3.3 Handling Reconfiguration-Induced Deadlocks

Traditional approaches for reconfiguring a network's routing function are based either on avoiding deadlocks in the strict sense or on regressively recovering from them. Both approaches rely mainly on dropping packets. Static reconfiguration techniques, for example, consist of first stopping and flushing *all* user traffic from the network before commencing and completing network-wide reconfiguration [3, 25,62]. Network flushing is typically done by actively *discarding* all nondelivered packets not yet reaching their destination nodes. In addition to this, the injection of packets into the network is halted during the entire reconfiguration process until it completes. This allows the routing function to be updated from old to new in one atomic action from the point-of-view of user packets, leading to the notion of *static* reconfiguration. As no packets are in the network at the time that the new routing function becomes active, no reconfiguration-induced cycles can form, thus strictly avoiding deadlocks. Alternatively, dynamic reconfiguration techniques allow reconfiguration to commence while user packets remain in the network and continue to be injected into the network. Reconfiguration-induced deadlock arising from dynamic interactions between packets routed by old and new routing functions during and after *dynamic* reconfiguration can be resolved regressively by reactively discarding packets in the event that buffers become full [1,3,63]. This simple approach is most applicable to systems implementing soft link-level flow control in which buffer overflow is solved through packet dropping, such as in wide-area networks. However, it is less applicable to systems that implement hard link-level flow control in which packet flow is regulated by means other than packet dropping, which is the case for most multiprocessor interconnects (including SIO, IPC, and NOCs), IPRF technologies, and many switched LAN/SAN/STAN technologies.

With both the strict avoidance and the regressive recovery-based approaches, a considerable number of packets may be dropped, possibly requiring upper layer protocols to be invoked and system state to be rolled back to ensure correct execution. This exacerbates the problem of providing real-time and quality-of-service support needed by some applications which have limited tolerance for recurring performance drop-offs. Moreover, disallowing packet injection during the reconfiguration process further degrades performance. For example, the transmission of video streams in a high-definition video-on-demand server would be halted during reconfiguration, leading to frozen frames for an undesirable, perhaps unacceptable, period of time. Similarly, when an interconnection network is used as the switch fabric within IPRFs [4], typically no packet dropping is allowed within a router. And since these switch fabrics are designed to operate close to their saturation point, the number of dropped packets using such degenerative approaches could be excessively high.

While no networking technology can guarantee that all packets will reach their destinations under all conditions, packet dropping should be the exception rather than the rule. Much research is currently being done on designing dynamic reconfiguration techniques implemented with hard link-level flow that do not halt the injection of user packets into the network nor rely on dropping user packets from the network before, during, or after the reconfiguration process. Such schemes aim to minimize restrictions on packet injection and delivery throughout the reconfiguration process. In addition, they aim to exploit the fact that reconfigurations often require only a few paths in the network to be changed (i.e., only those within the skyline [64], as shown in Figure 13.7), thus affecting only a few packets and router switches. A few such techniques have recently been proposed based on avoiding reconfiguration-induced deadlock in the wide sense. Deadlock is avoided by enforcing some total ordering on the usage of escape resources and remains consistent when taking into account the interactions of both the old and new routing functions. This is done by some form of logical separation of resources used to escape from deadlock by packets experiencing old and new routing functions and is accompanied by some form of step-wise or partial update of the routing function.

A. Reconfiguration-Induced Deadlock Avoidance in the Wide Sense

The basic idea behind techniques based on avoiding reconfiguration-induced deadlock in the wide sense is the following: it is possible to consider the reconfiguration of a routing function as a change from an old routing function R_{old} to a new one R_{new} in a sequence of k steps ($R_{old} = R_0 \rightarrow R_1 \rightarrow ... \rightarrow R_k = R_{new}$) that can be completed and k associated conditions ($cond_1, ..., cond_k$) that can be fulfilled. Every step contains one or more updates to the routing function in the previous step. At each router, multiple updates within each step may be carried out in any order (or it can be assumed that all updates complete in one atomic action), but steps are sequentialized so that in updating the routing function from R_{i-1} to R_i, $step_i$ cannot start before $step_{i-1}$ completes and condition $cond_i$ is fulfilled. Network-wide, however, the completion of steps is not necessarily synchronized unless specified by a condition. So, it is possible for $step_i$ to be completed at a router before $step_{i-1}$ is completed at another router if condition $cond_i$ does not require synchronization. The conditions are usually related to packets in the network; they determine in what ways the dependencies allowed by routing functions of previous and current steps can interact with the dependencies allowed by routing functions of current and future steps. That is, conditions are used to "filter out" unwanted ghost dependencies on resources that may cause deadlock. For example, if at some step, $step_i$: $R_{i-1} \rightarrow R_i$, the condition specifies that some set of network resources is required to be empty before the start of that step, then the old dependencies (in particular, ghost dependencies) allowed by all previous routing functions before R_i on those resources can simply be ignored. Therefore, it is possible to define a series of reconfiguration steps that impose a minimum set of conditions to eliminate harmful ghost dependencies—those which could result in permanent deadlock. If the final step in the series of reconfiguration steps can be reached and provides a routing subfunction that is connected and deadlock-free over an escape set of resources taking into account all remaining ghost and normal dependencies from previous and current steps, the reconfiguration protocol is provably deadlock-free [24].

One recently proposed dynamic reconfiguration scheme is the Partial Progressive Reconfiguration (PPR) scheme [65]. The PPR scheme systematically performs sequences of partial updates to routing tables which implement the routing function, progressively removing old and adding new entries until all routing tables are completely updated to the new routing function. This is done by sequentially moving *root* nodes (i.e., nodes that only have links with up ends connected to them, as shown in Figure 13.7) and *break* nodes (i.e., nodes that have two or more links with only down ends connected to them, as shown in Figure 13.7) one router position at a time to their final positions by partially updating entries in the routing tables along the path. To avoid deadlock, each switch must synchronize with some of its neighbors after each partial update of a break node movement in order to ensure that no dependency cycles form along escape resources. The existence of the break node is what guarantees that cycles cannot be completed, i.e., they act to break the cycle. The required synchronizing steps on break node partial updates increases implementation complexity, and some link changes on the escape set of resources may render some packets unroutable. Unroutable packets are those that reach a point in the network for which no legal route is supplied. These packets must be discarded in order to avoid permanent blocking that results in reconfiguration-induced deadlock.

Another set of recently proposed dynamic reconfiguration schemes is the *Double Scheme* [24], which avoids reconfiguration-induced deadlocks in the wide sense by *spatially* separating resource allocations that may permanently close dependency cycles along escape paths[4]. The most straightforward way of accomplishing deadlock-free spatial separation is simply to double the number of resources used by a routing algorithm to escape from deadlock and to allow dependencies only from one set to the other but not from both at any given time. This can be implemented by using two distinct sets of physical or virtual channels: one set is used by the current routing function (*old* with respect to the next reconfiguration) to escape deadlock and the other set is used by the next routing function (*new* with respect to the next reconfiguration) to escape deadlock. Consistent with Duato's Protocol, an escape routing subfunction is defined on each set of channels in such a way as to be connected and deadlock-free for its corresponding network, i.e., the old network before reconfiguration or the new one afterwards. Of the four varieties of the *Double Scheme* mentioned in [24], we discuss only the *Enhanced Basic Double Scheme* below.

B. The Enhanced Basic Double Scheme

The *Enhanced Basic Double Scheme* requires two sets of network channels for escape routing. The routing function is composed of three routing subfunctions: $R = R_I \circ R_{E_1} \circ R_{E_2}$, where R_{E_1} and R_{E_2} are two escape routing subfunctions defined on E_1 and E_2 channel sets, respectively, and R_I is the injection routing

[4]As an alternative to spatial separation of resource allocations, the *Single Scheme* [66] enforces *temporal* separation only on the one set of escape resources (as opposed to strict avoidance static schemes which enforce temporal separation on all resources, including non-escape resources).

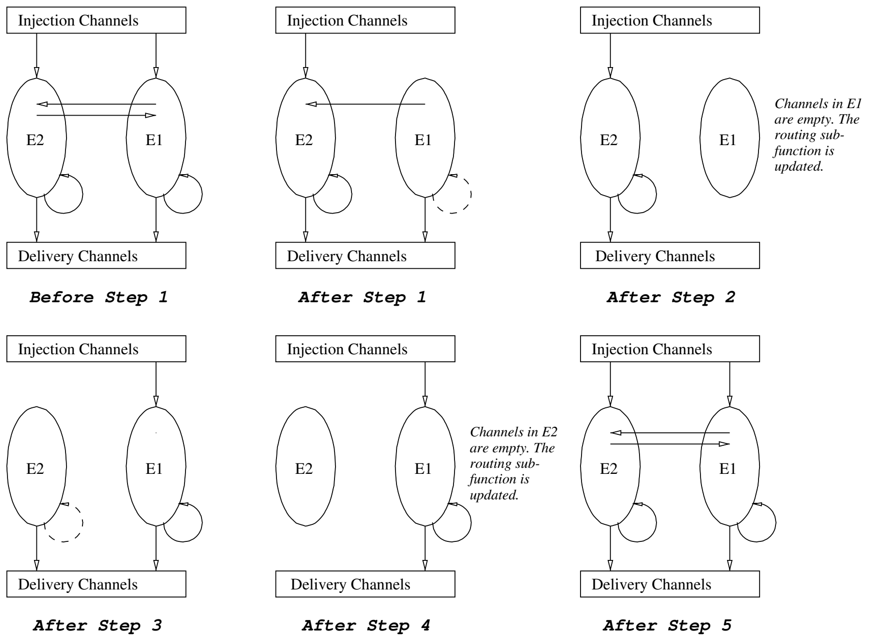

Injection Channels
E2
E1
Delivery Channels
Before Step 1
After Step 1
After Step 2
Channels in E1 are empty. The routing sub-function is updated.
After Step 3
After Step 4
Channels in E2 are empty. The routing sub-function is updated.
After Step 5

Figure 13.17. Illustration of the possible channel allocations occurring in the network for the *Enhanced Basic Double Scheme*. The dashed arc signifies that packets in a given escape set may continue to use those resources to drain from the network.

subfunction. Before and after reconfiguration, the routing function allows packets to route using both escape channel sets interchangeably. That is, the duplicate set of escape resources are always fully and efficiently used outside of reconfiguration, allowing no performance degradation to be suffered during normal network operation. During reconfiguration, however, packets are restricted to using only the escape channel set into which the packet is injected. This allows the routing subfunction defined on one set to be reconfigured while packets continue to be injected into the other set. Once the routing subfunctions defined on both sets are reconfigured, this restriction is relaxed. The five-step dynamic reconfiguration protocol implemented at each router is detailed below, and possible channel allocations resulting from it are illustrated in Figure 13.17.

THE ENHANCED BASIC DOUBLE SCHEME

Condition 1: A topology change is detected at the router or a reconfiguration notification is received.

Step 1: The routing function is changed ($R_{old} = R_0 \rightarrow R_1$) locally at the router such that R_I supplies only E_2 channels; R_{E_2} supplies only E_2 and delivery channels; and R_{E_1} and R_{E_2} are modified to discard only those packets encountering disconnectivity (if any).

Condition 2: All packets are drained from E_1 channels, and all new topology/routing information is acquired.

Step 2: The routing function is changed ($R_1 \rightarrow R_2$) locally at the router such that R_{E_1} is updated to supply only E_1 and delivery channels for the new escape routing subfunction. R_{E_2} continues to supply only E_2 and delivery channels, and it continues to discard only those packets encountering disconnectivity (if any).

Condition 3: All routers neighboring the router complete Step 2.

Step 3: The routing function is changed ($R_2 \rightarrow R_3$) locally at the router such that R_I supplies only E_1 channels.

Condition 4: All packets in E_2 channels that can route through the router are drained.

Step 4: The routing function is changed ($R_3 \rightarrow R_4$) locally at the router such that R_{E_2} is updated to supply E_1, E_2, and delivery channels for the new escape routing subfunction.

Condition 5: All routers neighboring the router complete Step 4.

Step 5: The routing function is changed ($R_4 \rightarrow R_5 = R_{new}$) locally at the router such that R_I supplies E_1 and E_2 channels. Also, R_{E_1} is modified to supply E_2 channels in addition to E_1 and delivery channels for the new escape routing subfunction.

Reconfiguration is completed once all routers complete this step.

Like all of the *Double Schemes*, the *Enhanced Basic Double Scheme* is based on the notion that an *empty* set of escape channels becomes available on which deadlock-free routing for the new network can take place, but it requires double the minimum number of resources as compared to static reconfiguration. Its advantage, however, is its lack of need to flush the entire network in order to nullify the effects of potentially harmful ghost dependencies. Instead, only a select subset of channels need to be drained. Packets do not need to be discarded in order to accomplish this drainage. All packets not encountering fault-induced disconnectivity are allowed to route normally in the network until draining at their destinations. That is, no reconfiguration-induced packet dropping is necessary (as may be the case with PPR) as no packets become unroutable on account of the reconfiguration algorithm. As injection of new packets into the network is allowed to continue during each reconfiguration step, the reconfiguration process is dynamic. Furthermore, as all future steps are reachable from each reconfiguration step and the routing function in the last step is deadlock-free, the dynamic reconfiguration protocol is provably deadlock-free.

13.4 CONCLUSION

Three important classes of deadlock in interconnection networks have been described in this chapter. Routing-induced deadlock has traditionally been the most widely studied of the three, but message-induced deadlock and reconfiguration-induced deadlock have equally devastating effects on system performance and robustness. Common to all three classes is the property of packets holding onto a set of network resources in a cyclic manner while waiting endlessly for some resource(s) within that set to become available. Unless somehow avoided or resolved once it occurs in the interconnection network subsystem, deadlock has the potential to bring an entire computer system down to a screeching halt.

Also presented in this chapter are a number of interesting techniques for handling all three classes of interconnection network deadlocks. The techniques presented are based either on deadlock avoidance or on deadlock recovery. Deadlock can be avoided in the strict sense, in the wide sense, in the weak sense, or by a combination of these approaches. What distinguishes one approach from another mainly has to do with the allowed degree of routing freedom and the possible manifestations of resource dependencies: the more routing freedom allowed, the more complex the resource dependencies are that can be manifested. The same two factors also distinguish deadlock avoidance approaches from deadlock recovery approaches. By allowing maximum routing freedom on normal network resources, knotted dependencies on fully occupied resources can form, from which there is no way of escape. The outcome of this is deadlock which must be recovered from. Deadlocks can be resolved regressively, deflectively, or progressively, depending on how deadlock resolving resources are supplied to recovering packets.

Since the same blocking property is inherent to all deadlocks, fundamental aspects of most deadlock handling approaches are universally applicable to all classes. Even though this may be the case, some approaches may be better suited to handle certain deadlock classes and less suited to handle others. This largely depends on the requirements of the applications running on the system and on system resources. In terms of design complexity, the best strategy is to have a unified solution that can synergistically handle all forms of deadlock efficiently. From a practical point-of-view, solutions that are not based on probabilistic events or have no single points-of-failure are more easily verifiable and dependable. For these and perhaps other reasons, techniques based mainly on avoiding deadlock in the strict or wide sense typically are the solutions of choice for interconnection networks implemented in commercial systems. Other proposed solutions, while intriguing, have largely been relegated to experimental systems up to this point with a few exceptions. However, as trends in design philosophy continue to advance toward that of designing more efficient systems capable of recovering from rare anomalous behavior including deadlock, approaches based on deadlock recovery may someday gain greater prominence.

13.5 Bibliographic Notes

This chapter touches only on the tip of the iceberg of the compendium of concepts and techniques proposed in the literature for handling deadlocks in interconnection networks. Much of what is discussed here can be understood in much greater detail by reading the original papers that introduce and analyze these techniques. Quantitative comparisons can then be made to better help one arrive at the best deadlock handling technique for his/her particular interconnection network subsystem. In addition to reviewing the journal references cited in this chapter and listed below, the interested reader is also encouraged to seek out other texts that treat this subject well. These would include a recent text by Duato, Yalamanchili and Ni entitled, *Interconnection Networks: An Engineering Approach*, and a new text by Dally and Towles entitled, *Principles and Practices of Interconnection Networks*, both published by Morgan-Kaufmann Publishers.

13.6 Acknowledgements

Many of the original ideas presented in this chapter have come about through the collaborative efforts of many of my former graduate student researchers and other research colleagues. I acknowledge the overall contributions made by all current and former members of the *SMART* Interconnects group at USC and, in particular, specific contributions made by the following individuals: Wai Hong Ho, Ruoming Pang, Dr. Yongho Song, Anjan Venkatramani, and Dr. Sugath Warnakulasuriya. I also owe special thanks to Professor José Duato for his participation in some of the research projects that have contributed to the contents of this chapter and for his helpful review comments.

References

1. N.J. Boden, D. Cohen, R.E. Felderman, A.E. Dulawik, C.L. Seitz, J. Seizovic, and W. Su. Myrinet-A gigabit per second local area network. In *IEEE Micro*, pages 29–36. IEEE Computer Society, February 1995.

2. R. Horst. ServerNet deadlock avoidance and fractahedral topologies. In *Proceedings of the International Parallel Processing Symposium*, pages 275–280. IEEE Computer Society, April 1996.

3. K. Malavalli, et al. Fibre Channel Switch Fabric-2 (FC-SW-2). *NCITS 321-200x T11/Project 1305-D/Rev 4.3 Specification*, pages 57–74, March 2000.

4. W. Dally, P. Carvey, and L. Dennison. The Avici Terabit Switch/Router. In *Proceedings of the Hot Interconnects VI Symposium*, pages 41–50, August 1998.

5. F. Petrini, W.C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):2–13, January-February 2002.

6. T.M. Pinkston, A.F. Benner, M. Krause, I.M. Robinson, and T. Sterling. InfiniBand: The "De Facto" Future Standard for System and Local Area Networks or Just a Scalable Replacement for PCI Buses? *Cluster Computing*, 6(2):95–104, April 2003.

7. W. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the Design Automation Conference (DAC)*, pages 684–689. ACM, June 2001.

8. M.B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, pages 341–353. IEEE Computer Society Press, February 2003.

9. W.H. Ho and T.M. Pinkston. A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, pages 377–388. IEEE Computer Society Press, February 2003.

10. P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, pages 267–286, 1979.

11. E. Baydal, P. Lopez, and J. Duato. A Simple and Efficient Mechanism to Prevent Saturation in Wormhole Networks. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pages 617–622, 2000.

12. M. Thottethodi, A.R. Lebeck, and S.S. Mukherjee. Self-Tuned Congestion Control for Multiprocessor Networks. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, January 2001.

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

14. L.-S. Peh and W. Dally. Flit-Reservation Flow Control. In *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, pages 73–84. IEEE Computer Society Press, January 2000.

15. Y. Tamir and G. Frazier. Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches. *IEEE Transactions on Computers*, 41(6):725–734, June 1992.

16. 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.

17. 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.

18. J. Duato and T.M. Pinkston. A General Theory for Deadlock-Free Adaptive Routing Using a Mixed Set of Resources. *IEEE Transactions on Parallel and Distributed Systems*, 12(12):1219–1235, December 2001.

19. W. Dally and C. Seitz. Deadlock-free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.

20. S. Warnakulasuriya and T.M. Pinkston. A Formal Model of Message Blocking and Deadlock Resolution in Interconnection Networks. *IEEE Transactions on Parallel and Distributed Systems*, 11(2):212–229, March 2000.

21. R.C. Holt. Some Deadlock Properties on Computer Systems. *ACM Computer Surveys*, 4(3):179–196, September 1972.

22. S. Warnakulasuriya and T.M. Pinkston. Characterization of Deadlocks in k-ary n-cube Networks. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):904–921, September 1999.

23. S. Warnakulasuriya and T.M. Pinkston. Characterization of Deadlocks in Irregular Networks. *Journal of Parallel and Distributed Computing*, 62(1):61–84, January 2002.

24. T.M. Pinkston, R. Pang, and J. Duato. Deadlock-Free Dynamic Reconfiguration Schemes for Increased Network Dependability. *IEEE Transactions on Parallel and Distributed Systems*, 14(8):780–794, August 2003.

25. M.D. Schroeder et al. Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links. *IEEE Journal on Selected Areas in Communication*, 9(8):1318–1335, October 1991.

26. V. Puente, R. Beivide, J.A. Gregorio, J.M. Prellezo, J. Duato, and C. Izu. Adaptive bubble router: A design to improve performance in torus networks. In *Proceedings of the 28th International Conference on Parallel Processing (28th ICPP'99)*, Aizu-Wakamatsu, Fukushima, Japan, September 1999. University of Aizu.

27. J.M. Martinez, P. Lopez, and J. Duato. FC3D: Flow Control Based Distributed Deadlock Detection Mechanism for True Fully Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(8):765–779, August 2003.

28. S. Scott and G. Thorson. Optimized Routing in the Cray T3D. In *Proceedings of the Workshop on Parallel Computer Routing and Communication*, pages 281–294, May 1994.

29. J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241–251. IEEE Computer Society, June 1997.

30. L. Ni and C. Glass. The Turn Model for Adaptive Routing. In *Proceedings of the 19th Symposium on Computer Architecture*, pages 278–287. IEEE Computer Society, May 1992.

31. B. Zerrouk, V. Reibaldi, F. Potter, A. Greiner, and A. Derieux. RCube: A Gigabit Serial Links Low Latency Adaptive Router. In *Proceedings of the Symposium on Hot Interconnects IV*, pages 13–17. IEEE Computer Society, August 1996.

32. K.D. Gunther. Prevention of Deadlocks in Packet-switched Data Transport Systems. *IEEE Transactions on Communications*, (4):512–524, April 1981.

33. D. Gunther. A DAG-Based Algorithm for Prevention of Store-and-Forward Deadlock in Packet Networks. *IEEE Transactions on Computers*, (10):709–715, October 1981.

34. D. Linder and J. Harden. An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1):2–12, January 1991.

35. A.A. Chien and J.H. Kim. Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 268–277. IEEE Computer Society, May 1992.

36. M. Galles. Spider: A High Speed Network Interconnect. In *Proceedings of the Symposium on Hot Interconnects IV*, pages 141–146. IEEE Computer Society, August 1996.

37. J. Carbonaro. Cavallino: The Teraflops Router and NIC. In *Proceedings of the Symposium on Hot Interconnects IV*, pages 157–160. IEEE Computer Society, August 1996.

38. W. Dally, L. Dennison, D. Harris, K. Kan, and T. Zanthopoulos. Architecture and Implementation of the Reliable Router. In *Proceedings of the Hot Interconnects II Symposium*, August 1994.

39. S.L. Scott and G.M. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Proceedings of the Symposium on Hot Interconnects IV*, pages 147–156. IEEE Computer Society, August 1996.

40. S.S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. In *Symposium on High Performance Interconnects (HOT Interconnects 9)*, pages 113–117. IEEE Computer Society Press, August 2001.

41. W. Barrett et al. An Overview of the Blue-Gene/L Supercomputer. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, CD ROM*, November 2002.

42. L. Gravano, G. Pifarre, P. Berman, and J. Sanz. Adaptive Deadlock- and Livelock-Free Routing With all Minimal Paths in Torus Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1233–1251, December 1994.

43. 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.

44. W. 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.

45. J. Flich, P. Lopez, M.P. Malumbres, and J. Duato. Boosting the Performance of Myrinet Networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(7):693–709, July 2002.

46. A.G. Greenberg and B. Hajek. Deflection Routing in Hypercube Networks. *IEEE Transactions on Communications*, COM-40(6):1070–1081, June 1992.

47. S. Konstantinidou and L. Snyder. Chaos Router: Architecture and Performance. In *Proceedings of the 18th International Symposium on Computer Architecture*, pages 212–221. IEEE Computer Society, May 1991.

48. N. McKenzie, K. Bolding, C. Ebeling, and L. Snyder. ChaosLAN: Design and Implementation of a Gigabit LAN Using Chaotic Routing. In *Proceedings of the 2nd PCRCW*, pages 211–223. Springer-Verlag, June 1997.

49. J. Kim, Z. Liu, and A. Chien. Compressionless Routing: A Framework for Adaptive and Fault-tolerant Routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):229–244, March 1997.

50. M. Coli and P. Palazzari. An Adaptive Deadlock and Livelock Free Routing Algorithm. In *3rd Euromicro Workshop on Parallel and Distributed Processing*, pages 288–295. San Remo, Italy, January 1995.

51. J.M. Martinez, P. Lopez, J. Duato, and T.M. Pinkston. Software-based Deadlock Recovery for True Fully Adaptive Routing in Wormhole Networks. In *Proceeding of the 1997 International Conference on Parallel Processing*, pages 182–189. IEEE Computer Society, August 1997.

52. K.V. Anjan and T.M. Pinkston. An Efficient, Fully Adaptive Deadlock Recovery Scheme: *DISHA*. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 201–210. IEEE Computer Society, June 1995.

53. T.M. Pinkston, Y. Choi, and M. Raksapatcharawong. Architecture and Optoelectronic Implementation of the WARRP Router. In *Proceedings of the 5th Symposium on Hot Interconnects*, pages 181–189. IEEE Computer Society, August 1997.

54. K.V. Anjan, T.M. Pinkston, and J. Duato. Generalized Theory for Deadlock-Free Adaptive Wormhole Routing and its Application to Disha Concurrent. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 815–821. IEEE Computer Society, April 1996.

55. K.V. Anjan and T.M. Pinkston. *DISHA*: A Deadlock Recovery Scheme for Fully Adaptive Routing. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 537–543. IEEE Computer Society, April 1995.

56. T.M. Pinkston. Flexible and Efficient Routing Based on Progressive Deadlock Recovery. *IEEE Transactions on Computers*, 48(7):649–669, July 1999.

57. Y.H. Song and T.M. Pinkston. A New Mechanism for Congestion and Deadlock Resolution. In *The 2002 International Conference on Parallel Processing*, pages 81–90. IEEE Computer Society, August 2002.

58. C.B. Stunkel et al. The SP2 high-performance switch. *IBM Systems Journal*, 34(2):185–204, 1995.

59. A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B-H. Lim, K. Mackenzie, and D. Yeung. The MIT alewife machine: Architecture and performance. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture (ISCA'95)*, pages 2–13, June 1995.

60. J.F. Martinez, J. Torrellas, and J. Duato. Improving the Performance of Bristled CC-NUMA Systems Using Virtual Channels and Adaptivity. In *Proceedings of 13th International Conference on Supercomputing*, June 1999.

61. Y.H. Song and T.M. Pinkston. A Progressive Approach to Handling Message-Dependent Deadlock in Parallel Computer Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):259–275, March 2003.

62. D. Teodosiu, J. Baxter, K. Govil, J. Chapin, M. Rosenblum, and M. Horowitz. Hardware Fault Containment in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 73–84. IEEE Computer Society Press, June 1997.

63. *InfiniBand^{TM} Architecture Specification Volume 1.* InfiniBand Trade Association, October 24, 2000.

64. O. Lysne and J. Duato. Fast Dynamic Reconfiguration in Irregular Networks. In *The 2000 International Conference on Parallel Processing*, pages 449–458. IEEE Computer Society, August 2000.

65. F.J. Quiles, J.L. Sanchez, R. Casado, A. Bermudez and J. Duato. A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks. *Special Issue on Dependable Network Computing. IEEE Transactions on Parallel and Distributed Systems*, 12(2):115–132, February 2001.

66. R. Pang, T.M. Pinkston and J. Duato. Dynamic Reconfiguration of Networks with Distributed Routing: The Single Scheme. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 2042–2048, June 2001.