Software-Based Deadlock Recovery Technique for True Fully Adaptive Routing in Wormhole Networks [*]

J. M. Martínez, P. López, J. Duato

Facultad de Informática
Universidad Politécnica de Valencia
Camino de Vera s/n
46071 - Valencia, SPAIN
E-mail: {jmmr,plopez,jduato}@gap.upv.es

T. M. Pinkston

SMART Interconnects Group
EE-Systems Dept.
University of Southern California
Los Angeles, CA 90089-2562
E-mail: tpink@charity.usc.edu

Abstract

Networks using wormhole switching have traditionally relied upon deadlock avoidance strategies for the design of deadlock-free routing algorithms. More recently, deadlock recovery strategies have begun to gain acceptance. In particular, progressive deadlock recovery techniques are very attractive because they allocate a few dedicated resources to quickly deliver deadlocked packets, instead of killing them. Deadlock recovery is based on the assumption that deadlocks are rare. Very recently, the frequency of deadlock occurrence was measured [21, 18], showing that deadlocks are highly unlikely when enough routing freedom is provided. However, deadlocks are more prone when the network is close to or beyond saturation. Additionally, some performance degradation has been observed at saturation. Similar performance degradation behavior at saturation was also observed in networks using deadlock avoidance strategies [9].

In this paper we take a different approach to handle deadlocks and performance degradation. We propose the use of an injection limitation mechanism that prevents performance degradation near the saturation point and reduces the probability of deadlock to negligible values even when fully adaptive routing is used. We also propose an improved deadlock detection mechanism that only uses local information, detects all the deadlocks, and considerably reduces the probability of false deadlock detection over previous proposals. In the rare case when impending deadlock is detected, our proposed recovery technique absorbs the deadlocked message at the current node and later re-injects it for continued routing towards its destination. Performance evaluation results show that our new approach to deadlock handling is more efficient than previously proposed techniques.

Keywords Wormhole switching, adaptive routing, deadlock recovery, deadlock detection, virtual channels.

[*]This work was supported by the Spanish CICYT under Grant TIC94–0510–C02–01 and an NSF Career Award, grant ECS-9624251.

1 Introduction

Wormhole switching [8] has become the most widely used switching technique for multicomputers and distributed shared-memory multiprocessors, and it is also being used for networks of workstations [5]. The use of virtual channels can increase network throughput considerably by dynamically sharing the physical bandwidth among several messages [7]. However, it has been shown that virtual channels are expensive, increasing node delay considerably [6]. Therefore, the number of virtual channels per physical channel should be kept small.

An alternative approach to increase throughput consists of using adaptive routing [13]. However, deadlocks must be handled efficiently. A deadlock occurs in an interconnection network when no message is able to advance toward its destination because the network buffers are full. A simple and effective approach to handle deadlocks consists of restricting routing so that there are no cyclic dependencies between channels [8]. A more efficient approach consists of allowing the existence of cyclic dependencies between channels while providing some escape paths to avoid deadlock, therefore increasing routing flexibility [9, 11]. However, such deadlock avoidance techniques require dedicated resources to provide those escape paths. Usually, those dedicated resources are virtual channels, thus preventing the use of all the virtual channels for fully adaptive routing. Deadlock recovery strategies overcome this constraint, but the cost associated with existing deadlock recovery strategies can be higher than necessary, especially with regressive techniques which kill and later re-inject deadlocked messages at the original source node [19, 14]. Progressive deadlock recovery strategies, like *Disha* [2, 3], are more efficient as only a few dedicated resources are allocated to quickly deliver deadlocked packets, instead of killing them. One central buffer per node is enough to route deadlocked messages to their destination by preempting network bandwidth from non-deadlocked packets only when impending deadlock is detected.

Progressive deadlock recovery techniques usually achieve a higher performance than deadlock avoidance techniques because they require less dedicated resources to handle deadlocks [3]. However, both techniques may produce severe performance degradation when the network is close to saturation. Performance degradation at the saturation point was studied in [9, 15] and, more recently, in [21, 18]. In [9], this situation was described as occurring when messages block cyclically faster than they are drained using the escape path. Although this can be mitigated by adding several virtual channels per physical channel, this solution is expensive and is overkill for most networks. In [21, 18], the frequency of deadlock occurrence on k-ary n-cubes using a true fully adaptive minimal routing algorithm with deadlock recovery was measured. It was shown that deadlocks rarely occur when sufficient routing freedom is provided, but they are more likely to occur when the network is close to or beyond saturation. Although this suggests that deadlock recovery techniques are viable, they suffer similar performance degradation at network saturation due to the phenomenon described in [9]. In fact, performance degradation at saturation can be more pronounced in Disha-based recovery techniques than in deadlock avoidance-based techniques since less resources (and, therefore, less bandwidth) are provided to drain cyclically blocked messages [4]. Thus, regardless of the technique used to handle deadlocks, performance degradation should be addressed. Draining cyclically blocked messages may require more bandwidth than is provided by existing deadlock recovery and deadlock avoidance techniques.

In this paper, a different approach is taken to handle both deadlocks and performance degradation. We propose the use of the injection limitation mechanism proposed in [15] to prevent performance degradation near the saturation point. It consists of limiting message injection when the network is heavily loaded. As a by-product, the probability of deadlock is reduced to negligible levels even when fully adaptive routing is used with only a few virtual channels. We also propose an improved deadlock detection mechanism that uses only local information to more accurately detect deadlocks and considerably reduce false deadlock detection over previous proposals [14, 2]. In the rare cases when deadlocks are suspected, we propose a new software-based progressive recovery technique that absorbs (as opposed to killing) the deadlocked message at the current node and later re-injects it from the current node for continued routing towards its destination. This technique has some points in common with the software-based fault-tolerant routing mechanism proposed in [20]. Indeed, both techniques can be combined for increased performance and reliability. Thus, our new progressive deadlock recovery technique incorporates simple mechanisms that minimize performance degradation at saturation as well as the occurrence of deadlocks, improves deadlock detection and simplifies the recovery procedure. The main contributions of this paper are a software-based progressive deadlock recovery technique that requires no buffers to handle deadlocks (although it requires some buffer space in the local node), an improved deadlock detection mechanism, and a detailed study of the behavior of those mechanisms.

Section 2 gives background on deadlock avoidance and recovery techniques, highlighting the motivation for this work. Section 3 describes the message injection limitation mechanism used to reduce deadlock probability and performance degradation. Section 4 presents our improved deadlock detection mechanism that more accurately differentiates between false deadlock (congestion) and true deadlocks. Section 5 presents our simpler yet efficient software-based deadlock recovery strategy that benefits from the other mechanisms proposed in this paper. Section 6 gives the performance results of true fully adaptive routing with our proposed deadlock recovery mechanisms compared against previously proposed adaptive routing algorithms using other deadlock avoidance techniques. Finally, some conclusions are drawn in Section 7.

2 Background

As presented in [4], the theory of deadlock avoidance proposed in [9] can be easily extended to support progressive deadlock recovery. Indeed, both deadlock handling techniques are very similar from a theoretical point of view. Both of them allow fully adaptive routing on some set of resources while providing dedicated resources to escape from deadlock. The theories proposed in [9, 4] provide a static view of the network, allowing one to formally prove that escape resources are enough to avoid or recover from any deadlocked configuration: if several messages block cyclically waiting for resources held by other messages, these theories guarantee that some resources will become available sooner or later and that all the messages will be able to proceed. However, from a more practical point of view, guaranteeing that escape resources will become available sooner or later may not yield the highest performance.

Consider a network using unrestricted fully adaptive wormhole routing over virtual channel resources. The routing flexibility provided by this algorithm produces cyclic dependencies between channels in most topologies. When the network is heavily loaded (close to or beyond the saturation point), messages block cyclically very quickly. If escape resources are not able to drain messages from those cycles fast enough, messages will have to wait for a long time. As a consequence, those blocked messages will occupy channel bandwidth and, thus, decrease network throughput considerably. Additionally, the latency of those messages will also increase considerably. This behavior was first described in [9]. The important point here is that at least one of the following must occur to mitigate this behavior: either escape resources must provide enough bandwidth to drain messages blocking cyclically (regardless of whether deadlock avoidance or recovery is used) or a mechanism(s) must prevent the build-up of cyclically blocked messages that would subsequently need to be drained.

Deadlock avoidance and progressive deadlock recovery techniques mainly differ in the way they supply escape paths and in when those paths are used. Consider first how escape paths are supplied. Deadlock avoidance techniques have traditionally relied upon virtual channels to supply escape paths [9, 11]. Progressive deadlock recovery techniques like Disha [2, 3, 4] use a flit-sized central buffer to supply the escape paths and route over these buffers by preempting network bandwidth from nondeadlocked packets so as to quickly resolve impending deadlock. In both cases, escape paths are implemented as additional dedicated resources in the router (although Disha-based recovery requires less router resources and, therefore, achieves higher performance before network saturation). Consider next the issue of when escape paths are used. Deadlock avoidance techniques typically allow the immediate use of escape resources when a message is blocked (although it is possible to limit their use by using time-outs [10]). Deadlock recovery techniques, however, generally limit the use of escape resources allowing only those messages suspected of being involved in deadlock to use them; otherwise, the limited bandwidth offered by recovery resources would quickly saturate. Existing deadlock detection mechanisms use only crude time-out information on blocked messages and do not use other relevant information such as physical channel activity. This makes the mechanism susceptible to mistaking congestion for deadlocks, particularly when messages are blocked for long periods of time waiting for resources occupied by long messages which are not blocked.

We believe a simple injection limitation mechanism can keep the network below its saturation point to prevent the build-up of cyclically blocked messages that could lead to deadlock formation and/or performance degradation. We also believe that deadlock detection can be made more accurate to keep recovery resources from becoming saturated with nondeadlocked messages by associating the time-out mechanism with physical channel inactivity instead of just message blocking. These fine-tuning mechanisms minimize the probability of packets recovering from suspected deadlocks to such infinitesimal levels that allow the router to be simplified by not requiring any edge or central buffers to supply escape/recovery paths. We believe that the buffer space already provided at each node can be utilized as a low-cost solution to this even more highly improbable case. Simulation results confirm our belief that highest possible performance without degradation can be achieved.

3 Message Injection Limitation

In this section, we briefly describe the message injection limitation proposed in [15]. As we mentioned above, there is some performance degradation when the network reaches saturation. The problem can be stated as follows: Latency increases with network traffic until a certain point (saturation point) is reached, at which time the latency value increases considerably while throughput (accepted traffic) tails off. In other words, accepted traffic noticeably decreases when the saturation point is reached.

Performance degradation within the network occurs when routing algorithms allow cyclic dependencies between channels. When traffic becomes high, messages block cyclically faster than they are drained by the escape paths, thus increasing latency and decreasing throughput. Provided that there are some escape resources to drain messages blocking cyclically, deadlock cannot occur, but messages wait for a long time in the network. One solution to this problem is to increase routing freedom by adding more virtual channels [9, 21, 18]; then messages will have less probability of being involved in cyclic dependencies. However, an excessive number of virtual channels could lead to a reduced clock frequency [6].

Another solution is to control network traffic to ensure that it is always under the performance degradation point. But traffic is often global in nature. Thus, it is not feasible to easily measure it at each node. As an approximation, traffic can be estimated locally by counting the number of busy virtual output channels at each node [15]. We have found that the average number of busy virtual output channels at each router monotonically increases with network load. Hence, we can establish that there is a useful correlation between the number of busy virtual channels and the network tending to saturation. This allows us to approximate global traffic rate by simply monitoring the number of busy virtual output channels local to a router. We use this in implementing our injection limitation mechanism to avert network saturation.

When the number of busy virtual output channels surpasses a threshold value, the router prevents the injection of new messages, keeping them at the source node. If we properly select the threshold value, there is a high probability that the network will never reach saturation and performance degradation can be mitigated. A simple implementation of this mechanism requires only a register which holds the threshold value, a comparator, and a counter associated with each router, which are not in the critical path. The counter is incremented each time a successful route is established (another output virtual channel becomes occupied) and is decremented when the tail of a message leaves the router. Of course, this mechanism will increase the delay of those messages that are not injected into the network at once, but the average message delay can actually be less than that obtained by the same adaptive algorithm without the injection limitation mechanism because of the degradation mentioned above. Results show that performance degradation as measured by the tailing-off of throughput is eliminated completely [15]. Moreover, the increment in message latency produced by the injection limitation is negligible for the whole range of network traffic. In [16] we present an in-depth discussion of this and other message injection limitation mechanisms.

4 Improved Deadlock Detection Mechanism

Previously proposed deadlock detection mechanisms are based on measuring the inactivity time of blocked messages [2, 14]. In this section, we describe a more accurate deadlock detection mechanism that better distinguishes between messages blocked due to network congestion and messages blocked due to likely impending deadlock.

A deadlock detection mechanism should have two important features. First, it should be simple; it should not add needless complexity to the network that could reduce performance and/or increase cost. Second, it should be implemented as a distributed mechanism, working only with local information available at each router.

Instead of measuring the time a message is blocked, the proposed mechanism measures the time that channels requested by messages are inactive due to the current messages occupying them remaining blocked. Transmission activity is monitored in all the virtual output channels that can be used by a given blocked message. A message is only presumed to be deadlocked if all the alternative virtual output channels that are requested by that message contain blocked messages. It should be noted that when the routing algorithm uses all the virtual channels in each physical channel in the same way, it is only necessary to monitor activity in the physical channels. This is the case for true fully adaptive routing.

This mechanism can be implemented as follows. A counter is associated with each output physical channel. This counter is incremented every clock cycle and is reset when a flit is transmitted across the physical channel. Thus, the counter contains the number of cycles that this channel is inactive. Note that this counter also indicates the number of cycles since the last flit transmission across any of the virtual channels in that physical channel. This time is continuously compared with a given threshold. If it is greater than this threshold, a one-bit flag (inactivity flag) is set indicating that the physical output channel is inactive. The flag is reset when a flit is transmitted across the physical channel.

The routing control unit is assigned to message headers in a round-robin fashion. Blocked headers are also routed in order to determine whether some of the output channels requested by them became free. Every time a message is routed, if all the feasible virtual output channels are busy, then the inactivity flags associated with the corresponding physical output channels are checked. If all of these flags are set, then there is no activity through any of the feasible physical output channels, and the message is presumed to be involved in a deadlock.

It is important to note that the counters and inactivity flags are associated with physical output channels, instead of virtual channels. This is only correct if the routing algorithm can use all the virtual channels of a given physical channel in the same way as with true fully adaptive routing. This considerably simplifies the implementation.

In order to implement the mechanism, the only required hardware is a counter, a comparator and a single bit latch associated with each physical output channel. If we want a programmable threshold, then another register is needed. However, in order to simplify the comparison between the counter and threshold value, it is recommendable to select a power of two for the threshold value. In this case, a single output bit of the counters is enough to indicate that the threshold has been reached. No comparators and registers are needed. In addition, the router must be modified to check the inactivity flags every time an unsuccessful routing is made.

Finally, it should be noted that the mechanism will detect all possible deadlocks, but also some false deadlocks depending on the threshold used. Thus, the mechanism must be properly tuned, choosing the appropriate threshold.

5 Software-Based Deadlock Recovery

Wormhole networks have traditionally relied upon deadlock avoidance for the design of deadlock-free routing strategies [17]. Thus, routing algorithms usually have some constraints in order to avoid deadlocks. Recently, the frequency of deadlock occurrence in *k*-ary *n*-cube networks using wormhole switching was measured emperically [21, 18]. From this study, we know that deadlocks are very rare, especially when two or more virtual channels are used with true fully adaptive routing. Moreover, the message injection limitation mechanism described in Section 3 can be used to further reduce the probability of reaching deadlocked configurations.

Let us assume that true fully adaptive minimal routing is used. This routing algorithm imposes no restrictions on the use of virtual and physical channels, except that paths should be minimal. Also, let us assume that message injection is limited by using the mechanism proposed in Section 3. The mechanism described in Section 4 is used for deadlock detection. Although deadlock detection is highly improbable, it could still be detected. So, a recovery mechanism is required.

It is easy to see that in a deadlocked configuration, at least one of the messages involved in it will have its header at the head of an input buffer, waiting for an output channel. Also, the proposed deadlock detection mechanism only presumes that a message is deadlocked if its header is being routed (it is at the head of an input buffer). Thus, all we have to do in order to recover from deadlock is to remove that message from the network by ejecting it at the current node. This can be easily accomplished by the router when it detects a possibly deadlocked message. The router selects the internal memory channel at the current node for this message, as if this node were its destination. A control bit is required to distinguish between normal and deadlocked messages.

Finally, removed messages must be re-injected into the network at a later time. This is also easy to accomplish.

If the software messaging layer detects reception of a message whose destination is not the current node, it must inject the message again into the network. The true message destination can be found in the message header.

The proposed mechanism is a low-cost progressive deadlock recovery technique. Instead of killing deadlocked messages [14, 19], it absorbs them at the current node, allowing them to make progress at a later time. The main advantage of this technique is its simplicity. The proposed recovery mechanism does not even require dedicated buffers in the router to recover from deadlock. It is enough to have some buffer space in the local node. Moreover, programmable network interfaces based on powerful processsors and large buffer memory are emerging as a viable host for communication operations [5], being the future host for message handlers without involving the processor. These network interfaces meet the buffer requirements of software-based deadlock recovery. By keeping the deadlock recovery operations in the interface, we gain some efficiencies, since messages do not have to traverse the I/O bus and the memory hierarchy of the local node. Therefore, this mechanism provides improvement over Disha by requiring a simpler router design. However, the software-based recovery mechanism assumes that a processor is associated with every node, and this is not true for all networks. In those cases, Disha should be preferred. On the other hand, the described mechanism is a software solution which is always slower than a hardware one. However, taking into account that deadlocks are not frequent, the proposed mechanism will not be used frequently. It may even happen that deadlocks are never detected, provided that the other mechanisms proposed in this paper are properly tuned. Additionally, routing can be done without any restrictions, increasing overall performance. Thus, the mechanism proposed in this paper makes the common case fast.

It must be noted that the proposed recovery strategy does not increase performance over Disha [2, 3], but it eliminates the requirement of using buffers (edge or central) for deadlock recovery. In addition, this mechanism provides more routing flexibility because it allows unrestricted use of non-minimal paths. However, Disha only allows non-minimal paths to normal messages; routing of deadlocked messages requires minimal paths. This property is interesting for the implementation of fault-tolerant routing. Moreover, the software-based deadlock recovery mechanism proposed in this paper can be easily combined with the software-based fault-tolerant routing strategy proposed in [20]. As this deadlock recovery technique imposes no routing restrictions, non-minimal routing could be used in the presence of faults, thus improving fault tolerance and performance with respect to [20]. Fault tolerant routing is beyond the scope of this paper and is the subject of future research.

6 Performance Evaluation

In this section, we evaluate by simulation the behavior of true fully adaptive routing algorithms using the deadlock recovery strategy proposed in this paper.

The evaluation methodology used is based on the one proposed in [9]. The most important performance measures are latency (time required to transmit a message) and throughput (maximum traffic accepted by the network). Traffic is the flit reception rate. Latency is measured in cycles. Traffic is measured in flits per node per cycle.

Taking into account the sizes of current multicomputers and the studies about the optimal number of dimensions [1], we have evaluated the performance of the new algorithms on a 8-ary 3-cube network (512 nodes).

6.1 Network Model

Our simulator models the network at the flit level. Each node of the network consists of a processor, its local memory and a router. The router contains a routing control unit, a switch, and several physical channels. The processor is connected to its router by four independent channels.

The routing control unit computes the output channel for a message as a function of its destination node, the current node and the output channel status. The routing algorithm can use any minimal path to forward a message toward its destination. In addition, several virtual channels per physical channel can be used. In other words, all virtual channels in all the feasible directions can be used. This algorithm is referred to as True Fully Adaptive Routing algorithm (TFAR). The routing control unit can process only one message header at a time. It is assigned to waiting messages in a demand-slotted round-robin fashion (including those messages generated in the local processor). When a message gets the routing control unit but it cannot be routed because all the alternative output channels are busy, it must wait in the input buffer until its next turn. The deadlock recovery strategy proposed in Section 5 is used, together with the deadlock detection mechanism described in Section 4.

The internal router switch is a crossbar. Thus, it allows multiple messages to traverse it simultaneously without interference. It is configured by the routing control unit each time a successful routing is made.

Physical channels can be split into several virtual channels. Virtual channels are assigned to the physical link using a demand-slotted round-robin arbitration scheme. Each virtual channel has an associated buffer with capacity for four flits.

6.2 Message Generation

Message traffic and message length depend on the applications. For each simulation run, message generation rate is assumed to be constant and the same for all the nodes. Once the network has reached a steady state, the flit generation rate is equal to the flit reception rate (traffic). We evaluate the full range of traffic, from low load to saturation. Message destination is randomly chosen among all the nodes. Short messages (16 flits), long messages (64 flits) and bimodal message lengths (60% of 16-flit messages and 40% of 64-flit messages) are considered.

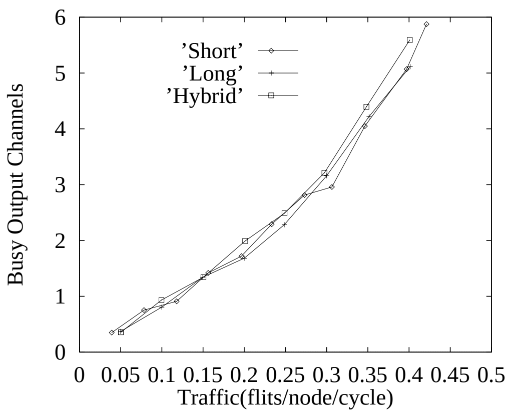

Figure 1: Busy output channels versus traffic for a 512-node 3-D torus with a true fully adaptive routing algorithm with 2 virtual channels per physical channel.

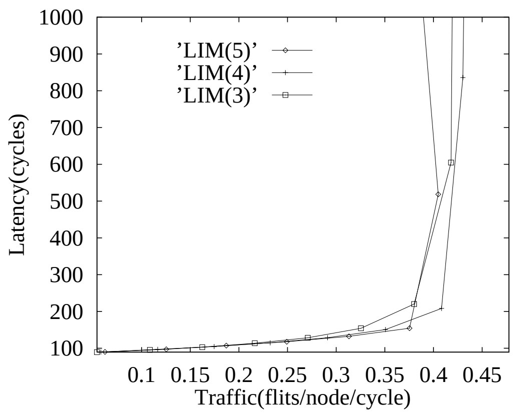

Figure 2: Average message latency versus traffic for a 512-node 3-D torus with a true fully adaptive routing algorithm with 2 virtual channels per physical channel.

6.3 Adjusting Message Injection Limitation

In this section, we select the appropriate threshold for the message injection limitation mechanism described in Section 3.

As an example, Figure 1 shows the average number of busy virtual output channels versus traffic for the TFAR routing algorithm with 2 virtual channels per physical channel. The number of busy virtual output channels when the network is close to saturation is almost the same for all the message lengths analyzed. In particular, there are almost 6 busy channels on average. The plot for the TFAR algorithm with 3 virtual channels per physical channel (not shown) has a similar shape. In this case, there are 9 busy virtual output channels when the network is saturated. The optimal value for the injection limitation threshold should be close to these values. In Figure 2 we can see the performance of the TFAR algorithm with 2 virtual channels per physical channelwith several injection limitation thresholds. Taking into account that the results do not depend on message length, for the sake of shortness, the results are only shown for one message length. In order to remove the performance degradation of the routing algorithm, message injection must be avoided if the number of busy output channels exceeds 4 virtual channels. For the TFAR routing algorithm with 3 virtual channels per physical channel, message injection must be avoided if the number of busy output channels exceeds 8 virtual channels. These values of busy virtual output channels were used as thresholds for the message injection limitation mechanism in the simulation results presented in Sections 6.4 and 6.5.

6.4 Frequency of Deadlock Detection

In this section we tune the deadlock detection mechanism proposed in Section 4. Tables 1 and 2 show the number of messages detected as possibly deadlocked for different values of the threshold for the TFAR routing algorithm with 2 virtual channels per physical channel, measured in clock cycles. A message is included in the count if it has been detected as possibly deadlocked using the indicated threshold in any of the checkpoints. The tables show the values for the new detection mechanism proposed in this paper (NDM) and previously proposed mechanisms based on time-outs (Tout). However, no message deadlocked during the simulations, since all messages arrived at their destinations. The statistics have been gathered periodically during the simulations. The simulations have been run for a number of cycles high enough to deliver 100,000 messages. As we can see in Table 1, the routing algorithm with 2 virtual channels per physical channel and short messages requires a threshold not lower than 64 cycles in order to avoid false deadlock detections using the NDM. From Table 2, if messages are long, the threshold must be 4 times higher ($\geq$ 256 cycles), matching the relationship between lengths for long and short messages. Similar thresholds are required when 3 virtual channels per physical channel are used in the true fully adaptive routing algorithm.

Threshold	64 cycles		32 cycles		16 cycles	
Traffic	NDM	Tout	NDM	Tout	NDM	Tout
0.30	0	0	0	2	0	11
0.35	0	1	0	4	0	15
0.40	0	2	2	11	2	30
0.44	0	13	4	43	11	107

Table 1: Number of messages detected as possibly deadlocked for the new detection mechanism (NDM) and previously proposed mechanism based on time-outs (Tout). True fully adaptive routing algorithm with 2 virtual channels per physical channel is used with 16-flit messages.

Therefore, the optimal value for the threshold depends on message length. The only simple solution consists of

Threshold	256 cycles		128 cycles		64 cycles	
Traffic	NDM	Tout	NDM	Tout	NDM	Tout
0.23	0	0	0	1	0	4
0.29	0	0	0	1	0	6
0.35	0	0	0	2	0	10
0.41	0	4	0	10	2	32
0.43	0	13	3	40	9	96

Table 2: Number of messages detected as possibly deadlocked for the new detection mechanism (NDM) and previously proposed mechanism based on time-outs (Tout). True fully adaptive routing algorithm with 2 virtual channels per physical channel is used with 64-flit messages.

splitting messages into fixed length packets. However, the proposed mechanism considerably reduces the number of false deadlock detections over crude time-outs. Moreover, no deadlocks are detected even when the network reaches the saturation point, provided that the threshold is properly tuned. This is not the case for detection mechanisms based on crude time-outs. These mechanisms detect a much higher number of false deadlocks for the same time-out value. Additionally, the number of detected deadlocks increases very quickly when the network approaches saturation. Hence, the proposed deadlock detection mechanism considerably improves over previously proposed ones.

We are currently evaluating the influence of message destination distribution on the deadlock detection mechanism. Also, we are trying to improve this mechanism to make it less dependent on message length.

6.5 Performance Comparison

In this section we compare the performance of true fully adaptive routing and the proposed progressive deadlock recovery mechanism with previously proposed fully adaptive routing algorithms using deadlock avoidance [12]. Note that preemptive progressive deadlock recovery techniques (like Disha [3]) achieve the same performance as the proposed recovery mechanism, assuming that both of them use the same injection limitation and deadlock detection mechanisms and that the additional router complexity required in Disha does not impact clock frequency.

In particular, the TFAR routing algorithm for a *k*-ary *n*-cube with 2 and 3 virtual channels per physical channel with the message injection limitation mechanism described in Section 3 and the deadlock recovery mechanism proposed in Section 5 is compared against the deterministic routing algorithm (Det) proposed in [8] and the fully adaptive routing algorithm (FAR) proposed in [12]. The deterministic algorithm uses two virtual channels per physical channel, while the fully adaptive one uses three virtual channels per physical channel. For message ejection and re-injection at intermediate nodes, we assumed a delay of

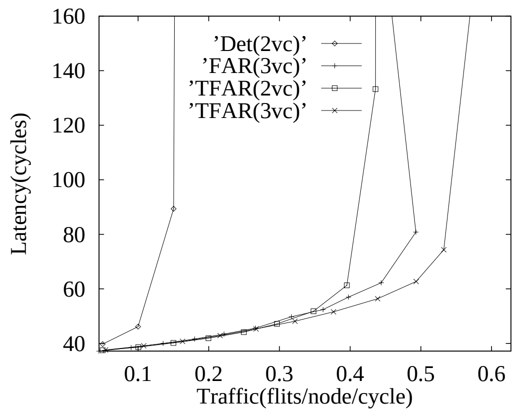

Figure 3: Average message latency versus traffic for a 512-node 3-D torus using different routing algorithms (Deterministic, Fully Adaptive Routing (FAR), and True Fully Adaptive Routing (TFAR). Short messages are assumed.

200 cycles. The threshold for deadlock detection was four times the size of the longest message.

Taking into account that the results for the three message lengths considered have the same shape, for the sake of shortness, we will only show the results for short messages in Figure 3. The TFAR routing algorithm achieves a throughput three times higher than the deterministic algorithm, with the same number of virtual channels. The fully adaptive routing algorithm with 3 virtual channels achieves only a slightly higher throughput than the true fully adaptive routing algorithm with only two virtual channels. The best results are achieved by the true fully adaptive routing algorithm with 3 virtual channels per physical channel, which achieves about a 15% more throughput than the fully adaptive one and lower latency than any other routing algorithm for the full range of traffic. No deadlocks were detected during the simulations. In addition, the true fully adaptive routing algorithm has no performance degradation at saturation. These results show the effectiveness of the proposed mechanisms for message injection limitation and deadlock detection, therefore enabling the use of simple software-based deadlock recovery mechanisms.

7 Conclusions

In this paper, we proposed a set of mechanisms that minimize the hardware requirements to handle deadlocks. In particular, we proposed an injection limitation mechanism that reduces the probability of deadlock to negligible values, and eliminates performance degradation at the saturation point. This mechanism only requires measuring the number of busy output channels at each node. We also proposed an improved deadlock detection mechanism that considerably reduces the probability of false deadlock detection. It is based on monitoring flit advancement across the channels requested by blocked messages. Both mechanisms are tunable through simulation. The combination of these two mechanisms is so effective that no deadlocks were detected during the simulations, even when using a true fully adaptive routing algorithm. Hence, these mechanisms enable the use of simple and inexpensive techniques for deadlock recovery. Thus, we proposed a software-based progressive deadlock recovery mechanism that requires no buffers to recover from deadlocks. It is based on the absorption of messages detected as being deadlocked at the current node and their re-injection into the network. Although the absorption of messages incurs a high latency, false deadlock detection has been reduced to negligible values, making software-based deadlock recovery feasible. In addition, this deadlock recovery mechanism can be easily combined with the software-based fault-tolerant routing strategy proposed in [20].

The proposed mechanisms were combined with a true fully adaptive routing algorithm for the 3D-torus that can route messages following any minimal path. Any number of virtual channels per physical channel can be used. This routing algorithm was evaluated and compared with other well-known routing algorithms (deterministic [8] and fully adaptive with a deadlock avoidance mechanism [12]). The results show that the true fully adaptive routing algorithm achieves a reduction in message latency for the full range of traffic while increasing throughput for both short (16-flit) and long (64-flit) messages.

In conclusion, we proposed a set of mechanisms to handle deadlocks that require a very small amount of hardware, eliminate performance degradation at saturation point, reduce the frequency of deadlock to negligible values, and considerably reduce the probability of false deadlock detection. To the best of our knowledge, this is the first feasible deadlock handling technique for wormhole networks that requires no dedicated buffer resources to handle deadlocks.

References

[1] A. Agarwal, "Limits on interconnection network performance", *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, Oct. 1991.

[2] Anjan K. V. and T. M. Pinkston, "DISHA: A deadlock recovery scheme for fully adaptive routing," in *Proc. of the 9th Int. Parallel Processing Symposium*, April 1995.

[3] Anjan K. V. and T. M. Pinkston, An efficient fully adaptive deadlock recovery scheme: DISHA," in *Proc. of the 22nd Int. Symposium on Computer Architecture*, June 1995.

[4] Anjan K. V., T. M. Pinkston and J. Duato, "Generalized theory for deadlock-free adaptive routing and its application to Disha Concurrent," in *Proc. of the 10th Int. Parallel Processing Symposium*, April 1996.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic and W. Su, "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29–36, February 1995.

[6] A. A. Chien, "A cost and speed model for k-ary n-cube wormhole routers," in *Proc. of Hot Interconnects'93*, August 1993.

[7] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.

[8] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Computers*, vol. C–36, no. 5, pp. 547–553, May 1987.

[9] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, December 1993.

[10] J. Duato, "Improving the efficiency of virtual channels with time-dependent selection functions," *Future Generation Computer Systems*, no. 10, pp. 45–58, 1994.

[11] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055–1067, October 1995.

[12] J. Duato and P. López, "Performance evaluation of adaptive routing algorithms for k-ary n-cubes," in *Proc. of the Workshop on Parallel Computer Routing and Communication*, May 1994.

[13] P. T. Gaughan and S. Yalamanchili, "Adaptive routing protocols for hypercube interconnection networks," *IEEE Computer*, vol. 26, no. 5, pp. 12–23, May 1993.

[14] J. H. Kim, Z. Liu and A. A. Chien, "Compressionless routing: A framework for adaptive and fault-tolerant routing," in *Proc. of the 21st Int. Symposium on Computer Architecture*, April 1994.

[15] P. López and J. Duato, "Deadlock-free adaptive routing algorithms for the 3-D torus: Limitations and solutions," in *Proc. of Parallel Architectures and Languages Europe 93*, June 1993.

[16] P. López, J.M. Martínez, J. Duato and F. Petrini, "On the reduction of deadlock frequency by limiting message injection in wormhole networks," in *Proc. of the Workshop on Parallel Computer Routing and Communication*, June 1997.

[17] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, no. 2, pp. 62–76, February 1993.

[18] T.M. Pinkston and S. Warnakulasuriya, "On deadlocks in interconnection networks", in *Proc of the 24th Int. Symposium on Computer Architecture*, June 1997.

[19] D. S. Reeves, E. F. Gehringer and A. Chandiramani, "Adaptive routing and deadlock recovery: A simulation study," in *Proc. of the 4th Conference on Hypercube, Concurrent Computers & Applications*, March 1989.

[20] Y. Suh, B.V. Dao, J. Duato and S. Yalamanchili, "Software based fault-tolerant oblivious routing in pipelined networks," in *Proc. of the 1995 Int. Conference on Parallel Processing*, August 1995.

[21] S. Warnakulasuriya and T.M. Pinkston, "Characterization of deadlocks in interconnection networks," in *Proc. of the 11th Int. Parallel Processing Symposium*, April 1997.