A New Deadlock Recovery Mechanism for Fully Adaptive Routing Algorithms

***Z. H. Al-Awwami, **M. S. Obaidat, and *M. Al-Mulhem**
Monmouth University and KFUPM
*** Department of Information and Computer Science, KFUPM, Dhahran 31261, Saudi Arabia**

****Corresponding Author: Professor M. S. Obaidat, Department of Computer Science, Monmouth University, W. Long Branch, NJ 07764, USA**
E-mail: Obaidat@monmouth.edu

Abstract

Routing algorithms used in wormhole switched networks must all provide a solution to the deadlock problem. If the routing algorithm allows deadlock cycles to form, then it must provide a deadlock recovery mechanism. Because deadlocks are anomalies that occur while routing, the deadlock recovery mechanism should not allocate any expensive hardware resources for the sake of handling such a rare event. Rather, it should only dedicate a minimal set of required resources to the recovery process in order to engage most of the hardware resources to the task of routing normal packets. This paper proposes a new deadlock recovery mechanism to be used with the True Fully Adaptive Routing algorithm. The new deadlock recovery mechanism takes advantage of the concept behind wormhole switching. The scheme is efficient in terms of hardware requirements, causes fewer deadlocks and can compete with other expensive deadlock recovery schemes.

Key words: Deadlock recovery, wormhole switching, adaptive routing, performance evaluation.

1. Introduction

Massively parallel processors (*MPP*) are an important class of parallel machines that can provide the envisioned computation platform for solving the so-called grand challenge problems. Several variations of these machines are commercially available today. These machines are composed of many nodes that communicate with each other over a set of switches and communication links collectively known as the interconnection network. The performance of the interconnection network is the most critical factor affecting the performance of the entire parallel machine.

Wormhole switching is the most popular switching technique that has been applied to the interconnection networks of parallel multicomputers. Wormhole switching is well suited for application in multicomputer interconnection networks as it allows for the design of simple, low-cost hardware router nodes, while providing low latency, high-bandwidth communication [1].

Routing algorithms, used to route packets in wormhole networks, are characterized by how they approach and overcome the issue of deadlock. Earlier work proposed in the literature has adopted the view that routing algorithms should be intrinsically deadlock-free. By providing a sufficient condition that prevents the occurrence of deadlocks, and by having the routing algorithm uphold this condition while routing, freedom of deadlock is guaranteed. Such algorithms are called *deadlock-avoidance* routing algorithms [2-4]. *Deadlock-recovery* algorithms on the other hand allow packets to be routed on all available channel resources without guarding against the possibility of deadlock. Once deadlocks occur, they are detected and a particular deadlock-recovery mechanism is invoked to resolve the deadlock situation [5-6]. Both of these techniques are similar in the sense that in each case the routing algorithm must utilize some resources, but they differ in the amount and expense of those resources. It has been established through research that deadlocks are generally rare events, and are more likely to occur as networks reach or are beyond their saturation point [7, 8]. To this effect, almost all deadlock-avoidance algorithms allocate more expensive resources to avoid deadlocks than is necessary. This observation has paved the way for deadlock-recovery techniques to be more widely accepted as a legitimate contender to deadlock-avoidance techniques.

Routing algorithms are also characterized by the amount of adaptivity they provide. *Deterministic* algorithms do not provide any adaptivity. The route taken is determined at the source node by considering only the source and destination node addresses. Deterministic routers allow for the design of fast and simple routers. They suffer from a serious drawback however; they do not allow the router to

react to the current status of the network [2, 9].

Adaptive routing algorithms on the other hand can react to network conditions as they allow packets to be routed along alternate paths. The routing decision is performed in a distributed fashion by each intermediate node along the way. Adaptive algorithms are further classified as to the amount of adaptivity they provide. Algorithms with limited adaptivity are called *partially adaptive*, while *fully adaptive* algorithms allow routing on all possible paths. When the routing algorithm is only allowed to route along available shortest paths between any pair of nodes, the algorithm is called *minimal* otherwise, it is called *nonminimal* routing, or *misrouting*. In minimal routing each route taken by a packet must bring the packet closer to its destination [9, 10]. Fully adaptive routing algorithms that employ deadlock-recovery techniques truly utilize all of the available buffer resources for routing packets, without excluding any for the sake of deadlock prevention. Therefore they are known as *True Fully Adaptive Routing* (*TFAR*) algorithms [5, 6, 11].

The amount of adaptivity provided by the routing algorithm is generally tightly correlated to the underlying router hardware complexity, which has the undesirable consequence of reducing the clock speeds at which the interconnection network can operate, and therefore to the overall performance of the multicomputer [11, 12].

The *Dimension-Ordered* Routing (*DOR*) algorithm is a deterministic deadlock-avoidance algorithm proposed by Dally and Seitz in [2]. Each packet is routed in one dimension at a time arriving at the proper coordinate in each dimension before proceeding to the next dimension. By enforcing a strictly monotonic order on the dimensions traversed, deadlock-free routing is guaranteed.

Planar-Adaptive-Routing (*PAR*) is a partially adaptive deadlock-avoidance algorithm. PAR restricts adaptability by only allowing a packet to be routed through a sequence of 2-dimensional planes at a time, until the packet reaches its destination. This reduces the requirements for deadlock prevention. PAR requires only three virtual channels per physical channel regardless of the network dimension [10].

Disha is a TFAR algorithm with a progressive deadlock-recovery mechanism. The main idea of this deadlock-recovery is to allocate a central buffer in each node. This buffer is connected to the router's crossbar via an input port, and is accessible by all neighboring nodes. When a deadlock is detected, one of the packets involved in the deadlock is moved to the deadlock buffer of the next node along its path. From that point on, the packet continues to advance using only this "floating lane" composed of all the deadlock buffers throughout the network until it reaches its destination. Hardware requirements for Disha include a central buffer that is connected to the crossbar through an extra input port at each node, a hardware token signal wires connected to all nodes in order to synchronize the recovery process, an extra status line to select the deadlock buffer, and a crossbar reconfiguration buffer is needed to temporarily store any broken connection inside the crossbar while forwarding the deadlocked packet [6]. The added hardware contributes, even if slightly, to the complexity of the crossbar switch. This hardware is on the critical path of routing all packets, and therefore will affect the overall speed of the switching process [11]. Disha nonetheless has been shown to outperform most other routing algorithms.

The proposed algorithm presented in this paper is a TFAR algorithm along with a new deadlock-recovery mechanism. It attempts to provide deadlock-recovery using minimal hardware resources, and more importantly this extra hardware is not on the critical path of routing normal packets so as not to affect the overall speed of the router.

In section 2, the proposed deadlock-recovery mechanism is presented. Section 3 outlines the evaluation methodology and the assumptions made about the simulation environment. Section 4 discusses the simulation results. Finally, section 5 presents a conclusion.

2. Proposed deadlock-recovery mechanism

The work proposes using the True Fully Adaptive Routing algorithm and a new deadlock-recovery mechanism. TFAR algorithm routes packets on all available channels and buffer resources without regard to the possibility of deadlock. Once deadlocks are detected, the proposed recovery mechanism is invoked. The deadlock detection mechanism used here is based on the one described in [13].

The deadlock-recovery scheme proposed is a new approach. Since deadlocks are rare, only as few resources as possible should be dedicated to handle this rare event. The proposed approach takes advantage of the concept behind wormhole routing. Namely the low number of edge buffer requirements per channel, which can be as low as one flit buffer, and the flow control mechanism already in place, and which carries control information in the opposite direction to that of data flow.

2.1 Operation of proposed recovery mechanism

The proposed mechanism allocates a central buffer of the same capacity as the edge buffer, which is usually one or two flits deep. This central buffer will be used to break a deadlock cycle at the node where the header of the blocked packet resides. The flow control mechanism is also extended by one more signal, the *break/reconnect* line.

When a deadlock situation is detected in a node, the edge buffer that contains the header of the blocked packet is

stored in the central buffer, and then cleared to receive other flit(s). Flow control in the opposite direction, using the added line, signals the previous node where the packet came from to break and free the connection through its crossbar, to enable other packets to pass through it. This operation is referred to as a *breakMode* operation. Each node containing part of the blocked packet propagates a breakMode signal to the previous node until the tail flit or the source node of the packet has been reached. A breakMode operation performs several steps. First, it moves the flit(s) of the deadlocked packet from the edge buffer to the central buffer. Second, it saves the input and output port numbers made by the packet through the crossbar of that node. Third, it clears this connection through the crossbar, so that other packets can utilize it. This way the remaining packets that are involved in the deadlocked situation can advance, and will eventually free up their occupied resources. Once the node with the separated header in its central queue has a free edge buffer, the header is rerouted again. If the routing is successful, a flow control reconnect signal is sent back to the node containing the remainder of the packet in order to reestablish the connection through its crossbar, and resume the interrupted operation of routing the packet. This signal, referred to as a *connectMode* operation, propagates back along the same path taken by the breakMode signal and reverses what the latter has done using the trails that were previously saved by it. Figure 1 provides an illustration of this mechanism. If packet 1 is marked as deadlocked, then the header of the packet (H1) will be moved to the CB of that node. A breakMode signal is sent to the previous node where the data flits(s) of the packet reside (D1). The breakMode operation will move the data flit(s) to the CB, and will clear the crossbar connection established by the packet. This way packet 4 which has been blocked by packet 1 can now advance.

In wormhole routing once a channel accepts the header of a packet, then it must accept all the remaining flits of the packet in a pipeline fashion. This definition does not allow for the interleaving of flits between packets. This definition is modified by this approach to state that once a channel accepts the header of a packet, then it must accept all the remaining flits of the packet eventually, and in a pipeline fashion during periods of data flow (connect mode). But here the limited interleaving of flits in order to break deadlock cycles is allowed.

2.2 Hardware requirements

The hardware required by this approach is moderate and more importantly does not increase the complexity of either the crossbar switch, the virtual channel controller (*VCC*), nor does it lie on the critical path of the routing decision for normal packets. Both of these factors have been shown to increase the cost and decrease the speed of the router [11].

A central buffer is required per node. The capacity of this central buffer should be the same as that of an edge buffer. The central buffer is connected to all the input ports of the switch and can output to any output port. It has a similar architecture to the central queue used in the Chaos router [14]. Two 7-bit registers per node are needed to store the input and output switch port. A break/reconnect signal wire is also needed in order to break and reconnect packets. This wire is very similar to the flow control wires already used for wormhole switching. Its main function is to signal the previous node to perform either a breakMode or a connectMode operation. Instead of having to use two wires for each mode, we just use one wire with a bit in each node. When a break signal is received by a node, it toggles this bit, so that the next time it receives it, it will actually be interpreted as a connect signal.

This hardware is sufficient to perform sequential deadlock recovery. That is one deadlocked packet is handled at a time. Sequential deadlock recovery will be used throughout our simulation. Concurrent recovery is beyond the scope of this paper.

3. Performance evaluation

The simulator developed as part of this effort is written in Java. It is configurable for a wide range of topologies, network sizes, routing algorithms, selection functions, arbitration policies, and various configuration parameters, such as the number of virtual channels, capacity of each virtual channel, and the packet length.

3.1 Multicomputer model

The following assumptions are made about the multicomputer and its interconnection network:

- Each node has a processor, a router, and the crossbar switch. The processor and the router are connected via one injection and one delivery channels
- Injection queues are allowed to grow without bounds. Latency figures calculated include source queuing time
- Flits passed to the delivery channel are assumed to be consumed immediately by the processor
- The router of each node is connected to neighboring routers via dual uniplex channels
- Each uniplex channel is associated with a multi-flit FIFO queue (input buffered)
- A physical channel is associated with a configurable number of multi-flit queues, referred to here as virtual channels, or edge buffers. The virtual channels are multiplexed over the physical channel. If more than one virtual channel can utilize the physical channel,

channel bandwidth is assigned to the virtual channels in a demand-slotted round robin fashion

- A flit can be transferred across a physical channel in one clock cycle
- A header flit can be processed by the node's routing unit within one clock cycle
- Only one header is processed by the routing unit at a time, if more than one header requires the routing unit, they are serviced in a round robin fashion

3.2 Message generation

A packet is broken down into flits. Each packet contains a header flit, data flit(s), and a tail flit. Header is assumed to fit in one flit, so routing can take place as soon as the header flit arrives at the input queue of a node. Packet generation rate is constant and the same for all nodes for a given run as determined by the injection period of that run. Packet destinations are chosen uniformly amongst all the nodes in the network for uniform traffic, otherwise the traffic pattern is non-uniform and would be specified. After generating a packet, a node waits for a random number of cycles before generating the next one. This random number of cycles is bounded between 0 and the current injection period value. The current injection period is determined by dividing the maximum injection period of the network by the specified load factor. The maximum injection period is determined from the maximum wire capacity supported by the network. The maximum wire capacity for a particular network is calculated as in [10]. For this simulation we normalize the maximum injection period to correspond to 2/3 (66.67%) of the maximum wire capacity for a better chart drawing scale. When the load factor is set to the maximum value of 1.0, then the injection period would equal the maximum injection period. The normalized load factor is used to drive the packet generation for a particular run, and is used to plot all the results in this simulation.

3.3 Performance metrics

The two most important performance metrics are latency and throughput. Latency is measured as the number of clock cycles from the time the packet has been generated by the processor, until the tail flit of the packet has been delivered to the destination node. Throughput is the maximum number of packets delivered per unit of time. It is normalized to be the maximum number of flits delivered per clock cycle. It can be derived directly from the current load factor unless the network is saturated at that point.

Because these metrics may fluctuate widely at the initial transient state of the network. The metrics are only collected after the network has reached the steady state. The end of the transient state is detected when the flit injection rate and the flit delivery rate have stabilized. More precisely the transient period is declared over when the flit injection rate and the flit delivery rate are within 0.05% of each other for two consecutive 500-clock cycle periods. Statistics are collected for 50,000 clock cycles plus the number of cycles it took to reach the steady state. This was found to be sufficient for the network size simulated.

A steady state solution exists if the flit generation rate and the flit delivery rate are equal, otherwise, the network is saturated for that run. If a steady state solution exists, then a simulation run can be terminated whenever the metric measured is within 0.05% of its previous value for 10 consecutive 500-clock cycle periods. This is to save simulation time and resources.

4. Simulation results

This section will present the simulations for a 256-nodes 2-dimensional, 16x16, mesh topology network with three virtual channels. The network size was selected for reasonable simulation times. Packets used will all be 32-flits long, while flits are 32-bits wide. The selection function, which selects a single path from the set of all possible paths to route the packet through, will attempt to locate a free channel for the packet first. If that is not possible, it selects amongst the choices according to the *Straight-First* flavor of selection. The timeout value, after which a packet is marked as deadlocked, is 10 clock cycles unless otherwise specified. The proposed routing algorithm is evaluated against three others: DOR, PAR, and Disha.

4.1 Cost model

This section compares the cost of the additional hardware resources, needed by both the Disha and the proposed model, in terms of the speed achievable by the two routers. We first consider the *unified-crossbar* design, which consists of a single large crossbar connecting all input to output ports. According to the cost model in [11], the router's data through latency can be computed using $T_{dt} = T_{fc} + T_{cb} + T_{vc}$, where T_{fc} is flow controller delay, T_{cb} is the crossbar switch delay, and T_{vc} is the virtual channel controller delay. For a 0.8 micron CMOS gate array technology, $T_{fc} = 2.2$ *ns*, $T_{cb} = 0.4 + 0.6 \log P$ *ns*, and $T_{vc} = 1.24 + 0.6 \log V$ *ns*, where P is the number of input ports to the crossbar switch, and V is the number of virtual channels (V=3). For our simulated network, the data through latency of Disha is $T_{disha} = 7.1$ *ns*, while $T_{proposed} = 7$ *ns*. Although the difference is small, it gives the proposed mechanism a slight advantage, especially that the performance results of the

next section are very close and since each clock cycle is actually larger for the Disha case.

The situation is compounded when we consider a modular router design such as the *enhanced-hierarchical* router. Modular routers will be more likely used in future multicomputers. Each subcrossbar input size can be calculated using $P = 2n + C + 1$, where n is the dimension of the network, and C the number of connect channels used between subcrossbars [15]. For our network and assuming one connect channel, $T_{disha} = 5.5$ *ns*, while $T_{proposed} = 5.4$ *ns*. These values are the delay through a single sub-crossbar. More than one sub-crossbar may be traversed to get the flit to the desired virtual channel, making the data through delay additive in nature. This again gives more of a performance advantage for the proposed recovery scheme.

4.2 Uniform traffic

Simulation results for the uniform traffic pattern is shown in figure 2. The PAR algorithm displays the worst performance, it saturates around 0.35 of the normalized wire capacity. DOR does relatively well as the traffic is already uniformly distributed, saturating around 0.65. The performance of Disha and the proposed algorithm are almost identical. They both saturate at 0.7 of wire capacity. As we mentioned in section 4.1, this gives an advantage to the proposed scheme as it has lower data through latency cycle. Deadlocks occur in both algorithms around the saturation point, but we see that the proposed deadlock recovery mechanism can match that of Disha. This is in spite of the fact that the hardware resources used by Disha can be used to progress deadlocked packets. The resources used by the proposed scheme however, are not progressive in nature; they are merely used to preempt the deadlocked packet to a storage that is distributed throughout the network. This suggests that the Disha extra lane can actually be used for routing normal packets, which will achieve a higher saturation point in the presence of an efficient deadlock recovery mechanism.

4.3 Non-uniform traffic

Three traffic patterns are evaluated, *bit-reversal*, *dimension-reversal*, and *hot spot* traffic. In *bit-reversal* traffic, a node with binary address $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ sends a packet to a node with binary address $a_0, a_1, \ldots, a_{n-2}, a_{n-1}$. This traffic pattern causes nodes on certain rows to send packets to nodes on certain columns, causing various packets of conflicts near the center of the network. The simulation results are shown in figure 3. The uneven traffic distribution causes DOR to saturate early, while adaptive algorithms do a better job in dissipating the congestion. Again Disha and the proposed algorithm have almost identical performance with a 0.65 saturation point.

The *dimension-reversal* traffic pattern causes a node with address (xy) to send packets to node (yx). For 2D networks this is the same as the matrix transpose traffic pattern. It has a similar effect as the bit-reversal pattern, but tends to concentrate the conflicts along the diagonal line of the network. In our simulation, whenever the source and destination nodes addresses are the same, a packet with a random destination is injected instead, so as to allow for more deadlocks to occur. Figure 4 shows that the performance of both the proposed and the Disha algorithms are similar, saturating around 0.65 of the normalized wire capacity.

The hot spot traffic pattern used causes 5% of all network traffic to be destined toward a single node that is randomly selected, while the remaining traffic is uniformly distributed. This pattern causes a serious congestion near the hot spot node, which in turn causes all the routing algorithms to have early saturation points. The timeout used in our simulation for this pattern is 35 clock cycles. Examining figure 5, we notice that PAR saturates at 0.2875 and DOR at 0.3 of the wire capacity. The proposed algorithm and Disha both saturate roughly at around 0.3125, but we can notice that the proposed scheme demonstrates a slightly higher saturation behavior. This is due to the fact that the number of deadlocks detected by the proposed mechanism is much lower than those detected by the Disha algorithm. Figure 6 illustrates this by plotting the number of deadlocks detected normalized to the total number of packets delivered by the network as a function of the normalized load factor. We can see that the number of deadlocks detected by Disha has almost an exponential growth, while the proposed mechanism maintains a linear nature. This important observation is generally true for all traffic patterns, but is more obvious in this case. The reason for this behavior is that in the proposed scheme, deadlocked packets are completely removed from the network quickly so other packets can advance. Disha on the other hand, progressively advances the deadlocked packet to its destination over a slower lane, causing more congestion to form behind it, and therefore more deadlocks. The proposed mechanism however, does not have a clear performance advantage because the latencies of the deadlocked packets are larger due to the fact that they wait for other packets to pass by. In this traffic pattern, the deadlocked packets may wait for longer periods before finding a free virtual channel to reroute the packet on. The congestion presented by this traffic pattern can be effectively dealt with by increasing the number of delivery channels at each node.

5. Concluding remarks

In this paper, we presented a new deadlock recovery mechanism to be used with True Fully Adaptive Routing algorithms in wormhole switched networks. The main advantages of the proposed scheme can be summarized as follows. First, this scheme creates a new category of deadlock recovery techniques. Previously known categories are either *Progressive*, meaning that when deadlocks occur, the deadlocked packets are advanced to their destinations using some reserved set of resources, as in Disha. *Abort-and-Retry* category means that when deadlocks are detected, the packet is disposed of and retransmitted at a later time. The proposed approach cannot be characterized as either of these two. Rather, it creates a new category with a preemptive nature. As a consequence, this approach uses moderate hardware resources, and more importantly the needed hardware does not increase the complexity of the crossbar switch, nor does it lie on the critical path of routing packets. Second, The scheme takes advantage of the concept behind wormhole routing, namely, the low number of edge buffer. requirements per channel, which can be as low as one flit buffer, and the flow control mechanism already in place, which carries control information in the opposite direction to that of data flow. Third, there are no extra delays for this approach, when using a router model that uses Input/Output buffered nodes. Disha suffers from this problem, as output buffers will have to be cleared, and stored in some extra hardware during the progressive deadlock recovery phase. This extra overhead is not modeled in this study. Fourth, the scheme causes less deadlocks to form as demonstrated by the Hot Spot Deadlock chart. Finally, Our scheme is more suitable to modular router designs, which are more likely to be adopted in the future

References

[1] William J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, Vol. 39, No. 6, pp. 775-785, June 1990.

[2] William J. Dally, Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547-553, May 1987.

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

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

[5] J. M. Martinez, P. Lopez, J. Duato, T. M. Pinkston, "Software-Based Deadlock Recovery Technique for True Fully Adaptive Routing in Wormhole Networks," In Proceedings of the 1997 International Symposium on Parallel Processing, pp. 182-189, 1997.

[6] Anan K. V., Timothy Mark Pinkston, "An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA," *In Proceedings of the 1995 International Symposium on Computer Architecture*, pp. 201-210, June 1995.

[7] Timothy Mark Pinkston, Sugath Warnakulasuriya, "On Deadlocks in Interconnection Networks," *In Proceedings of the 1997 International Symposium on Computer Architecture*, pp. 38-49, June1997.

[8] Sugath Warnakulasuriya, Timothy Mark Pinkston, "Characterization of Deadlocks in Interconnection Networks," *In Proceedings of the 1997 International Conference on Parallel Processing*, pp. 80-86, 1997.

[9] Lionel M. Ni, Philip K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, Vol. 26, No. 2, pp. 62-76, February 1993.

[10] Jae H. Kim, "Planar-Adaptive Routing (PAR): Low-Cost Adaptive Networks for Multiprocessors," *Master Thesis*, University of Illinois at Urbana-Champaign, 1993.

[11] Andrew A. Chien, "A Cost and Speed Model for k-ary n-cube Wormhole Routers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 2, pp. 150-162, February 1998.

[12] Kazuhiro Aoyama, "Design Issues in Implementing an Adaptive Router," *Master Thesis*, University of Illinois at Urbana-Champaign, 1993.

[13] P. Lopez, J. M. Martinez, J. Duato, "A Very Efficient Distributed Deadlock Detection Mechanism for Wormhole Networks," In the 1998 International Symposium on High-Performance Computer Architecture, pp. 57-66, 1998.

[14] Kevin Bolding, "Chaotic Routing – Design and Implementation of an Adaptive Multicomputer Network Router," *Doctoral Dissertation*, University of Washington, 1993.

[15] Yungho Choi, Timothy Mark Pinkston, "Crossbar Analysis for Optimal Deadlock Recovery Router Architecture," *SMART Interconnects Group http://www.usc.edu/dept/ceng/pinkston/SMART.html*, pp. 583-588, 1997.

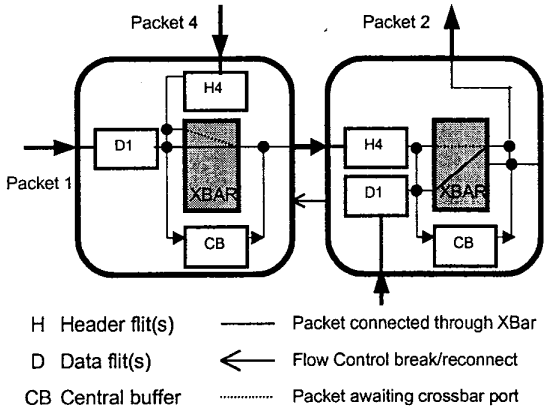

Figure 1. Detailed view of two deadlocked packets in the proposed router design

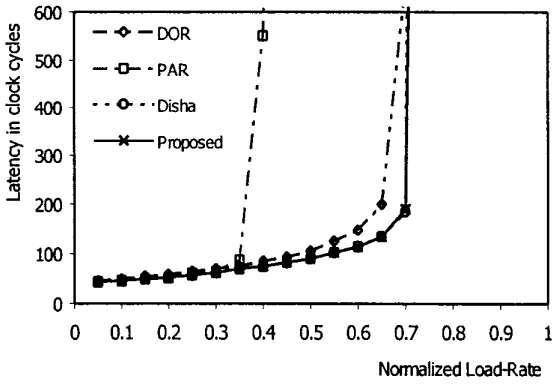

Figure 2. 2-dimension 16x16 mesh (Uniform traffic

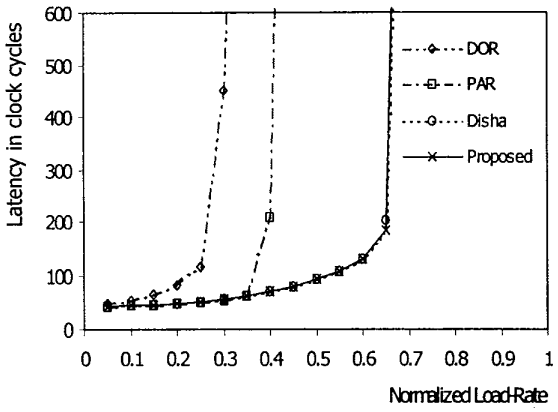

Figure 3. 2-dimension 16x16 mesh (Bit reversal traffic)

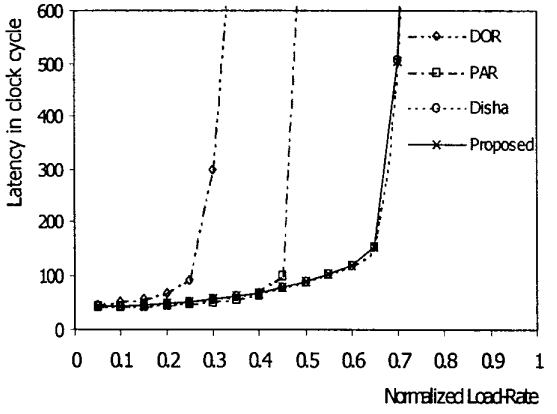

Figure 4. 2-dimension 16x16 mesh (Dimension reversal traffic)

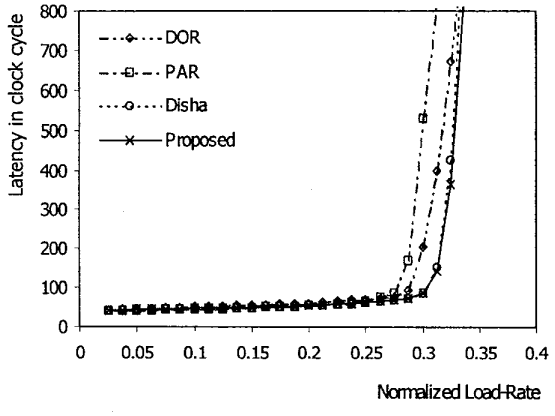

Figure 5. 2-dimension 16x16 mesh (Hot spot traffic)

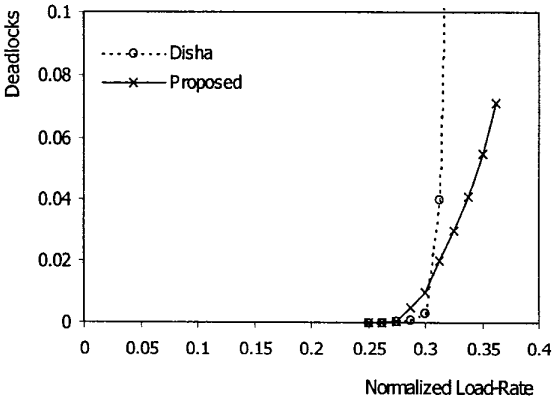

Figure 6. Frequency of deadlocks in hot spot traffic