DISHA: An Efficient, Fully Adaptive Deadlock Recovery Scheme

Anjan K.V. and Timothy Mark Pinkston

CENG Technical Report 94-23

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4482

September 1994

DISHA: An Efficient, Fully Adaptive Deadlock Recovery Scheme

Anjan K. V. Timothy Mark Pinkston

Electrical Engineering - Systems Department
University of Southern California
3740 McClintock Avenue, EEB-208
Los Angeles, CA 90089 - 2562
{*anjan@truth.usc.edu, tpink@charity.usc.edu*}

Abstract:

This paper presents an efficient and cost effective routing strategy that considers deadlock recovery as opposed to prevention. Routing is fully adaptive in the absence of deadlocks. Cycles are broken by re-routing a blocked packet through a deadlock-free network. *Disha* effectively uses the *same virtual lanes for recovery as for adaptive routing*. The proposed scheme is extremely simple, ensuring quick recovery from deadlocks and enables the design of fast routers.

1.0 Introduction:

The interconnect network is the backbone for communication in a multiprocessor/multi-computer environment. System performance is determined not only by the effective utilization of multiple processor nodes, but also, to a large extent, by the efficient communication amongst them.

The earliest interconnection networks employed store-and-forward routing, which suffers from large packet delays. Pipelined versions of this, such as virtual cut-through [17] and wormhole routing [6], were proposed as alternatives in order to reduce packet delay. While these schemes govern the flow of packets in a network, the routing algorithm determines the actual path. These can be deterministic or adaptive schemes.

Under deterministic routing (also known as oblivious), packets follow a fixed path depending entirely on the source and destination addresses. This scheme is highly restrictive, does not allow packets to adjust to varying traffic conditions and is not fault tolerant. It, however, has the advantage of being easy to implement, leading to simple and fast routers.

In adaptive schemes, message paths are selected based on local network traffic. Adaptivity allows an increased utilization of available paths from source to destination and provides improved fault tolerance [8]. These schemes offer the choice of routing packets along any of the shortest paths, i.e., minimal schemes, or the selection of any path backtracking if necessary, i.e., non-minimal schemes, to route around congestion or faulty nodes. Moreover the degree of adaptivity can vary depending on which subset of paths (minimal or non-minimal) are allowed in routing messages, i.e., partially adaptive or fully adaptive.

From the above discussion, it would appear that the scheme to adopt would have wormhole flow with fully adaptive routing capabilities. This potent combination, however, poses a few problems. Under wormhole movement messages continue to remain in the network, blocking other

packets that require the use of resources they occupy. Packets could end up waiting for resources endlessly, in which case, they would be *deadlocked.* Also, adaptive schemes allow packets to be routed away from the destination and, if continuously routed in this manner, they may never reach the desired node leading to a *livelock* situation.

This paper proposes a scheme to safely incorporate fully adaptive wormhole routing in interconnection networks. Although the examples considered are mesh type, this scheme can equally be applied to any other network topology. This scheme does not require a Virtual Channel Controller (VCC) and does not increase the router crossbar size even in comparison to deterministic schemes. It requires nominal additional decision logic which does not significantly reduce router speed. Hence, routers remain extremely fast and simple. What makes this possible is the way the proposed scheme handles potential deadlock situations.

Most wormhole adaptive schemes focus on deadlock prevention rather than deadlock recovery. Preventive schemes suffer from high hardware costs and/or losses in adaptivity. Considering first how adaptivity can suffer, one example is Dally and Aoki's [8] proposed *dynamic routing algorithm.* Here, every physical link is split into (r+1) virtual channels. Virtual channels are used to break deadlock dependency cycles. Each packet keeps track of the number of dimension reversals it suffers. A packet when blocked with all suitable output channels being currently used by packets with equal or lower values of dimension reversals is routed deterministically. Although this scheme prevents deadlocks from occurring, the number of virtual channels traversed ultimately places an upper bound on the algorithm's adaptivity.

Considering hardware costs, a study by Chien [3] shows that the increased complexity of virtual channel controllers adversely affects the machine performance by increasing the router's clock cycle. Because of this fact, deterministic schemes might, in fact, outperform certain adaptive schemes. To address this problem, Chien and Kim proposed the Planar Adaptive Routing scheme [2], which reduces the hardware requirements for deadlock prevention. This however, comes at the cost of restricting the degree of adaptivity to two dimensions at a time.

Duato's algorithm [9], is a more flexible adaptive routing scheme that ensures deadlock prevention. Virtual channels are split into two groups -- adaptive channels and escape channels. Packets can be routed adaptively using any available adaptive virtual channel. If blocked they are routed deterministically using the escape channels. Packets on the escape channels can come back onto the adaptive channels if they happen to be free at a subsequent router. The necessary and sufficient condition presented in [10] relaxes the situation even further by giving more flexibility to the packets on the deterministic channels. Although the number of virtual channels (and the concomitant VCC complexity) required by this scheme can be small, this scheme still has some disadvantages. For example, using Chien's model [3], if one determines that the optimum number of virtual channels for the expected load conditions is three, then for a torus type mesh architecture, Duato's scheme could use only one virtual channel for adaptive routing. Chien [3] goes on to show that every additional virtual channel would demand a 30 - 50% increase in load to justify its presence, due to the increased router latency. This being the case, it seems unjustified to devote some portion of the virtual channels exclusively for deadlock prevention.

The "Turn Model" [15] is a deadlock prevention scheme that does not require virtual channels. Freedom from deadlock is ensured by prohibiting certain turns. However, some disadvantages with this scheme are that it is only partially adaptive and applicable to limited network topologies. The West-first algorithm, for example, demands that all packets be routed west first before they can be routed otherwise. The link along this direction could become highly saturated and, in the presence of a fault, come to a standstill. Simulation studies by Boppana and

Chalasani [5] show that this scheme produces unbalanced traffic conditions and can be outperformed by even non-adaptive algorithms. Such schemes also cannot be fault tolerant.

Deadlock recovery is an alternative to prevention. Simulation studies in [12] and Section 5 of this paper show that deadlocks can be infrequent. Recovery thus seems to make more sense than prevention. "Compressionless Routing" [12] is an effort in this direction. This scheme, however, has the following disadvantages. Padding decreases effective channel utilization -- when the packet sizes are small compared to the network diameter or the buffers at routers are deep, the overhead due to padding is large. Packets have to be stored to allow retransmission if necessary and killed packets suffer increased latencies. Moreover injector and receiver interfaces, additional counters, and status lines increase the hardware complexity.

We believe that routing should be fully adaptive. If deadlock occurrences are extremely rare then it does not make sense to limit the adaptivity of the routing scheme to solve an infrequent event. Limiting adaptivity also reduces fault tolerance capabilities of the system. We also believe that the router design should be extremely simple. It does not make sense to devote virtual lanes specifically to prevent deadlocks; virtual channels should be used only as a means of improving flow control [7]. The goal here is to make the common case fast and *Disha* is a solution to this.

Disha provides the framework for supporting fully adaptive wormhole routing and can be applied to any interconnection network topology. It is a deadlock recovery strategy and, consequently, does not require virtual channels for prevention. It offers the following advantages: 1) deadlock-free fully adaptive wormhole routing, 2) applicability to any interconnection network topology, 3) no virtual channels required for prevention of deadlocks, 4) simple router design, and 5) no padding, storing or retransmitting of packets required.

The remainder of the paper is organized as follows. Section 2 describes the proposed scheme and validates it for any arbitrary interconnection network. Channel, router, and other implementation issues are discussed in Section 3. Section 4 presents our simulation and performance results. Conclusions and future work are given in Section 5.

2.0 DISHA Recovery Scheme:

Disha [†] aims at optimizing routing performance in the absence of deadlocks. This is achieved by allowing routing to be fully adaptive and dealing with the rare cases when deadlock does occur. This scheme requires limited hardware to be devoted to deadlock recovery.

Each router is provided with an additional special flit buffer (Deadlock Buffer) to be used in the case of deadlocks. These Deadlock Buffers form a dedicated deadlock free lane. On deadlock, one of the packets in the cycle is switched to the deadlock free lane and routed along this path until it reaches its destination where it will be consumed (sunk) to break a dependency cycle. To ensure that the special buffer path is deadlock free, only one packet is allowed to use it at any given time.

Disha seems to fall in place with Duato's algorithm [9] of having two virtual networks - one susceptible to deadlocks (possibly adaptive) and the other deadlock-free (possibly deterministic). However, there is a significant difference. As will be explained in detail later, *Disha*

[†] *Disha* means "direction" in Hindi.

uses the same virtual channels for recovery as for adaptive routing. It is the Deadlock Buffers that form a deadlock free lane and virtual channels are not devoted exclusively for this purpose. The commandeering of virtual channels for deadlock recovery will be occasional given that the number of deadlock occurrences are infrequent. Hence, fully adaptive routing on all the provided virtual channels is permitted almost all of the time and should result in significant performance gains. The key idea behind this is that the cost complexity of the VCC with each additional virtual channel grows at a much faster rate as compared to that of the crossbar with each additional input, as pointed out by Chien [3]. Because *Disha* does not require virtual channels for deadlock freedom and has nominal crossbar complexity, it should outperform preventive schemes which do require virtual channels and have large router crossbar sizes.

To illustrate how a potential deadlock situation can be recovered from, we provide the following example. Figure 1a below shows a deadlock situation with a dependency cycle formed by four packets waiting on one another, thus allowing none of them to proceed in a mesh type interconnection network. As shown P_1 -> P_2 -> P_3 -> P_4 -> P_1, where a -> b implies "a" is waiting on "b". The packets are deadlocked.

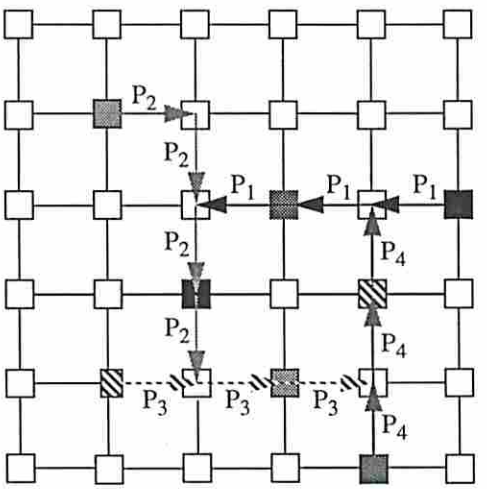

Fig. 1a shows a deadlock scenario in a mesh type interconnection network.

Figure 1b shows how the recovery scheme works. The special buffer path can be viewed as an alternate network interleaved with the original. Using this alternate path, a packet, upon detection that it is deadlocked, can reach its destination where it will be sunk. This breaks the deadlock cycle and other packets are able to proceed. In the figure, P_1 is able to use the alternate network to reach its destination. P_4 is now able to go through followed by P_3 and P_2. Hence, recovery is achieved.

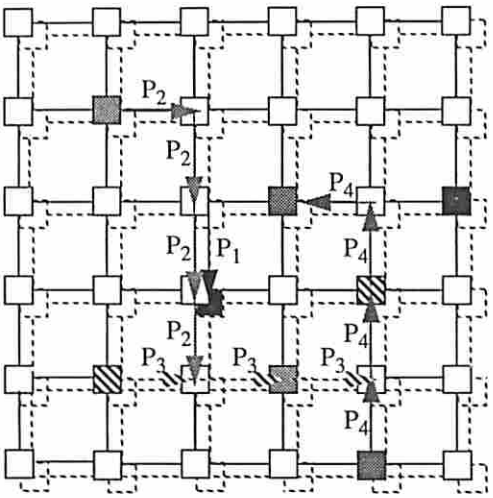

Fig. 1b shows deadlock recovery with *Disha*.

One possible implementation of the required mutual exclusion on the Deadlock Buffer path is to have a Token circulate in the network along a fixed predetermined cyclic path to include all routers. For implementation details, the reader is referred to Section 3.2. Figure 2 below shows why mutual exclusion is necessary and how a deadlock situation could arise in the absence of the Token even under deterministic routing.

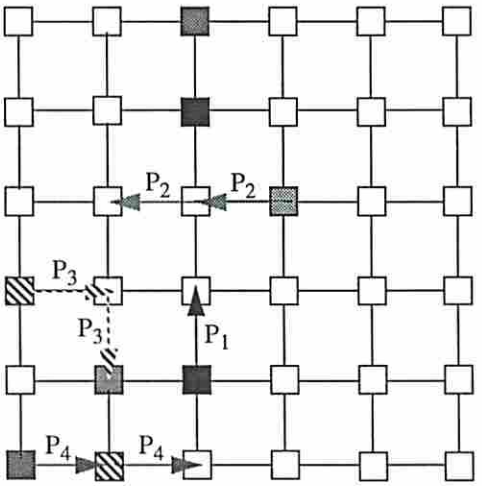

Fig. 2 shows how deadlocks could occur on the special buffer path under x-y routing.

In the figure, P_1 -> P_2. Even though P_1 is being routed along the y - dimension, it still waits on P_2 which is being routed along the x - dimension as P_2 has occupied the Deadlock Buffer of that particular router. Likewise P_2 -> P_3 ->P_4 -> P_1, leading to a dependency cycle on the special buffer path.

Breaking a cycle by routing deadlocked packets one after another can cause substantial individual packet delays especially if cycles are large and involve a large number of packets. However simulation studies show that a vast majority of the deadlock cycles are inherently small in size involving a few number of packets. Recovery is thus quick.

The *Disha* technique can be applied to any interconnection network. The proof that this scheme allows safe deadlock recovery is presented below.

Lemma 1:

There can be no dependency cycles among the Deadlock Buffers.

Proof:

The proof is simple following directly from the fact that only one packet at a time is allowed to use this buffer in the entire network.

Lemma 2:

A connected and adaptive routing function R for an interconnection network I is deadlock-free if there exists a subset of channels $C_1 \subseteq C$ that defines a routing function R_1 which is connected and has no cycles.

Proof:

For a detailed proof of this assertion the reader is referred to Duato [9].

Theorem:

"On the network being deadlocked if only one packet in the network is allowed to use the special Deadlock Buffer, then the network can safely recover from deadlocks."

Proof of the Theorem:

From Lemma 1, the Deadlock Buffer path is acyclic. This path thus constitutes channel C_1 defined in Lemma 2. Thus the routing scheme defined in *Disha* is deadlock free.

To get an intuitive feel of the proof assume the network is deadlocked. There may be one or more dependency cycles. Assume there are "k" cycles in the network related or otherwise. A packet can be put on the cycle free Deadlock Buffer path. Following this set of buffers the packet is guaranteed to reach its destination where it will be sunk. This eliminates one cycle reducing the number of cycles to (k-1). By induction it follows that the network safely recovers from deadlocks.

3.0 Implementing Disha:

In comparison with deterministic schemes, *Disha* requires minimal additional hardware in the form of a hardwired Token path, slight increase in router complexity and a possible additional status line. In comparison to many adaptive schemes, *Disha*'s hardware requirements are significantly less for about the same degree of adaptivity, resulting in faster router operation. This section examines these important implementation issues.

3.1 Channel Specifics:

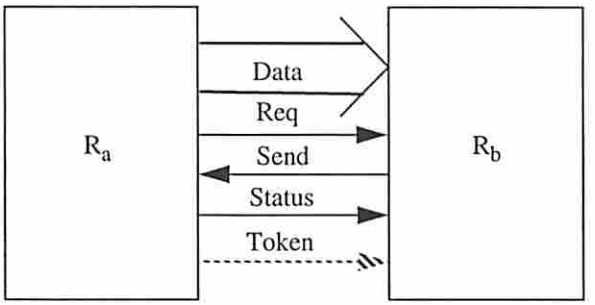

Fig. 3 Implementation Details.

Figure 3 shows the implementation details for a unidirectional channel. *Req* and *Send* are flow control signals used for handshaking between routers R_a and R_b. Router R_a asserts the *Req* line after it has placed data on the channel. R_b uses this to latch onto the data on the channel and asserts *Send* when it is ready to accept more. The *Status* line is used to differentiate between incoming flits intended for input buffers or the Deadlock Buffer. The *Token* line is used to implement the circulating Token. A router asserts this line to pass the Token on to the next node along the Token path. This *Token* line has been indicated by a dotted line as it is not required with every unidirectional channel; only those channels along the Token path have it. *Token* is additional to what is found in other routers, while *Status* would be additional if the number of virtual channels is $< 2^m$, where 'm' is the number of status lines required to distinguish between the virtual channels. For example if the number of virtual channels is 3,5,6,7,9... we would not need an additional *Status* line.

3.2 Token Specifics:

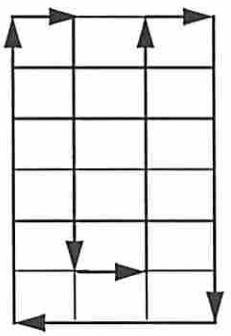

Fig. 4 Hardwired Token path.

Figure 4 shows one possible hardwired Token path for a mesh. The router, if unable to send out the *header*, checks to see if it is deadlocked. This is implemented by comparing the blocked time of the packet with the time-out threshold selected (see Section 3.5). If so, the router holds onto the token when it arrives. The deadlocked packet is then sent out with the *Status* line asserted, conveying to the receiving node that the incoming flit is to be placed into the Deadlock Buffer. This packet will be routed to reach its destination and break one dependency cycle. The router then

releases the token by asserting the *Token* line.

The Token logic requires just a latch and a couple of gates. The Token can therefore be clocked much faster than the router to enable quick deadlock recovery initiation.

Table 1: Token Logic.

Input Token	Deadlock	Output Token
0	0	0
0	1	0
1	0	1
1	1	0

Table 1 gives details of the logic to be followed for implementing the Token. The cost of this line is about '1/k' of a regular status line as it does not include all paths, where 'k' is the node degree.

3.3 Router Architecture:

Figure 5 illustrates, at a conceptual level, the router architecture for a mesh type interconnection network [1]. The S_i's are the *Status* lines that enable or disable the tri-state buffers. In the absence of deadlocks these lines are unasserted and the packet arriving on a channel is placed into the input buffer. When a flit comes in with the corresponding *Status* line asserted, data on the channel bypasses the input buffer to be placed directly into the Deadlock Buffer.

The Decision and Control Logic arbitrates amongst the signals it receives from the address decoders and, based on this, generates control signals to make connections in the Crossbar Switch.

Observe that there is no increase in the crossbar complexity compared to that of a generic mesh type dimension-order router. In fact, the crossbar complexity could be slightly decreased: both require 6 inputs but *Disha* requires only 5 outputs as opposed to 6 for the dimension-order router (Chien [3]). However, the Deadlock Buffer and the logic to implement the bypassing of the input buffers is additional to that found in typical dimension-order routers. More efficient implementations of the router to reduce clock cycle time are possible.

1. Note: Not all components of the router are shown (i.e., flow controller, address decoder, etc.). Only those components which make *Disha* distinctive from basic router designs are featured.

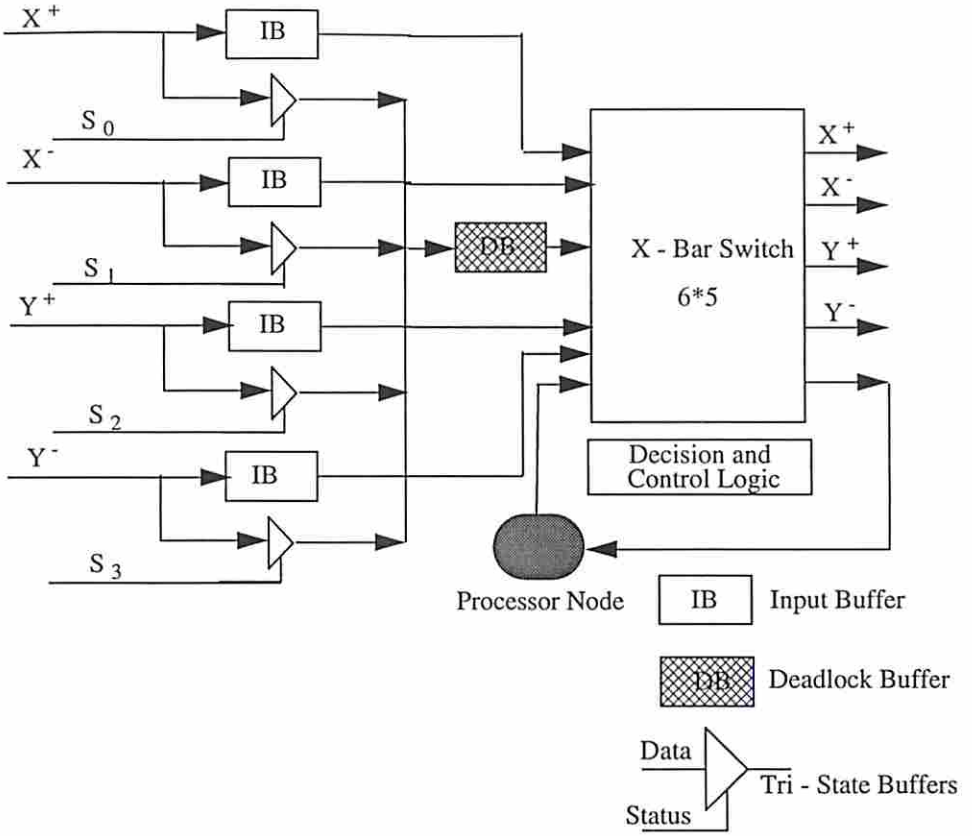

Fig.5 Router Block Design.

3.4 Crossbar and the Decision Logic:

In the absence of data on the Deadlock Buffer, the crossbar is configured as specified by the Decision and Control Unit's arbitration logic. However, under a deadlock situation, it could so happen that the Deadlock Buffer input to the crossbar is required to be connected to the same output as is currently being used by some other packet in a normal input buffer. The crossbar would then have to be reconfigured to connect the Deadlock Buffer to the required output terminal. The decision logic thus needs to remember the state of the crossbar before it was reconfigured so that it can be reconnected once deadlock has cleared.

To illustrate this point, Figure 6a shows the initial crossbar configuration before a deadlock situation. With deadlock and data requiring to be put out on X^-, the crossbar is reconfigured as shown in Figure 6b. The reconfiguration buffer stores the state of the crossbar before change over. This buffer does not need to store the entire crossbar state, but only that of the input that was disconnected. Data on the Deadlock Buffer is guaranteed to reach its destination and once the *tail* flit has gone by, the crossbar is reconnected using the information stored in the reconfiguration buffer.

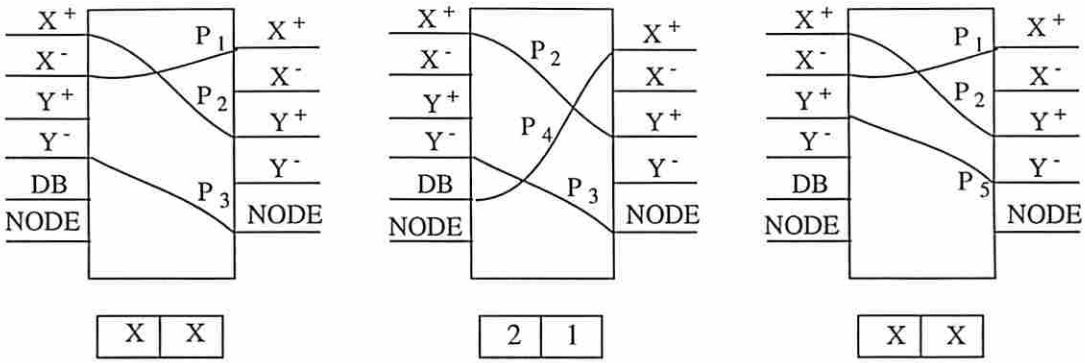

Fig. 6a Fig. 6b Fig. 6c

2 | 1 This notation implies that input 2 (X^-) needs to reconnected to output 1 (X^+).

Fig. 6 shows the reconfiguration of the crossbar.

Keep in mind that the flow is wormhole and the path is not released until the *tail* has gone by. Thus although there might be discontinuities, as for example the previous flits in packet P_1 could have advanced while the flits at this router are held back, the Decision Logic remembering the crossbar state before reconfiguration is sufficient guarantee that all flits will ultimately reach the correct destination. Simulations in Section 4 show that less than 2% of the total packets injected seize the Token. We could then expect the number of packets that are likely to suffer such discontinuities to be around 4 to 5% in the average case. We cannot do without reconfiguring because with the Token held at this router, packet P_1 could be blocked upstream leading to a deadlock situation.

Alternatively, it is possible to delay reconfiguration until the packet using that output (P_1) either is completely routed or becomes blocked. However, simulations show that prompt reconfiguration produces the best results. The Deadlock Buffer is thus effectively given priority over the other input buffers and data on this buffer is allowed to bypass normal data on the output channel if some exists. This ensures quick recovery.

Figure 4 shows the router architecture without output buffers. It might seem as if the introduction of output buffers or virtual channels (in our case, for increased flow and throughput), is a potential hazard. This, however, is not true. The output buffer or the VCC only accepts data that *can be delivered.* In doing so, the VCC cooperatively controls intranode data flow with the internal flow controller (not shown in the figure) of the subsequent node [4]. Hence the output buffer will eventually clear and the crossbar can be reconfigured.

The complete operation of *Disha* is explained below, assuming routers with output buffers. Figure 7a shows a possible initial state of routers R_a, R_b and R_c. Here packet P_1 is blocked by P_2 and is unable to proceed. Router R_a is unable to send out the *header* of P_1 for T_{out} clock cycles (see next section) and assumes packet P_1 to be deadlocked (P_1 -> P_2). R_a now sends out the header of P_1 on X^+ with the corresponding *Status* line asserted. Router R_b places the incoming *header* of P_1 on the Deadlock Buffer, bypassing the normal input buffer currently occupied by

packet P_2. Now assume that at router R_b, packet P_1 requires the same output channel as is currently being used by some other packet P_3. In such a case, as explained above, router R_b reconfigures the crossbar. The output buffer at R_b occupied by P_3 is bound to clear allowing packet P_1 to proceed. The state of the crossbar is stored in the reconfiguration buffer for later reconnection. Packet P_1 now follows the Deadlock Buffer path through subsequent routers to reach its destination. These actions are indicated in Figure 7b. *Disha* thus uses the same virtual channel for deadlock recovery as for adaptive routing. The flits of packet P_3 in router R_c could have moved forward as shown in Figure 7b. Although this might lead to discontinuities in flit receptions all flits of packet P_3 are bound to reach the correct destination.

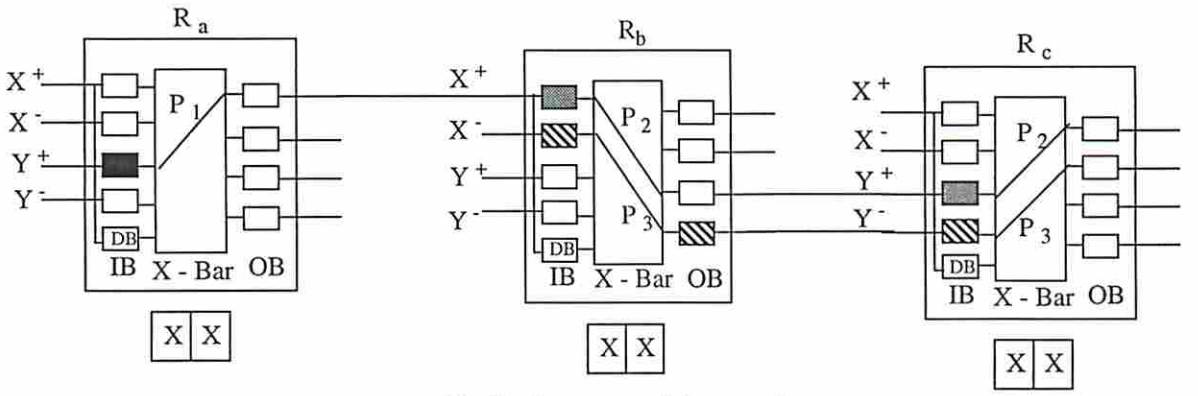

Fig. 7a shows a possible scenario

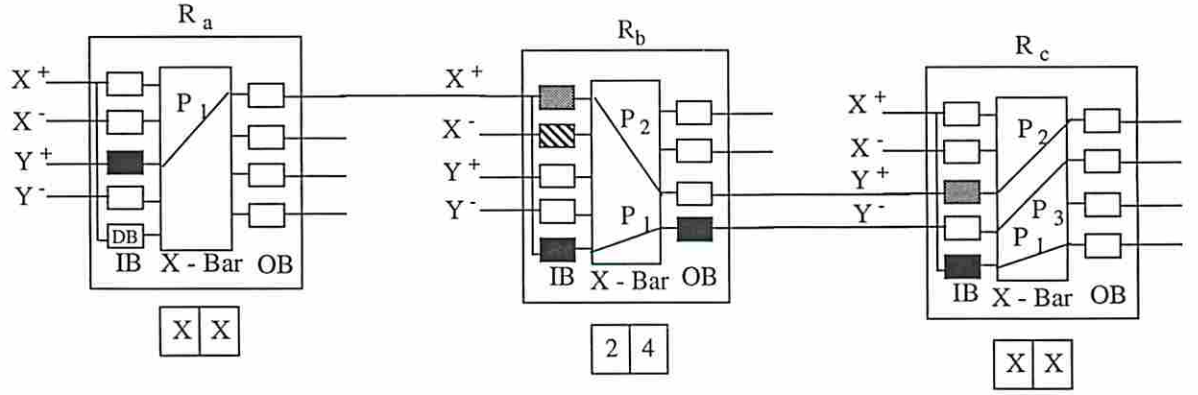

Fig. 7b shows how deadlocks are resolved.

The idea could be extended to support x-y deadlock buffers and dimension-order routing as an escape scheme to allow simultaneous recovery from a number of deadlock situations. This is probably an overkill as potential deadlock situations are so rare. Moreover, *Disha* would loose its simplicity (due to increase in crossbar sizes and additional arbitration logic requirements) and universality. As it now exists, *Disha* is applicable to any interconnection network topology. These supposed improvisations, as simulations show, are found to be unnecessary and may not always provide improvements in performance in accordance with the increase in implementation complexity.

3.5 Deadlock Detection and Handling:

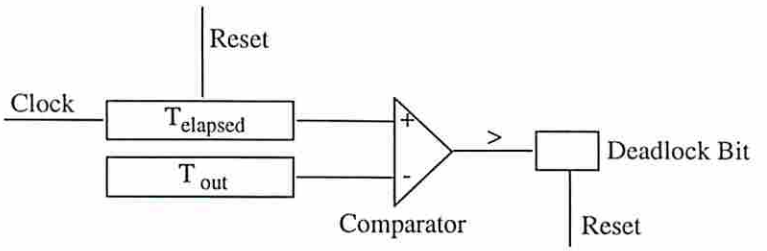

Fig. 8 shows the hardware for deadlock detection.

This section discusses the hardware necessary for deadlock detection and the action that ensues on such a determination. Figure 8 shows deadlock detection circuitry, where $T_{elapsed}$ and T_{out} are counters. $T_{elapsed}$ keeps track of the number of clock cycles for which the router is unable to send out the *header* and T_{out} is loaded with a constant threshold time-out selected to optimize system performance. The Deadlock Bit signals whether or not there is a deadlock. This deadlock detection circuitry is not required by deadlock preventive schemes.

The router waits for the arrival of a packet. On receiving one, it resets $T_{elapsed}$. It then attempts to send out the *header*. If unable to do so, $T_{elapsed}$ is incremented. The router continues to increment $T_{elapsed}$ every cycle until it is successful in sending out the *header* or until the time-out (T_{out}) interval for that packet has been reached. When $T_{elapsed} > T_{out}$, the router changes its state to 'deadlocked' and sets the Deadlock Bit. It then waits for the token to arrive. On receiving the token the router places the packet on the output channel with the corresponding status line asserted. The packet is guaranteed to reach its destination and the router resets the Deadlock Bit once the '*tail*' has gone by. The Token logic resumes Token propagation.

As an optimization, the router could be provided with a counter to keep track of the number of flits of this packet it sends out. This enables the router to determine exactly when the '*header*' has reached its destination. For instance, the '*header*' would take the minimum path to reach its destination in exactly (d-p) hops = (d-p)*buffer_depth cycles, where 'd' is the destination and 'p' the present position of the '*header*'. The router could then reset the Deadlock Bit instead of having to wait for the '*tail*' to go by. The '*header*' having reached the destination is sufficient guarantee that there would be no cycles on the Deadlock Buffer path as it would be sunk there. This allows the Token to propagate even earlier, resulting in even faster recovery of other deadlocks.

It should be noted that a once 'deadlocked' router might, at a later time, resume normal packet transmission without ever receiving the Token. This could come about by some other router in the dependency cycle seizing the Token and initiating deadlock recovery to break the cycle. In this case, $T_{elapsed}$ is reset and detection for deadlock starts all over again.

The selection of a proper time-out interval is important to obtaining optimum performance. Deadlock feeds upon itself in that if cycles are not broken quickly, more and more routers can get engulfed in cycles, and the entire network could come to a standstill. Since recovery from a number of cycles is essentially sequential, a large T_{out} can limit peak obtainable performance. On the other hand, small T_{out}'s trigger false deadlock detections. The choice of T_{out} thus becomes critical. Simulations in Section 4 show how performance varies with different time-out's.

Disha handles livelocks in a similar fashion to what is found in most other schemes, i.e., by placing an upper bound on the number of misroutes allowed.

4.0 Performance:

This section of the paper evaluates the performance of *Disha* using a modified version of FLITSIM 2.0, developed by Sudhakar Yalamanchili et al. at Georgia Institute of Technology. All simulations are run on a 16 by 16 two dimensional torus. Messages are 32 flits long. A buffer depth of 2 was selected to keep the routers simple and reduce clock cycle time.

This paper started out with the assumption that deadlocks are rare. Figure 9 verifies this assumption for two widely varying thresholds. The figure shows the number of packets that seized the Token, normalized with respect to the number of packets that were delivered by the network for time-outs of 4 and 64. The simulations were done with three virtual channels under uniform traffic. The number of deadlocks will be closely related to the number of packets that seize the Token. We can thus safely infer that deadlocks are extremely rare at least until the load at which the network saturates (*Disha* with three virtual channels saturates at 0.40 - Figure 13). This confirms earlier measurements by Kim, et al. [12], which showed that deadlock situations are rare.

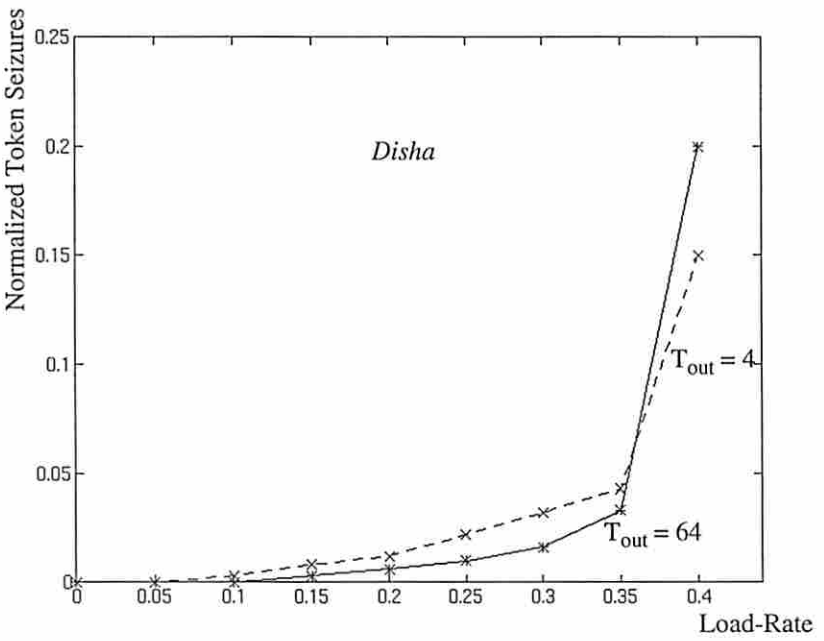

Fig. 9: Normalized number of packets that seized the Token.

A proper choice of the time-out interval is crucial to *Disha*'s performance. A value of T_{out} that produces the least number of Token captures is not necessarily the optimum value from a performance point of view. The performance of *Disha* under a wide range of time-outs is presented in Figure 10. These results again simulate *Disha* with 3 virtual channels. The number 3 was selected because we felt that it was good representation of the number of virtual lanes we could expect in future routers and also because it allowed us to simulate the performance over a wide load range before the network saturated. The message length was kept constant at 32 flits. We found that small time-outs triggered false deadlock detections, while large ones unduly delayed deadlock detection. A time-out of 8 to 16 was found to be appropriate. The default time-out for the remainder of our simulations is 8 clock cycles.

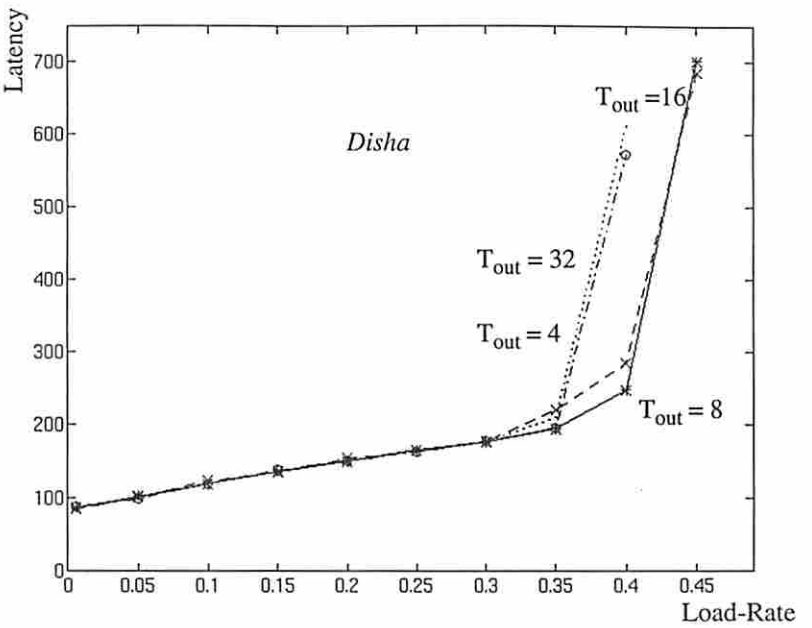

Fig. 10 Performance under various time-out's.

4.1 Disha Network Performance:

Disha's recovery scheme can provide fully adaptive routing even with one virtual channel. *Disha* (with one virtual channel) was modelled based on the approach suggested by Chien [3] to compare it with a dimension-order router which minimally requires two virtual lanes (for a torus).

4.1.1 Cost Model:

Dimension-order routing minimally requires 2 virtual channels for deadlock prevention for a two dimensional torus. *Disha* on the other hand does not require a VCC as multiplexing of the output is not done on a flit by flit basis. As explained in the previous section, if there exists some data on the Deadlock Buffer, the crossbar is reconfigured and the status line is raised to indicate that data is to be placed on the Deadlock Buffer at the next node.

The crossbar in *Disha* has 6 inputs and 5 outputs, including the attached Processor Node - Figure 5. The individual latencies can be calculated based on the constants and equations in [3] and [4]. Here we just present the final results, assuming the following parameters:

T_{ad} : Address Decoding Latency
T_{arb} : Arbitration Delay
T_{cb} : Delay through the Crossbar
T_{hsel} : Header Updation Latency (for adaptive routers)
T_{vc} : Virtual Channel Controller Delay
T_{fc} : Flow Controller Dealy
T_{dor} : Latency through the dimension-order router
T_{disha} : Latency through *Disha.*

Set-Up:

$$T_{disha} = T_{ad} + T_{arb} + T_{cb} + T_{hsel} + (T_{vc} = 0) = 6.5$$

A 20% increase in the complexity of the arbitration logic is assumed to account for the reconfiguration required once the deadlocked packet has passed through.

Dimension-order routing under it's best implementation requires a crossbar with only 6 inputs.

$$T_{dor} = T_{ad} + T_{arb} + T_{cb} + (T_{vc} = 0) = 6.0$$

Disha is thus comparable with dimension-order routing at set-up.

Data Through:

$$T_{disha} = T_{fc} + T_{cb} = 3.1$$
$$T_{dor} = T_{fc} + T_{cb} + T_{vc} = 4.5$$

This suggests that *Disha* is 30% faster than dimension-order routing for data flow-through. Since set-up is only required at the *header* flit, we assume that in the average case, the clock cycle in *Disha* is 25% faster than that in dimension-order routing. Nevertheless, for our simulations in Section 4.1.3 onwards, we assume equal cycle times because we assume the same resource costs for all the schemes being compared.

4.1.2 Uniform Traffic:

Figure 11 shows that *Disha* with 1 virtual channel provides more than a 25% improvement in performance over dimension-order routing at low loads, that is, for Load-Rate < 0.1. The improvement in performance comes mainly due to the reduced clock cycle time. The load is so low that a single virtual lane is sufficient to meet the network traffic needs. However, the saturation point for *Disha* is around 0.1 while dimension-order saturates at a higher load of 0.14. The network saturates because of wormhole blockage. If traffic demands are going to cause the network load to be any greater than 0.1, we definitely need to adopt virtual channels as a method towards increasing flow control.

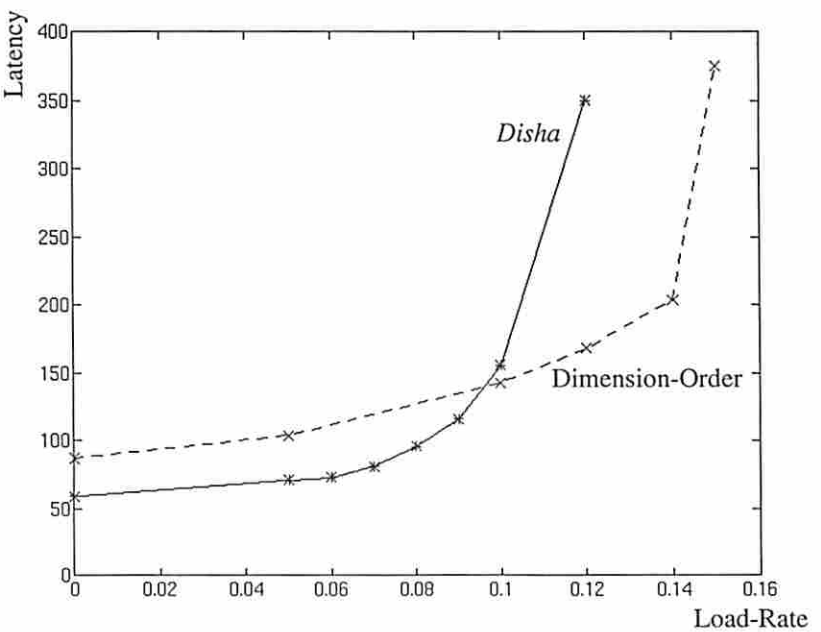

Fig. 11: Performance comparison of *Disha* with 1 VC with dimension-order.

Figure 12 compares *Disha* with 2 virtual lanes against dimension-order routing with an equal number of virtual lanes. *Disha*, with it's ability to provide fully adaptive routing over both lanes, prevails. As can be seen, dimension-order routing saturates at about 0.14 load, while the saturating point for *Disha* is around 0.20. Here both schemes are assumed to have the same clock cycle time, although T_{disha} would be slightly higher for link set-up.

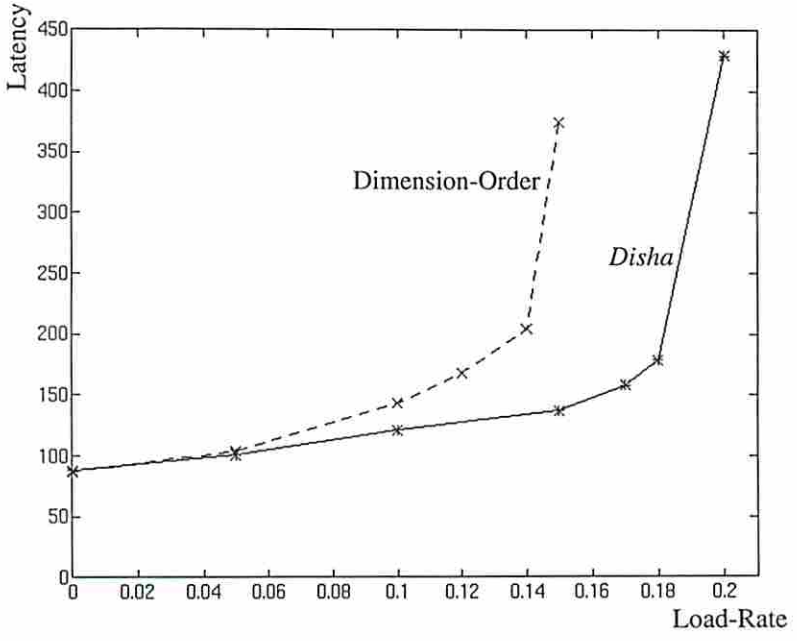

Fig. 12: Comparison of *Disha* with 2 VC's with dimension-order.

We next compare *Disha* with an adaptive scheme like Duato's in Figure 13. This algorithm is based on the results derived in [9]. Duato's algorithm for a torus would require at least 3 virtual lanes (2 for deterministic routing) and, to make the comparison fair, equal resources were given to *Disha*. The adaptive algorithm in both the schemes was the same. Here, too, *Disha* is a winner, saturating at 0.40 which was twice that of Duato's.

If a comparison is drawn between Figures 12 and 13, it can be observed that the performance of *Disha* with 2 virtual lanes is comparable to Duato's scheme. This is not taking into account the fact that the complexity of a 2 virtual lane *Disha* would be much less than that of Duato's with 3 lanes (consequently leading to a faster clock). This observation implies that even if a virtual lane is devoted exclusively to implement the token, *Disha* would still outperform existing adaptive schemes. The relaxed set of conditions derived by Duato in [10] may provide better results, but we do not expect the performance of such a scheme to approach *Disha*'s.

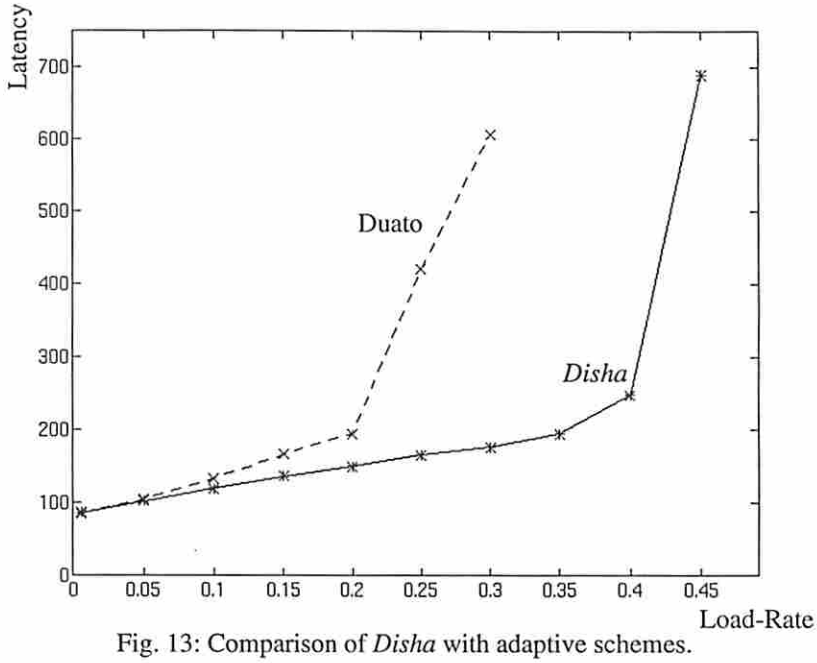

Fig. 13: Comparison of *Disha* with adaptive schemes.

4.1.3 Non-Uniform Traffic:

Thus far we have compared *Disha* with other schemes under uniform traffic. To complete the set of results we do an analysis wherein we compare dimension-order routing with *Disha* for hot spot traffic patterns in Figure 14.

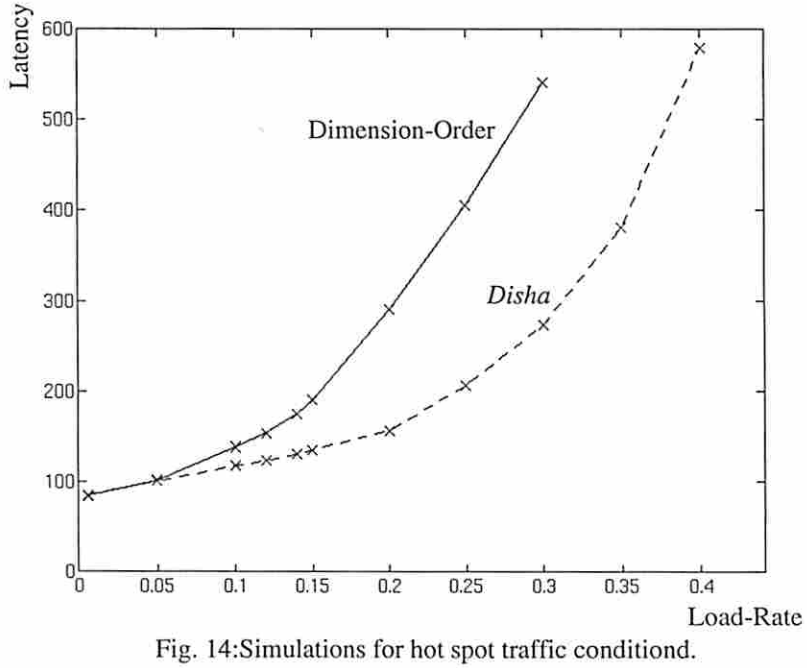

Fig. 14:Simulations for hot spot traffic conditiond.

In these simulations, we assumed that up to 5% of the network traffic is hot spot in nature, and the number of locations at which these patterns exist to be 4. Although in these simulations, hot spot conditions are a small fraction of the total traffic, it still indicates that the performance of a deadlock recovery scheme like *Disha* is not likely to be affected by non-uniformities in traffic. These simulations were run with the same time-out value as for uniform loads. We expect the performance to improve with fine tuning in the form of increasing time-outs to filter out false detections.

Some simulations were also done by varying the network load. The load was raised to a point above saturation for some time and then reduced (useful to simulate locality). Even under such cases *Disha* performed well, indicating that recovery from the increased number of deadlocks after saturation is quick.

5.0 Conclusion:

Disha provides the means for safely incorporating fully adaptive wormhole routing to support efficient and fault-tolerant communications in a parallel processing environment. The scheme is universal in that it can be applied to any arbitrary network topology. It employs deadlock recovery as opposed to prevention with the objective of making the common case faster. Consequently, it does not require virtual channels for deadlock freedom. Hardware devoted to recovery is minimal, enabling the routers to be fast.

Simulations show that the number of potential deadlock situations is very small until the point at which the network saturates. If enough virtual channels are provided to match the traffic flow demands of the network, then *Disha* should provide excellent performance. This is due to the fact that all virtual channels are used for adaptive routing, not deadlock prevention, and the routing complexity is minimal. *Disha* is also able to perform well under bursty and hot spot traffic loads, and provides maximum fault tolerance capability.

There are many areas of future research resulting from this work. Without major changes to the basic scheme, it might be possible to allow multiple tokens to circulate, thus allowing recovery from a number of simultaneous deadlock occurrences. This and other such hardware optimizations are possible.

Reducing the probability of deadlocks is another related area of interest. It has been observed that the number of deadlocks depends on topology. Also, some routing algorithms are more sensitive to deadlocks than others.

A complete study on the characterization of deadlocks is necessary to extract maximum performance from a recovery scheme, such as *Disha.* As has been shown in this paper, the performance of *Disha* depends on the selection of a suitable time-out interval. The optimum value of time-out will not be the same for different message lengths, buffer depths, etc. The same can be said when we go from uniform to non-uniformities in traffic. Such a study will help us fine tune the system to obtain peak performance.

Acknowledgments:

The authors are extremely grateful to Sudhakar Yalamanchili and Binh Vien Dao at Georgia Institute of Technology for providing and helping us set-up Flitsim 2.0, which was used in all the simulations presented in this paper. Enlightening discussions with Jose Duato also proved to be very useful. We thank these individuals.

References:

[1] J. D. Allen et al. Ariadne*--- An Adaptive Router for Fault-tolerant Multicomputers. In *Proceedings of the 21st Annual International Symposium on Computer Architecture, IEEE Computer Society*, pages 278-288, April, 1994.

[2] Andrew 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, IEEE* Computer Society, pages 268-77, May, 1992.

[3] Andrew A. Chien. A cost and performance model for k-ary n-cube wormhole routers. In *Proceedings of Hot Interconnects Workshop*, August 1993.

[4] K. Aoyama. Design Issues in Implementing an Adaptive Router. Master's thesis, University of Illinois, Department of Computer Science, 1304 W. Springfield Avenue, Urbana, Illinois., January 1993.

[5] R. V. Boppana and S. Chalasani. A Comparison of Adaptive Wormhole Routing Algorithms. In *Proceedings of 20th International Symposium on Computer Architecture, IEEE* Computer Society, pages 351-360, 1993.

[6] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5), pages 547-553, May, 1987.

[7] W. Dally. Virtual channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194-205, March, 1992.

[8] W. Dally and H. Aoki. Deadlock-free Adaptive Routing in Multicomputer Networks using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems,* Vol. 4, No. 4, pages 466-475, April, 1993.

[9] J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies. In *Proceedings of Parallel Architectures and Languages Europe,* pages 390-405, June, 1991.

[10] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. In *Proceedings of the International Conference on Parallel Processing*, pages I142-I149, August, 1994.

[11] K. D. Gunther. Prevention of Deadlocks in Packet-Switched data Transport Systems. IEEE *Transactions on Communications*, Vol. Com-29, No. 4, pages 512-524, April 1981.

[12] J. Kim, Z. Liu and A. Chien. Compressionless routing: A framework for adaptive and fault-

tolerant routing. In *Proceedings of the 21st International Symposium on Computer Architecture, IEEE* Computer Society, pages 289-300, April, 1994.

[13] D. Linder and J. Harden. An Adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers,* C-40(1):2-12, January 1991.

[14] Z. Liu and A. Chien. Hierarchial Adaptive Routing: A Framework for Fully Adaptive and Deadlock-Free Wormhole Routing. To appear in the *Symposium on Parallel and Distributed Processing*, 1994.

[15] L. Ni and C. Glass. The Turn Model for Adaptive Routing. In *Proceedings of the 19th International Symposium on Computer Architecture, IEEE* Computer Society, 20(2):278-287, May, 1992.

[16] L. Ni and P.K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. IEEE *Computer Magazine*, February 1993.

[17] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks 3 - North Holland Publishing Company, pages 267-286,* 1979.