True fully adaptive routing employing deadlock detection and congestion control.

16 May, 2001

Dimitris Papadopoulos, Arjun Singh,
Kiran Goyal, Mohamed Kilani.
{fdimitri, arjuns, kgoyal, makilani}@stanford.edu

1. Introduction

Networks using wormhole routing have traditionally relied upon deadlock avoidance strategies for the design of routing algorithms. More recently, deadlock detection and recovery strategies have begun to gain acceptance. If deadlocks are detected efficiently, deadlock recovery schemes can be very attractive because they can quickly preempt some held resources and deliver deadlocked messages to their destination. Without the limitations that deadlock avoidance imposes on virtual channels (VCs), *true fully adaptive routing* ***(TFAR)*** can be effectively used, avoiding congested channels and resolving deadlocks by choosing other virtual channels. As long as the network is kept below the saturation level, TFAR works very efficiently – since the chances of a deadlock at low congestion is small. However, near saturation, deadlock detection mechanisms start detecting false deadlocks and the recovery operations act to relieve congestion rather than deadlock. Faulty messages detected as deadlocked may saturate the bandwidth offered by recovery resources, thus degrading performance considerably. Hence, the problem of avoiding saturation assumes importance. Many congestion control mechanisms have been proposed which limit the injection of messages at the current node (message throttling) under saturation conditions. However, most message throttling schemes are local in nature, i.e. they ignore the global state of the network.

In this paper we will examine the feasibility of TFAR with the aid of deadlock detection/recovery and congestion control. The plan of the paper is as follows: in section 2, we describe how TFAR lends tremendous flexibility if deadlock detection/recovery is done efficiently. In section 3, we examine the pros and cons of deadlock detection/recovery over the traditional avoidance schemes. However, for TFAR to perform well, saturation must be avoided. In section 4, we study some of the proposed congestion control mechanisms that avoid saturating the network. Sections 3 and 4 are based on the study of four papers, namely [1], [2], [3] and [4]. Section 5 summarizes the problem and states our conclusion. Finally, section 6, gives a brief glimpse of the aspects of this problem not tackled yet.

2. True Fully Adaptive Routing (TFAR).

TFAR allows true fully adaptive routing of packets in which unrestricted routing on all virtual channels over all router ports is permissible. Since, no constraints are imposed on the usage of virtual channels, the critical issue of

deadlocks must be addressed. Deadlock avoidance has traditionally dominated the design of interconnection networks. However, such designs result in routing schemes that restrict routing options. In [1] it is suggested that TFAR be used together with a deadlock detection and recovery scheme. The argument is that deadlocks occur rarely (especially under low congestion) and so it is wasteful to restrict the routing options in the common case of no deadlocks.

In the rare case of deadlock, it is detected and efficiently resolved by progressively routing one of the blocked packets through a connected, deadlock-free recovery path. This path need not be implemented by devoting physical or virtual channel resources to this purpose. Instead, these resources can be used for the sole purpose of increasing flow capabilities of the network, not for guarding against infrequent deadlocks. Deadlock handling resources are decoupled from normal channel resources, allowing adaptive routing to occur using any number of virtual channels [1]. In order to consider the strong points of TFAR we examine how deadlock avoidance routing affects performance by preventing packets to route around congestion. Then we examine the behavior of TFAR as load increases and the network moves toward saturation.

2.1 Deadlock Avoidance Routing

Deadlock avoidance schemes are either partially or fully adaptive. Fully adaptive routing **(FAR)** implies full adaptivity across all physical links, but only *partial* adaptivity across the virtual channels. That is, there is at least a set of VCs that connects to all other nodes but the rest of the VC sets are restricted to specific dimensions. In contrast TFAR, can use all VCs over all ports.

Partially adaptive avoidance schemes like deterministic routing and turn model limit adaptivity considerably. The turn model [11], which requires one VC for meshes and two for Tori, restricts adaptivity. For example the West-First algorithm does not provide any adaptivity as long as the destination node is still on the west. Similarly, in the North-last scheme as long as the destination node is at the north, the packets have no option for adaptivity when they start going north. If they encounter congestion they must block, resulting in poor performance and can be outperformed even by non-adaptive algorithms.

Although Duato's fully adaptive deadlock avoidance algorithm [5] provides better flexibility in routing with a minimal number of virtual channels (VCs), it is not "truly fully adaptive" i.e. it must restrict fully adaptive routing to only one VC. VCs are divided into two classes, one fully adaptive with minimal routing and the other deterministic, free of cycle dependencies. If three VC's are used, only one of them can be used for fully adaptive routing. Thus, adaptivity is limited to only a fraction of VC's.

As previously mentioned, TFAR imposes no constraints on the set of VCs to be used. Together with deadlock detection/recovery, it works very well as long as the network does not approach saturation. At high load, however, TFAR starts to choke.

2.2 True Fully Adaptive Routing at saturation.

TFAR (using deadlock detection/recovery) extends the throughput of the network beyond the point fully adaptive algorithms do because it can easily avoid blocked VCs (under low load). Since deadlocks are not often (in a lightly loaded network), a deadlock recovery mechanism can work efficiently and free the resources

before a tree of congested packets is formed. But the network should be able to deliver messages effectively, even under saturation. The question remains, if the network enters saturation which of the two deadlock polices (avoidance or detection/recovery) works more effectively. Saturation cannot be avoided and the behavior of the network should not degrade considerably. With FAR, it is sure that all blocked packets are simply congested and not deadlocked, thus, as the offered load is increased, the throughput (accepted traffic) is sustained at a high value with an increasing latency. However, in TFAR, at the saturation point, congested messages are misread as deadlocked. Thus, real deadlocks take more time to be recovered, additional routes get engulfed in the deadlock dependency and eventually the entire network could come to a standstill. This can be seen from the graphs of [1] where throughput has the tendency to tail off as the network is unable to continue sustaining traffic at the rate at which traffic is applied.

It is clear that the effectiveness of TFAR is dependent upon the deadlock detection, recovery and congestion control used. We start by examining deadlock recovery.

3 Deadlock recovery and detection.

3.1 Deadlock recovery (paper [1])

The progressive deadlock recovery mechanism proposed in [1] allocates resources on a connected deadlock free path through which an eligible packet is progressively routed to its destination. This progressive recovery scheme called "Disha" has two implementations [6]. It is an attractive solution in comparison to regressive recovery schemes (which kill and re-inject the packets) and deflection schemes (which misroute packets out of deadlock).

The Disha Sequential scheme requires that a recovering deadlocked packet obtain exclusive access to the set of *deadlock buffers* before using it and route minimally on the recovery path. This mutual exclusion on the deadlock buffers can be enforced by a circulating token, which visits all routers in a predetermined cyclic path. Once a router detects a potential deadlock situation, it becomes eligible to capture the circulating token. Fair access to the recovery path is obtained by capturing the token and arbitrating fairly among the one or more deadlocked packets at a router. The router progressively routes the deadlocked packet over a preempted output physical channel while having the corresponding control line set to indicate that this packet should be directed to the deadlock buffer at the next router in sequence. The destination node releases a new token once the packet header is received. As there exists at least one packet in each deadlock that can fairly gain access to and route progressively on the recovery path, which is connected and deadlock-free, this routing scheme safely recovers from all deadlocks.

An alternative implementation of Disha, referred to as Disha Concurrent, routes on a recovery path such that some total resource ordering is enforced that does not result in the formation of cyclic dependencies. One way of enforcing such structured routing is to use short-cut Hamiltonian path ordering on the connected recovery path.

Strong points of [1]

- It provisions for true fully adaptive routing.

- No virtual channels are required for handling deadlocks – making it cost-effective.
- The scheme is independent of network topology and switching techniques.
- There is very little overhead of flit buffering that is faced by abort-and-retry schemes of recovery.
- The deadlock recovery is predictable and efficient and is based on *progressive* recovery.

Drawbacks of [1]

- The performance of the recovery scheme is strongly dependent on the low frequency at which deadlocks occur.
- Excess hardware is required for the implementation of the token logic described in this scheme.
- Its performance degrades if the detection mechanism is slow. Deadlocks should be recovered faster than they are formed and detected, otherwise the resources allocated for recovery will saturate and that will limit performance.

The detection should distinguish between blocked and deadlocked messages quickly and efficiently. It should also detect only one of the virtual channels forming a deadlock. Freeing only one of the many channels involved in the deadlock is sufficient to break the deadlock. If all the channels that are deadlocked are freed, the recovery resources are unnecessarily loaded. We will examine the deadlock detection mechanism next.

3.2 Deadlock Detection (paper [2])

Deadlock detection has been traditionally based on measuring the time that channels requested by blocked messages are inactive for a given time threshold. Transmission activity is monitored on every physical link because a message is presumed to be deadlocked only if all the alternative virtual channels requested by the message contain blocked messages. That definitely assumes a truly fully adaptive routing. Obviously this method is unable to precisely distinguish between true deadlocked and blocked messages waiting for a channel to be released. Furthermore, it will mark as deadlocked *all* the packets involved in the deadlock.

In [2] the deadlock detection combines the above threshold comparison together with the following mechanism. Suppose a packet arbitrates for virtual channels but is stalled because all are occupied. Occupied virtual channels can either be blocked (no flit movement on the wire) or have flits advancing. If the VCs requested by a newly arrived packets are all busy but at least one not blocked and later all of them block, it should detect deadlock when the threshold is exceeded. However, if an arriving packet stalls for a VC and all the VCs are already blocked, that situation should never lead to a deadlock even if the timer exceeds the threshold.
To demonstrate the above scheme we show an example:

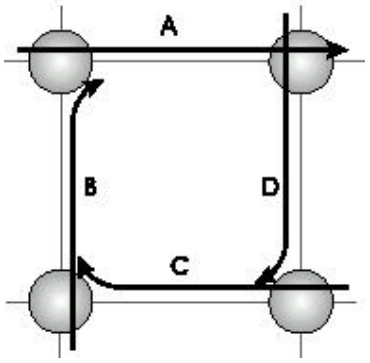

Fig. 1. No deadlock detected since A is moving

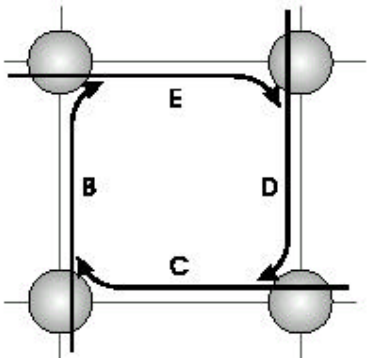

Fig. 2. Only B is detected as deadlocked

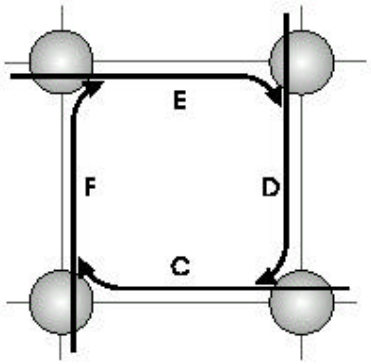

Fig. 3. Only C is detected as deadlocked

Consider only one VC is available and packet B is blocked by A, then C arrives and is blocked by B (and similarly for D as in Fig.1). As expected, no deadlock will be identified because all channels were already blocked when the packet arrived. Thus, even if they are blocked for a long time, as long as A does not block, they will not be marked as deadlocked. Suppose that message A leaves the channel it occupied, and that a newly arrived message E occupies that channel (see Fig. 2). Consider also that E requests a channel occupied by D and a deadlock is formed. Packets C and D are not eligible to be detected as deadlocked but B is, because when it had arrived at this node, A was advancing and now E gets blocked. Thus, when it will exceed the given threshold, a deadlock will be detected and the recovery mechanism will be triggered *only* in the node containing the header of B. As a consequence, the deadlock is removed. If another packet F acquires the channel that message B has just freed and requests a channel occupied by packet E (see Fig. 3), another deadlock is reached. Packet F cannot trigger deadlock recovery because E is already blocked. The same applies to E and D. However, packet C detects the transmission of F which later blocks and C will now detect the deadlock and trigger the recovery mechanism.

This scheme will always identify the deadlock formation irrespective of the arrival patterns, as there will always be at least one packet identified as already moving by another blocked one. It may also happen that several messages involved in a deadlock block simultaneously. In this case, deadlock is detected for several packets, because each is blocked by another packet that is still advancing. Though recovery is invoked, the overhead is higher than in the other cases. This is the worst case and is expected to be infrequent because it requires several packets arriving simultaneously.

Implementation:

Deadlocks can only occur when every VC in the considered output link is busy. Therefore, it is sufficient that the activity be monitored in the physical link and not on each virtual channel separately. The counter that sets the deadlock detection (DT) flag should be enabled only when all the VCs over a physical link are blocked and inactive for some period.

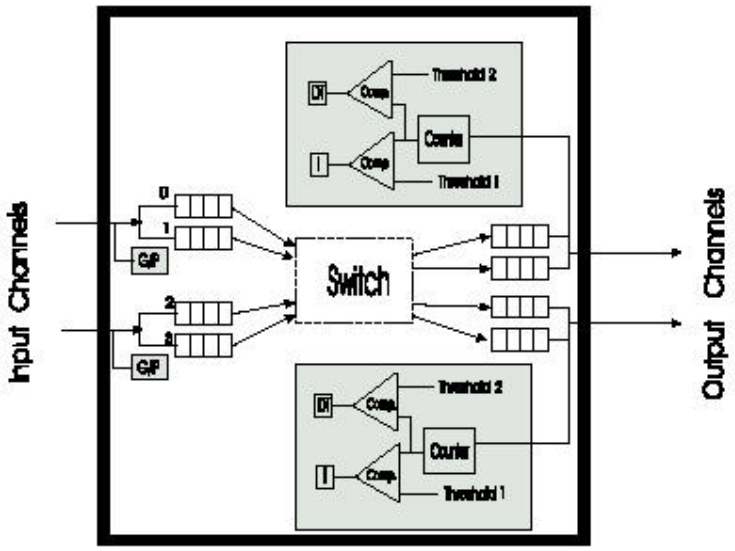

Fig. 4 Implementation of the efficient detection scheme.

Referring to Fig. 4, the I flag is set if there is no activity in the link for a cycle. Note that the counter is only incremented if at least one of the allocated input VC is occupied by a packet. An incoming flit header that is blocked, for first time in the current node, will examine the I flags of every output link the routing function has returned. If one of them is reset (packets are advancing along a virtual channel) the G/P flag in the status register of the input buffers is set to G. Otherwise, if all I flags are set (no activity over any VC over all links), the G/P flag is set to P. For every successive, unsuccessful attempt, all the DT flags and the G/P flag associated with the current status register are checked. When all the DT flags are set and the flag G/P=G, the threshold has been exceeded and the current packet is supposed to be involved in a deadlock. If one or more DT flags is not set, means that there is at least one active virtual channel and thus no deadlock can exist yet. The packet should wait until an output channel is freed.

Drawbacks of [2]:

- When the network is congested, this scheme detects false deadlocks. All virtual channels of all the links the routing function returns may be busy and then blocked. That happens if there is a deadlock somewhere ahead and packets which do not form the deadlock, are blocked waiting for the deadlock to be resolved. Packets block and trigger the recovery mechanism faster than the escape paths can drain them. Thus, performance degradation appears. Self tuning and increasing the DT flag threshold when the network experiences saturation would help so that less false deadlocks are identified. However, the

real deadlocks would be resolved late and congestion around them would be even worse.

Strong points of [2]:

- Below saturation, this scheme can identify all the true deadlocks effectively and, in the best case, trigger the recovery of only one of the packets forming the deadlock. This way, we get less deadlock detections per deadlock and the recovery mechanism can provide the necessary bandwidth efficiently.

In their simulations, a message injection limitation mechanism is used in order to avoid the performance degradation of the network when it reaches saturation. In [3] and [4], more efficient congestion control mechanisms are proposed and simulated together with the above deadlock detection and recovery mechanism. We will examine this next.

4 Congestion control schemes

We study schemes where congestion control is approached from both a local perspective (as in [3]) as well as from a global one (as in [4]).

4.1 Local Congestion estimation (paper [3])

An approach to avoid the network traffic reaching saturation point is proposed in [3] (called U channels). It estimates the network traffic locally and when network load is considered to be high, message throttling is used in each node to keep the network out of saturation. Message injection is completely stopped until traffic goes down again. Before a newly generated message is injected into the network, the routing function is applied to obtain the total number of "useful" channels that the message may use to reach its destination, U. Then the number of free VCs (F) out of the "useful" VCs is calculated. Finally, the quotient between the free useful output VCs and the total useful output VCs, Q=F/U, is computed. If Q is greater than a threshold, the message can be injected. Otherwise the message is queued. Injection of messages is allowed when Q exceeds the threshold again.

Strong points of [3]:

- If F alone were used to decide whether or not to inject, then the threshold value would have been different for distinct message-destination distributions. However, [3] uses the fraction F/U to decide whether or not the message can be injected. This makes it possible to fix the threshold value and use it for all message destination distributions.

- No more signals or buffers are required for sending extra information and although the hardware may add some delay for newly generated messages, it does not reduce the clock frequency, as it is not on the critical path for routing incoming messages.

- Latency versus throughput for the above scheme is compared against a different mechanism, DRIL [10] and another one which does not use any congestion control (No-lim). DRIL is a congestion control mechanism previously proposed by the same authors that computes a dynamic threshold depending on the network load, for each node. This way a different degree of

injection limitation was applied determined by the load. As seen in the graphs of Fig. 6 and 7, the performance of U- channels is better than the other mechanisms.

- Fairness in [3] is considered for the case that the injection rate is the same for all nodes in the network. Fig 5. Shows the percentage of messages sent by each node. With a uniform distribution of message destinations, the differences in sent messages per node are about 15% in worst cases. The DRIL mechanism, however, gave a difference in sent messages per node that reached 60%.

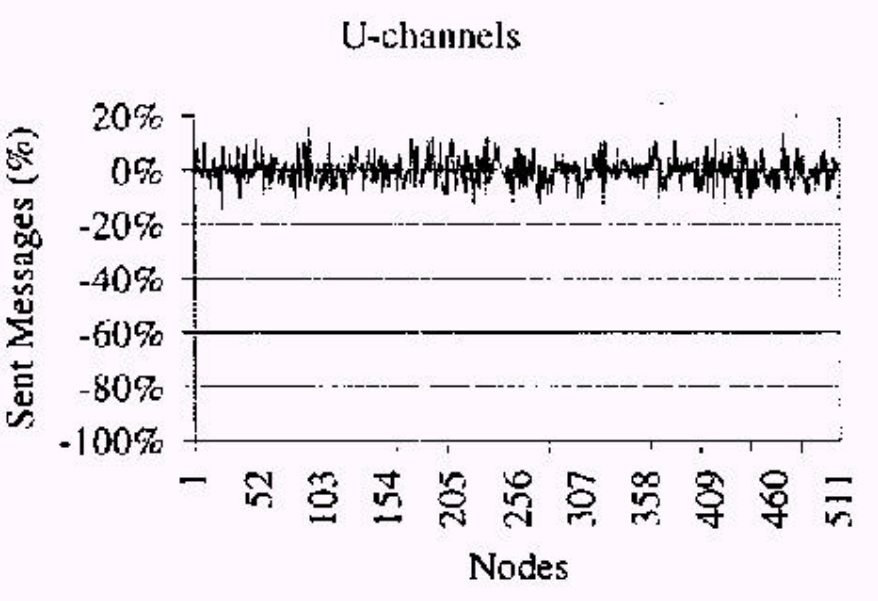

Fig. 5 Differences in sent messages per node for uniform traffic.

- The graphs in Fig 6 show that throughput increases with increasing latency as the offered load is increased. However this latency corresponds to an average over all messages over all nodes and thus cannot provide information about the delay of newly generated messages blocked in the source node (latency as estimated includes this initial blocking time as well). Therefore, a low standard deviation of latency (estimated as the standard deviation of the average delay of all the packets injected from each node, between all nodes) versus throughput (accepted traffic) gives a good indication of fairness. In [3], this simulation result is provided for the uniform and butterfly pattern.

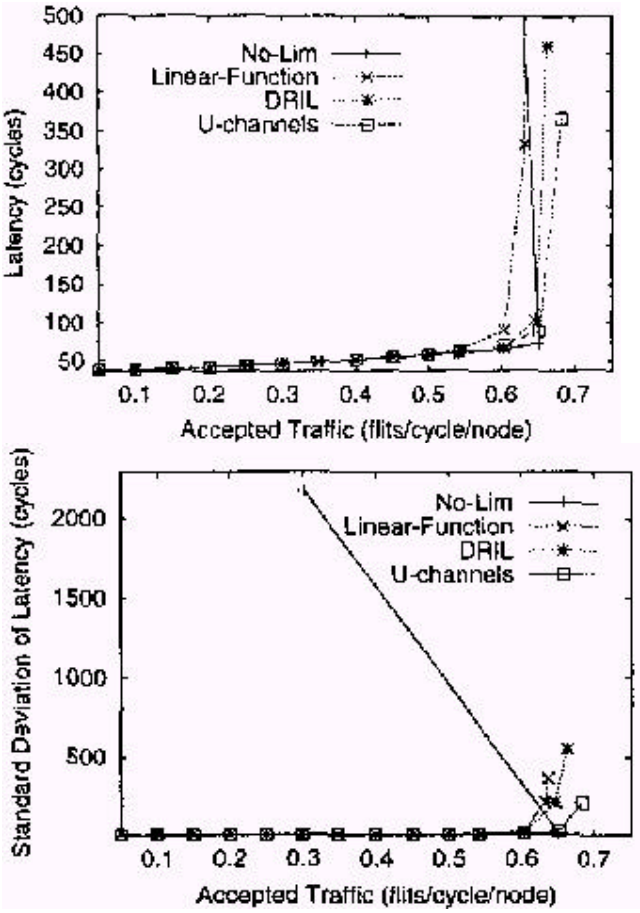

Fig 6 Uniform traffic- 16 flit messages.

The graphs show that the standard deviation is very low at low loads. Even at the max load it accepts (0.65), the standard deviation is significantly lower than the other schemes. However, it does get quite large at a load of 0.7, which is expected as the network is congested and there is unfairness between the packets sent from different nodes.

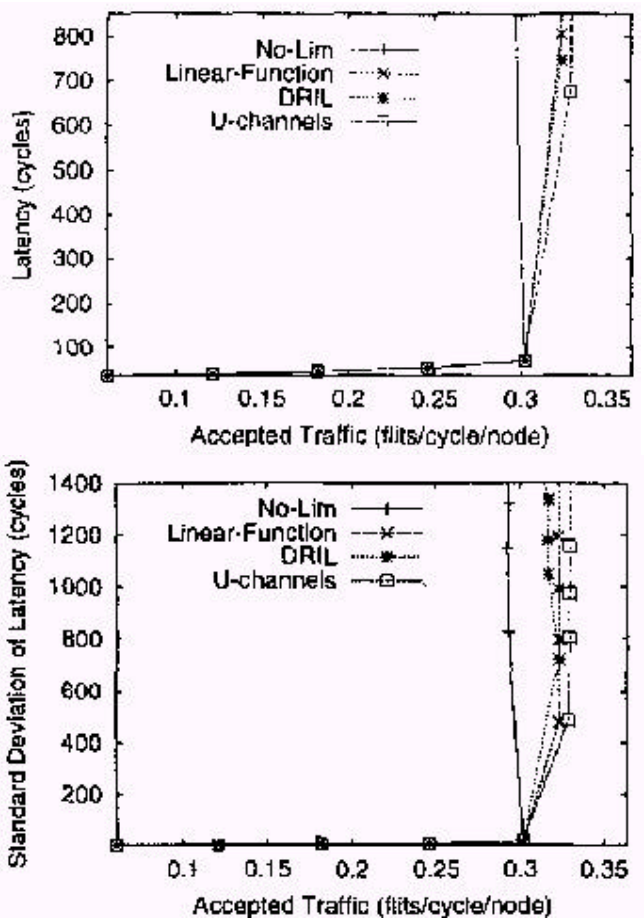

Fig. 7 Butterfly traffic- 16 flit messages.

Drawbacks of [3]:

- This scheme tries to keep the network below saturation by local injection control. However, this does not imply that it really manages to keep virtual channels less congested. This is because other uncongested nodes will route traffic through them. A point will be reached where all useful VCs of a node are used and priority will be given to incoming messages generated by nodes, which do not suffer from congestion. To alleviate this situation, some of the nodes that are out of congestion must stop generating messages. Then the congestion will start decreasing and blocked new messages will be able to go ahead. Otherwise, as long as non-congested nodes go on sending, taking advantage of their priority over congested ones, the situation is irreversible. Only if congestion spreads out fast enough, can the situation be reversible. That means that despite the local throttling, congestion will still expand across more nodes, until they all have to throttle. From this point the alleviation from congestion is straightforward, as all nodes start doing self-throttling to their messages.

- Even though, the authors claim that U-channels are more fair compared to DRIL, they only simulate for uniform traffic (as in Fig. 5). However, this is the best-case scenario for this scheme since all nodes will

face the same injection limitations. No simulation result has been provided for non-uniform message destination distribution for the difference in sent messages.

Behaviour of the deadlock detection mechanism with congestion control.

After proposing the above congestion control mechanism, they simulate it with the deadlock detection scheme of [2]. The recovery mechanism is the software one proposed in [9]. The choice of the threshold value for the congestion control, in [3], was based on the trade-off between low latency penalty and low number of detected deadlocks.

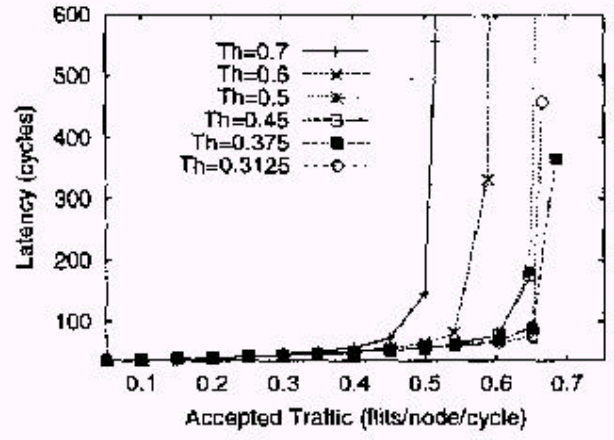

Fig . 8 Uniform traffic –16 flit packets.

Referring to Fig. 8. for big threshold values, latency gets bad because the limitation is severe, but the number of detected deadlocks is low. However, for smaller threshold values (up to 0.375), latency becomes better while the number of detected deadlocks increases. At a threshold below 0.375 latency vs. throughput starts getting worse again and this is the point where the congestion control starts to take effect. This threshold value, 0.375, was chosen for all the simulations. For all simulations the percentage of detected deadlocks was estimated and they were considerably reduced because of the message injection limitation used. Otherwise the number of detected deadlocks is very high (greater than 70% for bit-complement, 35% for perfect-shuffle and 20% for bit-reversal patterns).

At this point it is clear that there is a great dependence of detected deadlocks to the network saturation. While real deadlocks are increased because of a shortage of free VCs, a lot more false deadlocks are created by the delay of simply blocked packets that timeout and set DT. These two factors, in combination with the limited recourses of the deadlock recovery mechanism, lead to real deadlocks being resolved at a rate slower than they are generated. Thus, network performance deteriorates rapidly. Therefore, emphasis should be given to a good congestion control mechanism that is fair and controls congestion before it spreads everywhere. This could be achieved by analyzing the network's global state and acting proactively before the local node gets congested. This is what we examine next.

4.2 Global Congestion estimation (paper [4])

Most congestion control techniques estimate congestion using a locally observable quantity like local buffer occupancy and packet timeouts. These are less useful because by the time such things are detected the network is already overloaded. In [4] congestion is detected earlier by using a fraction of full buffers of *all* nodes. The global estimate is compared against a threshold to control packet injection. If the estimate is higher than the threshold, packet injection is stopped. When the estimate drops below the threshold, packet injection is resumed.

Further, they have a self-tuning mechanism that automatically determines appropriate threshold values based on throughput feedback. This allows the scheme to adjust to variations in communication patterns.

The authors of [4] suggest three ways to implement this scheme. One way is to piggy pack extra information on normal packets. However, this reduces the effectiveness as not all nodes may get all the information. Another way is to send high priority meta packets. This consumes bandwidth and may add to congestion. Their implementation uses an exclusive side band reserved for communicating congestion and throughput information alone. This is costly in terms of hardware and complexity. Aggregation of information is done dimension-wise.

Consider the relationship between offered load, full buffers and delivered bandwidth. As offered load increases from zero, the number of full virtual buffers and delivered bandwidth also increase. When saturation occurs, the delivered bandwidth decreases while the number of full virtual buffers continues to increase. Their self-tuning attempts to find the number of full virtual buffers that maximizes delivered throughput. They do this by using hill climbing and an algorithm to avoid a local maxima.

Drawbacks of [4]:

- The scheme counts full buffers to estimate congestion but does not take into account the distribution of these full buffers among the nodes. It is not clear how their scheme differentiates between highly localized congestion in which the full buffers are in a very small group of neighboring nodes and the case in which the full buffers are distributed more or less evenly throughout.

- The implementation involves a dedicated side-band which is quite expensive in terms of hardware and also adds complexity. The algorithm has only been simulated and not really built in to a machine. We suspect there will be a lot of issues (addition in complexity) in implementing the side band and the dimension wise gather.

- The traffic used on their simulations is synthetic and not from a real workload. They do not describe how they generated the traffic synthetically.

- A lot of their parameters (like the steps for incrementing and decrementing the threshold) are found by simulating and then tuning it to get best performance. These parameters are functions of the network and have to be done separately for every network. It would be nice if they could comment on how to adjust them and how are they related to the network.

- We think that there is some merit to modifying the frequency of tuning according to the network conditions. For example we would like to tune more frequently if it is suspected that the network has some congestion. However, the paper does not discuss this.

5. Conclusion

True Free Adaptive Routing (TFAR) gives routing complete flexibility over all VCs of all ports. However, with full flexibility comes the problem of deadlock detection and recovery (since avoidance will constrain the use of VCs, thus preventing TFAR). Recovery mechanisms (like Disha) are very effective as long as detection is efficient and the network is lightly loaded. However, at loads nearing saturation, TFAR (along with the detection/recovery) performs poorly. Hence, we looked at controlling congestion to reduce the chance of saturation as far as possible. We studied various schemes (both using local measures as well as global ones) of congestion control analyzing their strengths and weaknesses.

With state of the art schemes of deadlock detection ([2]), recovery ([1]) and congestion control ([3] and [4]) working in tandem, TFAR can be very effective in all scenarios under any traffic pattern.

6. What lies ahead?

The various heuristics proposed for the different schemes of deadlock detection, recovery and congestion control might have significant bearing on the network topology. Till now no concrete study has been done to show whether or not such heuristics are independent of topology. The heuristics may perform differently for various topologies. If that is the case, a designer must employ the best heuristic for the given topology.

Moreover, the ever-important metric of QOS is not addressed by any of the schemes we have proposed. For instance, the congestion control schemes may throttle the injection rate but they also increase the potential delay of the packets they end up queuing at the source node. There is a lot of research possibility in the area of providing both delay and bandwidth guarantees to certain high priority packets.

Main references

[1] T.M.Pinkston, “Flexible and Efficient Routing Based on Progressive Deadlock recovery”, IEEE Transactions on Computers, July 1999.

[2] P.Lopez, J.M.Martinez, J.Duato, “A Very Efficient Distributed Deadlock Detection Mechanism for Wormhole networks”, High perf. Computer arch., 1998 proceedings, IEEE.

[3] E.Baydal, P.Lopez, J.Duato, “A Congestion Control Mechanism for Wormhole networks”, Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on , 2001

[4] Thottethodi, M.; Lebeck, A.R.; Mukherjee, S.S., “Self-tuned congestion control for multiprocessor networks**”,** High-Performance Computer Architecture, 2001. HPCA.The Seventh International Symposium on, 2001 Page(s): 107 -118

Additional references

[5] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks", IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 12, pp. 1,320-1,331, Dec. 1993.

[6] K.V. Anjan and T.M. Pinkston, "DISHA: A Deadlock Recovery Scheme for Fully Adaptive Routing", Proc. Ninth Int'l Parallel Processing Symp., pp. 537-543, Apr. 1995. editor for the IEEE Transactions on Parallel and Distributed Systems.

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

[8] P. L? opez, J.M. Mart?inez, F. Petrini and J. Duato, "On the Reduction of Deadlock Frequency by Limiting Message Injection in Wormhole Networks," *Parallel Computer Routing and Communication Workshop*, June 1997.

[9] J.M. Mart?inez, P. L? opez, J. Duato and T.M. Pinkston, "Software-Based Deadlock Recovery Technique for True Fully Adaptive Routing in Wormhole Networks," *1997 International Conference Parallel Processing*, Aug. 1997.

[10] P Lopez, J. Martinez and J. Duato, "DRIL:Dynamically reduced message injection limitation mechanism for wormhole networks", 1998 Int. Conf. Parallel Processing, Aug '98.

[11] L. Ni and C. Glass, "The Turn Model for adaptive routing", Proc. Symp of Computer Architecture, May 1992.