In-Order Packet Delivery in Interconnection Networks using Adaptive Routing*

J.C. Martínez, J. Flich, A. Robles,
P. López, and J. Duato
Dept. of Computer Engineering
Universidad Politécnica de Valencia
Camino de Vera, 14, 46071, Valencia, Spain
jc@disca.upv.es

Michihiro Koibuchi

Dept. of Information and Computer Science
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama,
223-8522 Japan

Abstract

Most commercial switch-based network technologies for PC clusters use deterministic routing. Alternatively, adaptive routing could be used to improve network performance. In this case, switches decide the path to reach the destination by using local information about the state of the possible outgoing links. However, there are two drawbacks that discourage adaptive routing from being applied to commercial interconnects. The first one concerns the possible switch complexity increase with respect to deterministic routing. The second drawback is due to the fact that adaptive routing may introduce out-of-order packet delivery, which is not acceptable for some applications.

For the best of our knowledge, there are no works that analyze the degree of out-of-order packet delivery caused by different network and traffic conditions. In this paper, we take on such a challenge. We show that only for high traffic conditions (reaching saturation) out-of-order delivery is introduced. Moreover, by using small buffers and simple sorting mechanisms at destination, we show that high network throughput can be obtained at the same time packets are delivered in order. Thus, the paper demonstrates that it is possible to use adaptive routing, while still guaranteeing in-order packet delivery, without using large buffer resources nor degrading significantly its performance.

1. Introduction

Several high-performance network interconnects have been used for building large cluster systems. InfiniBand [6, 7], Myrinet [12], Quadrics [1], and Gigabit Ethernet [14] are some examples. These networks are designed around a switch-based interconnect technology with high-speed point-to-point links providing high bandwidth and low latency. The use of high bandwidth links enables the possibility of achieving a high network throughput. A good topology choice together with the use of a routing algorithm that tries to balance network traffic among all the network links helps in maximizing the eventually achieved network throughput.

Most of these technologies, such as InfiniBand, Quadrics, and Myrinet, use deterministic routing. In deterministic routing, the path from a source node to a destination node is fixed in advance. Packets will use the same path regardless of the current status of the network, thus leading to an ineffective use of the network. Moreover, using deterministic routing will lead to an uneven traffic balance. These circumstances will negatively influence network performance.

Traffic unbalance could be mitigated by dynamically selecting an alternative deterministic path at the source every time a packet is going to be injected. However, performance increment is not very significant [5], especially for large networks.

On the other hand, adaptive routing at switches dynamically balances network traffic by itself. It allows switches to select different output ports (when forwarding a packet) based on the current traffic status. For instance, the switch chooses the proper output port from a set of alternative output ports based on the status of the ports (busy or free) and, therefore, the busy links are skipped. By doing this, network contention in the network is reduced, and in turn, traffic is balanced in a natural way. This allows to improve the utilization of the network resources (links and buffers), leading to a significant increment in network performance. As an example, in [15, 4], adaptive routing algorithms allow network throughput to be increased by a factor of six when used in large irregular networks.

However, adaptive routing has two significant drawbacks. The first one is that additional logic is required at switches. Switches must compute more than one valid output port per packet and must select the final output port

*This work was supported by the Spanish Ministry of Science and Technology under Grant TIC2003-08154-C06-01 and by the JCC de Castilla-La Mancha under Grant PBC-02-008.

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

based on some information. Additionally, for fully adaptive routings [2, 15, 4], virtual channels (implementing escape channels) and the proper selection logic are required in order to ensure deadlock freedom.

There are several works that demonstrate the cost-effectiveness of adaptive routing [3]. On the other hand, adaptive routing feasibility has been proved in some commercial machines, such as the Cray T3E [13] and the Chaos router [8]. Most recently, we have proposed two simple designs in order to support partial [9, 11] and fully [10] adaptive routing in InfiniBand switches. The proposals showed that small changes in the switch architecture were required to support adaptive routing. In fact, only additional logic was required to implement the arbiter and some additional entries in the forwarding table. These proposals demonstrated that switch complexity is not a serious problem preventing the use of adaptive routing.

The second drawback is the out of order delivery of packets that adaptive routing may introduce. Deterministic routing provides in order packet delivery, because all the packets follow the same path through the network when routed towards the same destination (assuming only one path is used for each source-destination pair). Additionally, the FIFO policy applied to queues at the switches and network interfaces contributes to guarantee the in order arrival. However, adaptive routing does not ensure packets to be delivered in order. As switches may choose different output ports for each packet, if network contention appears in the path followed by a packet, the following packets may use other non-congested paths, and thus, may arrive first to destination.

In-order arrival of packets is required by some applications. This is also the case when a reliable transport service is provided. For instance, file transfers must be divided in several ordered packets that must arrive in order to the destination node. Therefore, when using such applications, adaptive routing can not be used unless some special actions are taken.

2. Motivation

Although several works mention that adaptive routing may introduce out of order delivery of packets, also suggesting that a buffer at the source or the destination node will solve the problem [18], to the best of our knowledge, the impact of adaptive routing on the out-of-order degree introduced has not been quantitatively characterized. This analysis would help in choosing the proper mechanism to fix the problem. For instance, if rearrangement buffers are used at destinations, what is the proper buffer size? Moreover, given the size of buffers, what is the strategy that must be used to manage them?

From our point of view, the degree of out-of-order delivery will depend on different network and traffic parameters. The most important parameter is the current load of the network. If the network is low or medium loaded, then contention will be low and therefore, switches will usually provide either the same routing option (output port) to forward packets to the same destination or different options that allow reaching destination with the same number of hops. thus not introducing out-of-order delivery. For high traffic loads, the percentage of contention in the network will be higher, thus introducing delays to some packets, and therefore, increasing the probability of introducing out-of-order delivery.

Also, the out-of-order delivery degree will depend on the way the sources inject packets into the network. As the out-of-order will be produced only among packets of the same flow (related to a particular source-destination pair), if the source interleaves packets from different flows, the injection of consecutive packets of the same flow will be spaced in time, thus lowering the chances of introducing out-of-order delivery. Also, for the same injection rate, larger packets will increment the distance among packet headers.

Finally, the out-of-order delivery degree will depend on different network parameters like the switch architecture and the routing algorithm used. The arbitration policies used in switches may give higher priorities to some input ports, thus potentially delaying the advance of some packets of a connection, increasing the probability of out-of-order delivery at destination nodes. Also, more aggressive adaptive routing algorithms (like fully adaptive routing) may increase the out-of-order delivery degree.

In this paper we take on the challenge of analyzing the impact of adaptive routing on out-of-order delivery. We will quantitatively measure this impact in different scenarios (different traffic patterns, different packet sizes, and different traffic loads).

Once we know the extent of the problem we can propose some solutions to arrange the packets. Although actual systems with adaptive routing already deal with this issue, to the best of our knowledge, there are not detailed descriptions of the involved mechanisms.

In order delivery can be enforced by rearranging packets at the destination network interfaces prior to deliver messages to the host. In order to perform the sorting of packets at destination, some buffers are required to temporarily store all the possible packets that may arrive out of order. From a theoretical point of view, an infinite buffer space will be required at each destination in order to sort all the packets in all the possible arrival orderings. However, the study about the out-of-order delivery degree performed in this paper will help in accurately sizing these buffers.

Once chosen the buffer size, we will propose several mechanisms to manage the rearrange buffers. These mech-

anisms are inspired on well-known rearrange mechanisms, such as the ones used in communication protocols as TCP. In particular, we propose a packet sorting mechanism, an end-to-end flow control, and a simple injection control mechanism that will guarantee in-order packet delivery. We will also tune the resources devoted to it.

To sum up, the main goal of this paper is to quantify the degree of out-of-order packet delivery. Also, we demonstrate that it is possible to use adaptive routing and, at the same time, guaranteeing in-order packet delivery without using large buffer resources. In addition, we will show that it is possible to achieve similar throughput as plain adaptive routing (i.e. without enforcing in-order delivery).

The rest of the paper is organized as follows. In Section 3 we will present the rearrangement mechanism we propose to sort the packets received out of order at destinations and the required end-to-end flow control, together with the injection control mechanism applied. In Section 4 we will show evaluation results. Finally, in Section 5 some conclusions will be drawn.

3. Enforcing In-Order Delivery

Packets may arrive to the destination node out of order when adaptive routing is used. In order to solve this problem, we propose a mechanism that stores packets arrived out-of-order at destination network interface (NIC), together with an end-to-end flow control and a packet injection limitation mechanism at the source nodes. In what follows, we will describe how this mechanism works.

First, the mechanism needs to know the actual order of the packets, so a sequence number should be assigned to each packet when it is generated. Hence, the source node should assign consecutive sequence numbers to consecutive packets belonging to the same connection. Moreover, source and destination nodes must agree on the first sequence number to use when a connection is established.

When a packet arrives at its destination node, the sequence number is checked to know whether this packet is the next packet to be delivered to the host or, on the contrary, it has arrived out of order. In the former case, the NIC can immediately deliver the packet to the application. In the latter case, two alternative actions could be carried out: storing the packet at the NIC or rejecting it. We propose an hybrid scheme that stores the packet if possible, rejecting it if there is not available free space. In this latter case, the source node must be notified, because it will have to inject the packet again. To this end, an end-to-end flow control protocol able to notify if a packet is either delivered to the hosts or rejected should be provided. Indeed, source nodes should temporarily hold packets that have not been acknowledged yet.

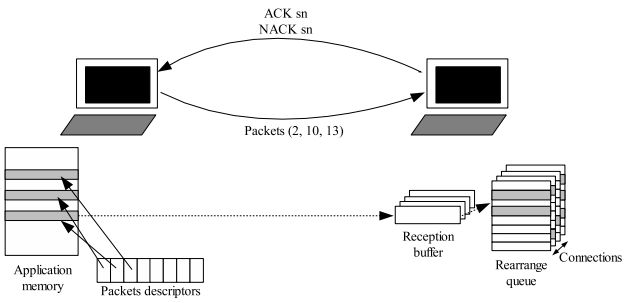

Figure 1. Rearrange buffers at destination node.

3.1. End-to-end Flow Control Mechanism

A simple flow control mechanism is proposed. It requires that the network supports packet acknowledgments. This feature is available when reliable communication is supported (for instance, reliable transport services in InfiniBand). We assume that *<ACK sn>* and *<NACK sn>* control packets, where *sn* stands for a sequence number, are available. Both of them are sent back to the source by the destination node in order to report whether the referenced packet has been either correctly delivered or rejected. An *<ACK sn>* packet acknowledges that all packets whose sequence number is lower than or equal to *sn* have been correctly delivered to the application. On the other hand, a *<NACK sn>* packet reports that the packet whose sequence number is equal to *sn* has been rejected.

When the source node receives an *<ACK sn>* belonging to a certain connection, it removes from the sending queue all the requests belonging to all the acknowledged packets that were temporarily stored. Also, the source application will be reported about the correct reception of the packets at the destination host. When the source receives a *<NACK sn>* packet, it has to resend the packet whose sequence number is equal to that specified by the *<NACK sn>* packet.

This flow control, on its own, ensures that all packets will be delivered in-order to the destination application, even if there is no buffer at the destination nodes. However, if buffers were not used at the destination node, this mechanism would be inefficient because most packets that arrive out of order would have to be injected again into the network, causing an increment in the network traffic and leading to a severe degradation in performance.

3.2. Rearrange Buffers

In order to avoid rejecting all the packets that arrive out of order, a rearrange buffer per connection may be implemented at the destination network interface (see Figure 1). Each connection has a reception buffer and a rearrange

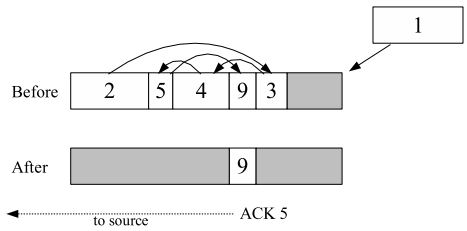

Figure 2. Rearrange queue. Example 1.

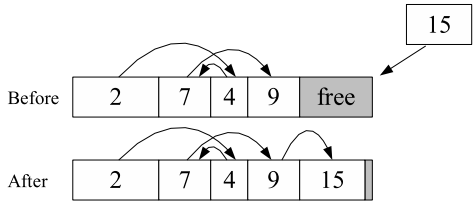

Figure 3. Rearrange queue. Example 2.

queue at destination node. When a packet is being received, it is stored in the reception buffer. Then, its sequence number is checked to decide whether this packet has to be directly delivered to the application or it has just arrived out of order and, then, has to be temporarily stored in the rearrange buffer.

When the packet that is being expected at the destination node arrives, this one and all of the packets that follow it in the sequence order and which are already stored in the rearrange buffer will be also delivered to the application, thus freeing the rearrange buffer and sending an *<ACK sn>* for all the accepted packets (see Figure 2). If a packet arrives out of order to the destination node and there is enough free rearrange buffer space for storing it, then this packet is stored in the rearrange buffer (see Figure 3) and no ACK packet is sent. On the other hand, if a packet arrives out of order to the destination node and the buffer is full (see Figure 4), the NIC will try to free enough buffer space rejecting packets with greater sequence number than that of the packet newly arrived (i.e., the packets that were sent later). If the NIC succeeds in freeing the buffer, the incoming packet will be stored in the buffer whereas the packet

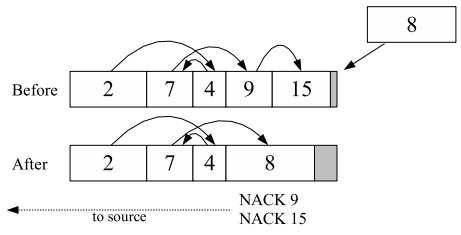

Figure 4. Rearrange queue. Example 3.

or the packets removed from the buffer are rejected. In this case, one *<NACK sn>* packet per rejected packet will be returned towards the source node. If the NIC cannot free enough buffer space, no packet in the buffer is removed, and only the incoming packet is rejected (the appropriate *<NACK sn>* packet is sent back to the source).

The destination buffer may be managed in a static or a dynamic way. In the former case, there exists a buffer dedicated to each connection whose size is fixed. Therefore, number of connections is bounded by the memory used. Obviously, this solution is not scalable. When dynamic memory allocation is used, the buffer allocated to each connection is variable. Indeed, all the memory is shared by the buffers associated with all connections. In order to avoid that one connection fills up all the available space, a maximum size of buffer occupation per connection can be imposed.

The network interface has to implement some new data structures in order to track the order of the packets stored in the buffers. The management of these structures requires some processing time and may increase the time required for processing a packet in the network interface. However, these actions may be overlapped with the reception of the previous packet and, hence, no penalty is considered in the reception of the packets.

3.3. Injection Control Mechanisms

When a *<NACK sn>* packet is received at the source node, it may react in several ways. The easiest way is resending immediately the rejected packet. However, notice that this packet actually arrived to its destination and was rejected due to the lack of buffer space. Hence, it is likely that this time it will be again rejected. In fact, this packet arrived to destination before some older packets because they are surely blocked in the network. If these packets are stopped for a long time, the destination buffer assigned to this connection may become full, causing some packets to be repeatedly rejected. So, injecting immediately the packets that have been rejected will increase even more the network traffic, increasing congestion and decreasing performance.

In order to avoid this performance degradation, a different strategy has been developed. It is based on keeping two states per each connection, namely unlimited injection (UI) and limited injection (LI) states. When a connection is in the UI state, all newly generated packets are injected into the network as soon as possible. Also, send requests are stored in the sending queue queue until the corresponding *<ACK sn>* packet is received. When there are one or more outstanding packets that have been rejected at least once, then the connection switches to the LI state. Notice that when a connection receives a NACK is because

some paths that connect source and destination nodes are congested. Sending more packets through these paths may increase network contention. Limiting the injection rate for this connection may help in decreasing contention. In the LI state, a packet is injected or re-injected only when an *<ACK sn>* packet for this connection has been received. Indeed, in this situation, a number of packets up to the number of acknowledged packets may be injected. Finally, the connection status is switched to the UI state when all the pending rejected packets are acknowledged by the destination (through the corresponding *<ACK sn>* packets).

Notice that when using static memory allocation at the NIC, a TCP-based injection limitation may be used. In this situation, the source node may take advantage of knowing the free buffer space at destination node. Thus, when the connection is in the LI state, the source node will just inject the number of packets that fit at destination buffer.

4. Evaluation

In this section we will first evaluate the degree of out-of-order delivery of packets when using adaptive routing. Then, we will select the proper buffer size at destinations and evaluate the proposed mechanisms to guarantee in-order packet delivery and their impact on the network performance. We have chosen an IBA-based network to evaluate these mechanisms. For this purpose, we have developed a detailed simulator that allows us to model the network at the register transfer level following the IBA specs [7].

4.1. The IBA Subnet Model

The IBA specification defines a switch-based network with point-to-point links, allowing the user to define any topology. Packets are routed at each switch by accessing the forwarding table, that contains the output ports to be used at the switch for each possible destination. Several routing options are provided by using the adaptive routing algorithm proposed in [4]. The output port is selected at arbitration time by considering the status of the requested output ports and the number of credits available.

Switches can support up to 16 virtual lanes. We will use a non-multiplexed crossbar on each switch. 8 KB buffers will be used both at the input and the output side of the crossbar. The escape queues implemented on each buffer [4] will be fixed to the MTU size in order to store only one packet. The rest of the buffer implements the adaptive queue. IBA allows the definition of different MTU values for packets, ranging from 256 to 4096 bytes.

The switch routing time will be set to 100 ns, including the time to access the forwarding tables, the crossbar arbiter time, and the time to set up the crossbar connections. Links in InfiniBand are serial. 10/8 coding [7] is used. In the simulator, the link rate will be fixed to the 1X configuration [7] (2.5 Gbps). We will model 20 meter copper cables with a propagation delay of 5 ns/m.

The IBA specification defines a credit-based flow control scheme for each virtual lane with independent buffer resources. A packet will be transmitted over the link if there is enough buffer space (measured in credits of 64 bytes) to store the entire packet. Additionally, the virtual cut-through switching technique is used.

We have evaluated the network using both synthetic traffic and traces. In the former case, uniform and bit-reversal packet destination distributions have been used. Only one connection per each source-destination pair is allowed. Packets are 256 bytes long. In all the presented results, we will plot the average packet latency[1] measured in nanoseconds versus the average accepted traffic[2] measured in bytes/ns/switch.

We will analyze irregular networks of 16 and 64 switches, randomly generated but following some restrictions. First, we will assume that every switch in the network has the same number of ports (we used 8 ports) and the same number of nodes connected to every switch (4 ports in our simulations). And second, neighboring switches will be interconnected by just one link. Ten different topologies will be randomly generated for each network size. We will plot the results for some representative topologies for every network size. Also, we will analyze a 4x4 Torus network. As far as the mechanism that allows adaptive routing in InfiniBand [4] only defines one escape channel and one adaptive channel, the routing algorithm applied for escape channels will be up*/down* in both regular and irregular networks.

Also, I/O traces have been used. They were provided by Hewlett-Packard Labs [16] . They include all the I/O activity generated from 1/14/1999 to 2/28/1999 at the disk interface of the Cello system. A detailed description of similar traces of 1992, collected in the same system, can be found in [17]. As these I/O traces were obtained several years ago, taking the improvement of I/O devices along the years, we have applied several time compression factors to the traces in order to generate higher injection rates to the network. We will use packets with a payload equal to the size specified in the trace for the I/O accesses. However, if the size is larger than the MTU, we will split the packet into several packets with a payload of up to the MTU. The number of workstations of the I/O traces is 128. Network size is 32 switches when using irregular networks and 16 switches for the torus network. In the first case, four workstations are connected to each switch, whereas in the second one the

[1] Latency is the elapsed time between the generation of a packet at the source host until it is delivered at the destination end-node.

[2] Accepted traffic is the amount of information delivered by the network per time unit.

number of workstations connected to each switch is eight. The disks will be randomly attached to 23 switch ports.

4.2. Buffer Requirements

First, we assume a mechanism which only relies on the buffer resources available at the destination NICs to temporarily store the packets received out of order. The requirements of buffer size at the destination nodes to rearrange all the packets arrived out of order will allow us to quantify the degree of out-of-order delivery. This evaluation has been made using uniform and bit-reversal traffic patterns.

Figure 5 shows the obtained results. Network throughput is double for small network sizes, and more than four times for large networks when using adaptive routing. As expected, the amount of buffer needed depends on the traffic pattern. With uniform traffic (Figure 5), it is unlikely that two consecutive messages are sent from the same source to the same destination. So, messages belonging to the same connection are separated in time. Hence, it is unlikely that a packet overtakes another one belonging to the same connection. This is the reason by which the buffer space needed by uniform traffic is very small. Only less than 600 bytes (2 packets) per buffer (connection) is required in the worst case.

On the other hand, with bit-reversal traffic (Figure 5) each source always communicates with the same destination. So, out-of-order delivery is more common and buffer requirements are higher. In fact, as traffic increases, the buffer size required to avoid out-of-order packet delivery grows very fast.

In both cases, the greatest buffer size is needed only when the network is near saturation. As far as the network does not reach saturation, the buffer requirements are small. This behavior is similar for small and large networks.

4.3. Limited Buffer with Static Allocation

In this section, we will evaluate the network behavior when each destination has one rearrange buffer per connection with a fixed size.

Firstly, we are going to analyze the influence of the buffer size on network performance when using synthetic traffic (see Figure 6) and different network sizes. We use the simplest technique of sending back to the source a <*NACK sn*> packet, if the buffer is full. The source will re-inject the packet. We have chosen bit-reversal traffic because it has more buffer requirements. Figure 6 shows that using adaptive routing improves the performance with respect to deterministic routing, even when the buffer size is smaller than the packet length (Adap_32B and Adap_128B). In fact, both curves are superimposed. In this case, no packet will be stored in the buffers, all packets that arrive out of order will be discarded and later re-injected. Also, as the queue size is increased, the network performance is close to the original adaptive routing.

Bit-reversal traffic only connects each source to one destination, thus the destination node only needs memory for one buffer. On the other hand, uniform traffic does not need large buffers, as it has been seen in the previous section. So, in order to know the overall amount of memory needed at destination node, we need to use a more realistic traffic. In what follows, we will evaluate network performance by using I/O traces. We will compare the deterministic routing algorithm with a fully adaptive routing algorithm when using different strategies to manage packet reinjection at source nodes. As we stated before, the simplest one consisting of re-injecting packets as soon as a <*NACK sn*> packet has been received is not effective. In fact, simulations show that if no injection limitation is used, trace execution time may double the trace execution time with deterministic routing. Therefore, we have only shown the results for the other techniques that use injection limitation. The first one only re-injects a rejected packet after having received <*ACK sn*> packets when the sender is in the limited injection (LI) state. It will be referred to as ADAP-LI. The second mechanism injects a number of rejected packets according to the available space at destinations when the sender is in the LI state. It will be referred to as ADAP-LW (Limited per Window). Because this mechanism requires knowing in advance the allocated buffer space at destination nodes, it can not be applied when using dynamic memory allocation (refer to Section 3.3). For comparison purposes, we will also evaluate the behavior of a plain adaptive routing algorithm that does not track the in-order delivery of packets. We will refer to this case as ADAP_OOO.

In Tables 1 and 2 we can see how the proposed mechanisms influence the execution time of the applied trace with regular and irregular networks. As can be seen, adaptive routing outperforms deterministic routing if out-of-order packet delivery is allowed. So, trace execution time is lower. However, when applying the mechanisms proposed for guaranteeing in order packet delivery using buffer sizes of 2 KB per connection, the advantage with respect to deterministic routing totally disappears. In fact, the results obtained by adaptive routing are worse than the ones for deterministic routing. Also, we can see that waste traffic[3] ranges from 20% to 30%. Taking into account that the percentage of <*ACK sn*> and <*NACK sn*> packets injected in the network is about 1% of the traffic, more than 20% of traffic that arrives to a destination is rejected because there is no free buffer at destination NIC due to the small buffer size. When a connection changes its state to the UI

[3] Traffi c generated by the reorder mechanism for acknowledgments and for resending packets that have been rejected at destination nodes.

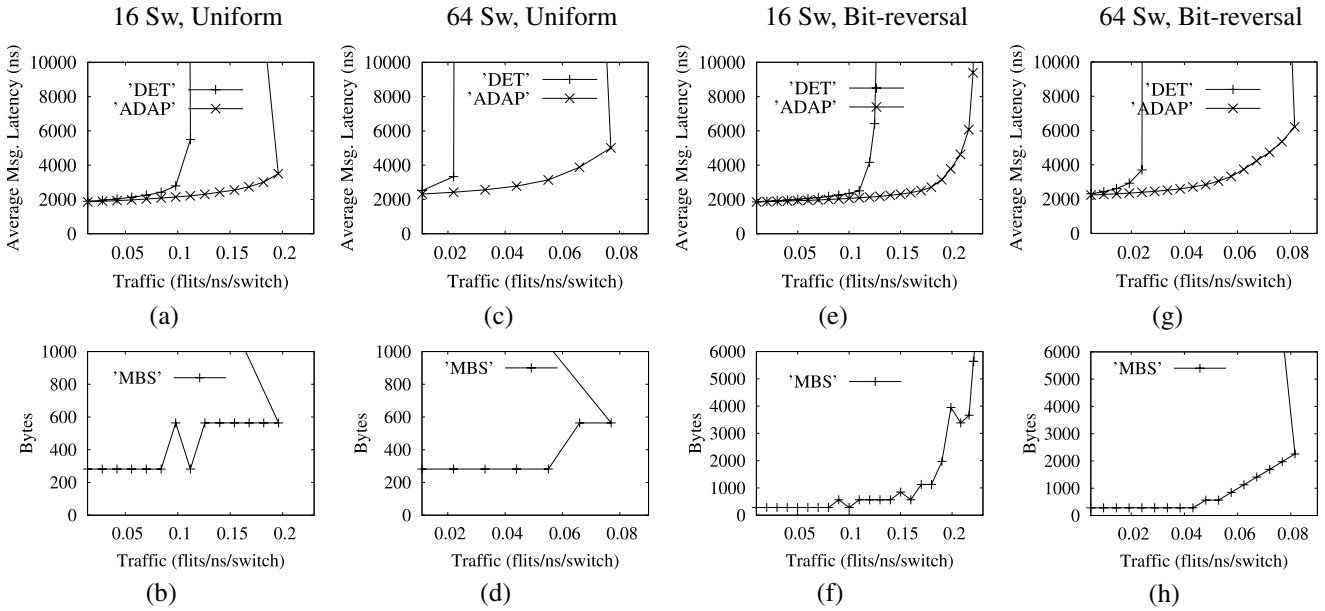

Figure 5. (a,c,e,g) Average packet latency vs. traffic. (b,d,f,h) Maximum buffer size needed vs. traffic. Packet length is 256 bytes. MBS stands for Maximum Buffer Size per connection.

state, the source node injects a burst made up of packets that have been delayed at the source node. Note that this burst passes through a possibly congested location in the network. Thus, these packets will probably arrive out-of-order to the destination, probably causing the destination buffer to overflow again.

Now, we will analyze what happens when increasing the queue size. In the same tables we can see the performance of the proposed mechanisms when using a buffer size of 32 KB with static memory allocation. In this case, adaptive routing with injection limitation decreases the trace execution time with respect to deterministic routing by about 18% for Torus network and 25% for irregular networks (see Tables 1 and 2). In the case of the Torus network, note that the improvement of 18% is similar to the one achieved by adaptive routing with out-of-order delivery. Moreover, waste traffic does not exceed 8% in irregular networks and 4% in the Torus network. As can be seen in these tables, the best mechanism is the one that achieves the greatest reduction in waste traffic, while still taking advantage of adaptive routing. Either Adap-LI or Adap-LW work fine, being Adap-LI preferable due to its easier implementation. In fact, the achieved improvement with respect to deterministic routing is quite close to that achieved by adaptive routing with out-of-order packet delivery.

By analyzing Tables 1 and 2, we can see that the maximum amount of memory that is used at destination nodes when using static memory allocation never exceeds 20% of the total amount of memory available. For instance, when using buffers of 32 KB, the NIC has to reserve a total amount of 4 MB for destination buffers. However, the maximum amount of required memory is only 400 KB. In Figure 7, the maximum amount of required buffer space for all nodes is shown. We can see that almost all the buffers that contain at least one packet are either full or occupied above 50%. The rest of the buffers are completely empty. Therefore, in the worst case for every node, the active connections require more buffers, whereas buffers dedicated to inactive connections are not used at all (almost 80% of unused buffers).

To conclude, we have obtained that using rearrange buffers of a reasonable size allows to use adaptive routing and at the same time guarantee in-order delivery, keeping the same performance than the original adaptive routing with out-of-order delivery. If static memory allocation is used, every connection has to arrange a large buffer. However more than 90% of the buffers will be empty (see Figure 7), wasting storage resources.

4.4. Limited Buffer with Dynamic Allocation

Alternatively, memory could be dynamically allocated, trying to meet the trade-off between maximizing network performance and minimizing the overall amount of memory really used for rearranging those packets delivered out of order.

Tables 1 and 2 show results when using dynamic memory allocation at destination nodes with a total amount of memory at each destination NIC of 256 KB and a maximum buffer size per connection limited to 128 KB. We can

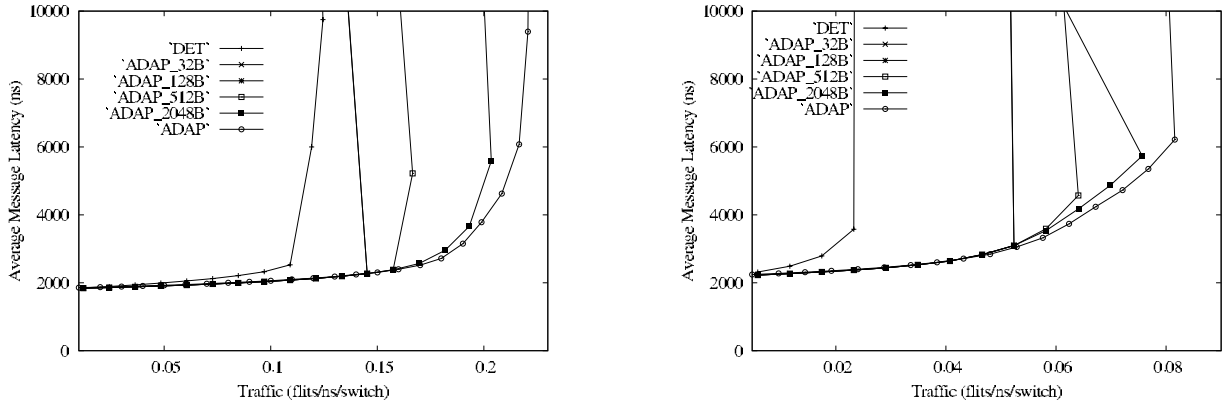

Figure 6. Average packet latency vs. traffic. (a) 16 switches and (b) 64 switches. Bit-reversal traffic. Packet length is 256 bytes.

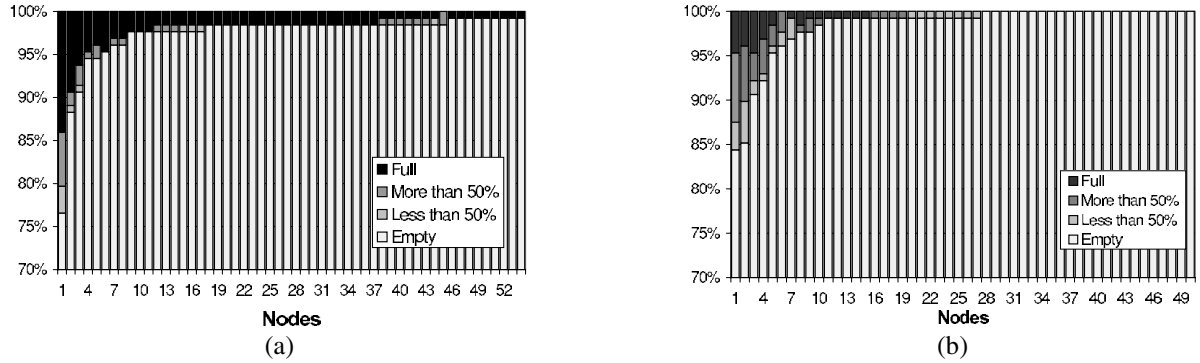

Figure 7. Worst case of total buffer occupation at the same time in each destination. ADAP-LI. Buffers = 2 KB. (a) Irregular network using 2 compression factor. (b) 4x4 Torus network using 2.5 compression factor.

see that the time required to deliver all the injected load is reduced by 28% and 20% for irregular and regular networks, respectively. As can be seen, these results are quite similar to the ones obtained by using static memory allocation with buffer size of 32 KB per connection at each destination. However, the total memory size is reduced by a factor of sixteen (256 KB instead of 4 MB). Dynamic allocation makes an efficient use of the overall amount of memory dedicated to rearrange buffers. Active connections that require more resources use more memory at the expense of the idle ones. Therefore, the number of rejected packets is reduced and waste traffic is lower than 4% and 1% for irregular and regular networks, respectively.

In the same tables we can observe the results when increasing the shared memory size up to 4 MB. The limit of the buffer size available to each connection is fixed to 1 MB. In this case, using adaptive routing with rearrange buffers allows to roughly achieve the same performance as adaptive routing with out-of-order packet delivery. Indeed, there is no buffer that needs more than 650 KB, so no packet will be rejected. In this scenario, the waste traffic is only due to the traffic generated by flow control packets.

Another factor that could affect the out-of-order delivery degree is the size of the packets. A network with a small MTU causes more packetization and, hence, more overhead, due to the header and the tail of packets, affecting the network performance. Also, out-of-order degree may increase because small packets can overtake each other easier than if they are long packets. This might cause the amount of memory needed at destination buffers to increase. In Figure 8 we explore the effect of MTU on the trace execution time. ADAP_LI with Dynamic memory allocation is used with several memory sizes ranging from 64 KB to 2 MB. We have evaluated MTUs ranging from

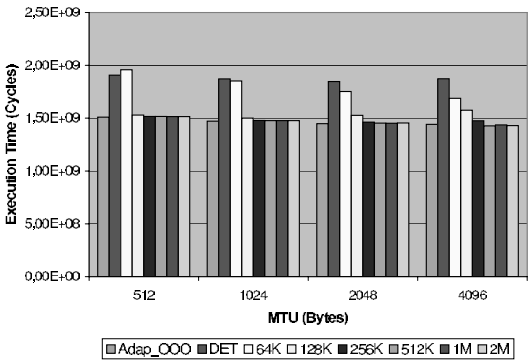

Figure 8. Execution time vs. MTU. 4x4 Torus. Compression factor is 2.5. Dynamic memory allocation is used with several sizes for adaptive routing.

512 bytes to 4 KB. As can be seen, performance of deterministic routing and Adap_OOO are roughly independent of MTU size. Indeed, for both there is a slight decrease in execution time as the MTU size increases. This is due to the fact that less packetization is required for larger MTUs and there is a lower overhead of packet headers. More interesting, the Adap_LI behaves different depending on the combination of MTU and memory size. For 64 KB memory size, Adap_LI reduces its execution time as the MTU increases, because larger MTUs reduce the out-of-order degree. As 64 KB seems to be not enough to efficiently deal with out-of-order packets (execution time is similar to the one obtained by deterministic), reducing the out-of-order degree has a greater impact. This can be deduced from the results for larger memory sizes. A particular situation occurs when the memory size is equal to 128 KB, observing an slight increase in the execution time as the MTU increases. This fact means that this memory size is not large enough as to satisfy the greater memory requirements due to the use of larger MTU sizes, which cannot be offset by the reduction in the out-of-order degree. From 256 KB upwards, these requirements are met and the behavior of ADAP_LI is very close to ADAP_OOO.

5. Conclusions

Out-of-order delivery of packets has been traditionally argued against the use of adaptive routing in interconnection networks. In this paper we first quantify the impact of adaptive routing on the out-of-order delivery of packets. We analyze the impact of several issues such as the network size, the traffic pattern, and the traffic rate. The results show that out-of-order delivery only appears when the network load is intense. The problem is also more prominent with traffic patterns that send many packets between

Table 1. Comparison of ADAP-LI and ADAP-LW under trace traffic for a 32-switch irregular network using different environments. MBS and MMS stand for Maximum Buffer Size per connection and Maximum Memory Size, respectively.

Irregular topology with 32 switches. Compression factor: 2					
Without rearrange mechanism					
	Time	Waste %	MBS	MMS	Impr. %
DET	1961.09M	0%	-	-	-
Adap-OOO	1286.21M	0%	-	-	34.41%
Static memory 2 KB/conn. (total memory: 256 KB)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	2011.33M	25.8%	2560	52812	-2.56%
Adap-LW	1954.89M	20.8%	2560	48106	0.32%
Static memory 32 KB/conn. (total memory: 4 MB)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	1421.1M	7.4%	33280	345272	27.54%
Adap-LW	1412.56M	6.8%	33280	405528	27.97%
Dynamic memory of 256 KB (max. 128 KB/conn)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	1404.24M	3.8%	131584	262656	28.39%
Dynamic memory of 4 MB (max. 1 MB/conn)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	1298.59M	0.3%	666494	915552	33.78%

the same source-destination pair.

Out-of-order delivery can be easily solved by using rearrange buffers at destinations. The fact that out-of-order delivery only appears with high traffic rates suggests us to select a reasonable buffer size, also proposing some mechanisms to manage it. Taking into account the memory available at current network interface cards, we have run several simulations considering that up to 4 MB are available to rearrange buffers. In addition, we use an end-to-end flow control that sends negative acknowledgments in the case there is no space in the rearrange buffers. Also, we propose a simple injection control mechanism that temporarily avoids injecting packets to a destination that has rejected packets.

Evaluation results under real traffic show that the proposed mechanisms are able to provide in-order packet delivery when using adaptive routing, while still maintaining similar network performance to that provided by adaptive routing with out-of-order delivery. The efficiency of the mechanisms strongly depends on the memory allocation strategy. When using static memory allocation, it is required a huge buffer space. This solution is not scalable and inefficient, because more than 90% of allocated buffers remain empty all the time. On the other hand, using dynamic memory allocation allows the required buffer space to be significantly decreased (up to a factor of 16) at the expense of slightly degrading the network performance with respect to adaptive routing with out-of-order delivery. In this case, the required buffer space at the destination network interfaces can be considered reasonably small (about

Table 2. Comparison of ADAP-LI and ADAP-LW under trace traffic for a 32-switch irregular network using different environments. MBS and MMS stand for Maximum Buffer Size per connection and Maximum Memory Size, respectively.

4x4 Torus. Compression factor: 2.5					
Without rearrange mechanism					
	Time	Waste %	MBS	MMS	Impr. %
DET	1906.72M	0%	-	-	-
Adap-OOO	1509.12M	0%	-	-	20.85%
Static memory 2 KB/conn. (total memory: 256 KB)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	2324.63M	29.4%	2560	52378	-21.91%
Adap-LW	2197.88M	24.2%	2560	45644	-15.27%
Static memory 32 KB/conn. (total memory: 4 MB)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	1573.82M	4.1%	33280	164462	17.46%
Adap-LW	1559.49M	3.6%	33280	154528	18.21%
Dynamic memory of 256 KB (max. 128 KB/conn)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	1516.31M	0.7%	131580	262604	20.48%
Dynamic memory of 4 MB (max. 1 MB/conn)					
	Time	Waste %	MBS	MMS	Impr. %
Adap-LI	1515.60M	0.3%	475522	571066	20,51%

256 KB).

To conclude, by using rearrange buffers of a reasonable size at destinations and a set of simple mechanisms to manage them, in-order packet delivery is enforced, thus eliminating one of the traditional drawbacks that prevent the use of adaptive routing in commercial switches.

References

[1] Quadrics homepage *http://www.quadrics.com*.

[2] W.J. Dally and H. Aoki, "Deadlock-free Adaptive Routing in Multiprocessor Networks Using Virtual Channels," *IEEE Transactions Parallel and Distributed Systems*, vol. 34, no. 4, pp. 466-475, April 1993.

[3] J. Duato and P. López,"Performance evaluation of adaptive routing algorithms for k-ary n-cubes", *Parallel Computer Routing and Communication (PCRCW'94)*, Seattle, USA, Lecture Notes in Computer Science 853, Springer-Verlag, 1994.

[4] J. Duato, A. Robles, F. Silla, and R. Beivide, "A comparison of router architecture for virtual cut-through and wormhole switching in a NOW environment," in *Journal of Parallel and Distributed Computing* 61, 224-253 (2001).

[5] J. Flich, *Improving performance of networks of workstations with source routing*, Ph. D. Thesis, March 2001.

[6] InfiniBand℠ Trade Association. *www.infinibandta.com*.

[7] InfiniBand℠ Trade Association, *InfiniBand℠ Architecture. Specification Volume 1. Release 1.0*. Available at *www.infinibandta.com*.

[8] S. Konstantinidou and L. Snyder, "The Chaos router," in *IEEE Transactions on Computers* 43, 2 (1994), 1386-1397.

[9] J.C. Martínez, J. Flich, A. Robles, P. López and J. Duato, "Supporting Adaptive Routing in InfiniBand Networks," in *11th Euromicro Workshop in Parallel, Distributed and Network-Based Processing*, pag. 165-172, 2003.

[10] J.C. Martínez, J. Flich, A. Robles, P. López and J. Duato, "Supporting Fully Adaptive Routing in InfiniBand," in *Proceedings of the 17th International Parallel and Distributed Processing Symposium* 2003.

[11] J.C. Martínez, J. Flich, A. Robles, P. López and J. Duato, "Supporting Adaptive Routing in IBA Switches," in *Journal on system Architecture* vol 49, pag 441-456.

[12] Myricom homepage, *www.myri.com*.

[13] S.L. Scott and G.M. Thorson, "The Cray T3E Network: Adaptive Routing in High Performance 3D Torus," *Proceedings 4th Hot Interconnects Symposium*, August 1996.

[14] R. Sheifert, "Gigabit Ethernet," in *Addison-Wesley*, April 1998.

[15] F. Silla, M.P. Malumbres, A. Robles, P. Lopez, and J. Duato, "Efficient adaptive routing in networks of workstations with irregular topology," in *Proceedings of the Workshop on Communications and Architectural Support for Network-based Parallel Computing*, 1997.

[16] SSP homepage, *http://ginger.hpl.hp.com/research/itc/csl/ssp*.

[17] C. Ruemmler, J. Wilkes, "Unix Disk Access Patterns", in *Winter Usenix Conference*, Jan 1993.

[18] W.-D. Weber *et al.*, "The Mercury Interconnect Architecture: a cost-effective infrastructure for high-performance servers" in *ISCA '97: Proceedeings of the 24th annual international symposium on Computer architecture*, ACM press, pag 98-107, 1997