A Method for Applying Double Scheme Dynamic Reconfiguration over InfiniBand^{TM}

Timothy M. Pinkston, Bilal Zafar and Jose Duato

CENG Technical Report 02-03

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213) 740-4475

March 2002

Abstract

InfiniBand Architecture is a newly established general-purpose interconnect standard applicable to local area, system area and storage area networking and I/O. It is designed to provide significantly higher levels of reliability, availability, performance, and scalability than alternative technologies. Networks based on this standard should be capable of tolerating topological changes due to resource failures, link/switch activations, and/or hot swapping of components. In order to maintain connectivity, the network's routing function may need to be reconfigured on each topological change. Although the architecture has various mechanisms useful for configuring the network, no strategy or procedure is specified for ensuring deadlock freedom during dynamic network reconfiguration. In this paper, a method for applying the Double Scheme [1] over InfiniBand networks is proposed. The Double Scheme provides a systematic way of reconfiguring a network dynamically while ensuring freedom from deadlocks. We show how features and mechanisms available in InfiniBand Architecture for other purposes can also be used to implement dynamic network reconfiguration based on the Double Scheme.

Keywords: InfiniBand Architecture, Dynamic Reconfiguration, Double Scheme.

1 Introduction

In the wake of increasing market demand for high performance computing, clustered systems have emerged as a favored solution for low-end commodity systems as well as high-end servers [2, 3]. The flexibility, scalability and cost/performance capabilities of clustered systems are among some of the things that make them so attractive. While certain interconnect subsystems like Ethernet [4], Autonet [5], Myrinet [6] and Fibre Channel [7] have traditionally been used, cluster computing/storage systems are shifting toward an open, non-proprietary, low-overhead, switched interconnect paradigm that provides not only high-performance communication but also high reliability, availability and dependability. These attributes are increasing in importance with the emergence of bandwidth-hungry applications such as high-definition video/audio on-demand processing, distributed on-line transaction processing, database and decision support systems, and the like. Such applications impose a great demand on the communication subsystem not only to be high performance but also to be highly robust.

Toward this end, InfiniBand Architecture (IBA) [8] is a newly established general-purpose interconnect standard designed to solve a wide spectrum of interprocessor communication and I/O problems associated with servers and cluster systems. In addition to providing low latency and high bandwidth point-to-point communication support, it also includes certain features and mechanisms useful for improving system reliability and availability. Features such as subnetwork management, service levels, data and control virtual lanes, table-driven routing, end-to-end path establishment, packet time-out and virtual destination naming are all useful for implementing reconfiguration functions in IBA networks. To be truly dependable, however, IBA-compliant networks should be capable of *deadlock-free dynamic reconfiguration*, able to efficiently adapt to changes in real-time if and when voluntary or involuntary changes occur. That is, IBA networks should remain up and running with high performance in the presence of hot-swapping of components, failure or addition of links/nodes, activation or deactivation of hosts/switches, partitioning or isolation of network components, etc., arising from changes in users' needs and/or system state. The reliability, availability, and performance predictability (overall, the dependability) of IBA-compliant cluster computing and storage systems depend critically on the networks's ability to efficiently support such functions while maintaining certain service level targets.

Although IBA has various mechanisms useful for network configuration, no strategy or procedure is specified in the standard for ensuring deadlock freedom during dynamic reconfigura-

tion of the network. Network configuration mechanisms are specified for detecting and acquiring information about constituent network components (e.g., active links, ports, switches/routers, host/target channel adapters, buffer/queue sizes, etc.), assigning an identity to those components, and mapping (or topologically graphing) connections among the active components physically established through active links. Source-based routing functions—which supply the possible paths packets can take in the network to reach their destinations from their current location—are supported by various mechanisms using forwarding tables within IBA network switches and channel adapters. Mechanisms are also specified for periodically monitoring system state to allow the network to react to events that may require it to undergo reconfiguration. Although many things such as these are specified, no details are included in the standard that specify how routing or reconfiguration should be carried out in a deadlock-free manner. Deadlock can occur when packets block cyclically waiting for resources while holding onto other resources indefinitely [9, 10]. If allowed to persist, deadlocks can bring the entire system to a standstill, making it vitally important for both the routing algorithm and the reconfiguration technique to guard against them.

Recent work has been done on computing deadlock-free routing paths in IBA networks [11, 12], but there is no literature to-date on solving the difficult problem of deadlock-free dynamic reconfiguration of IBA-compliant networks. What makes this a hard problem to solve is that dynamic reconfiguration (unlike static reconfiguration) does not halt the injection or delivery of user packets in the network before or during the reconfiguration process. Although severe performance degradation is prevented as measured by network latency, throughput, and packet loss (as shown in [1, 13]), dynamic reconfiguration allows packets to be routed in the network under the influence of multiple routing functions—an old one existing before reconfiguration and a new one existing afterwards. If one or more packets are subjected to both routing functions, residual dependencies on network resources from the old routing function can interact in an illegal way with current dependencies from the new routing function. This can cause deadlock even if both routing functions independently are designed to be deadlock-free.

Several schemes have been proposed in the literature to combat this problem, but most do not appear very applicable to InfiniBand Architecture. The NetRec scheme [14] requires every switch to maintain information about nodes some number of hops away and is only applicable to wormhole networks. IBA does not provide mechanisms to keep track of such information in switches or channel adapters, and it is packet-switched, not wormhole switched. Partial Progressive Reconfiguration (PPR) [15] requires a complicated sequence of synchronizing steps to progressively

update old forwarding table entries with new ones while ensuring that no cycles form. Although this scheme works for packet-switched networks, it is not clear whether InfiniBand has all the complicated mechanisms needed to support this complex technique.

In this paper, a straightforward method for applying the *Double Scheme* [1] over InfiniBand networks is presented. In prior work, the Double Scheme is proven to provide a systematic way of reconfiguring a network dynamically while ensuring deadlock freedom. It is generally applicable to virtual cut-through (packet-switched) networks, independent of the routing function or topology being implemented. In this work, we show how features and mechanisms available in InfiniBand Architecture for other purposes can also be used to implement Double Scheme dynamic reconfiguration. Performance advantages for the Double Scheme in comparison to static reconfiguration have been documented in prior work [1]. The contribution of this paper, therefore, is in the straightforward methodology presented for applying this technique to InfiniBand using mechanisms already included in the standard. This work allows InfiniBand networks to better support applications requiring certain quality of service (QoS) guarantees that do not well tolerate intermittant performance drops-offs, as would be the case without deadlock-free dynamic reconfigurabilty.

The remainder of this paper is organized as follows. Section 2 gives an overview of the Double Scheme and useful InfiniBand mechanisms. Section 3 describes the proposed method for implementing the Double Scheme over IBA, followed by an example and discussion in Section 4. Finally, conclusions and future work are presented in Section 5.

2 The Double Scheme and Exploitable InfiniBand Mechanisms

2.1 The Double Scheme

The Double Scheme [1] provides a straightforward way of updating a network's routing function in a deadlock-free manner when the network undergoes dynamic reconfiguration. Many variations of the scheme exist, however the basic idea behind the scheme can be summarized as follows. At all times, packets are routed under the influence of one and only one routing function—either the old routing function (R_{old}) existing before reconfiguration or the new one (R_{new}) corresponding to the new configuration, but never both. This is accomplished simply by spatially and/or temporally separating the routing resources used by each routing function into two sets: one used exclusively by R_{old} and the other by R_{new}. By allowing dependencies to exist from one set of re-

sources to the other but not from both at any given time, a guarantee on deadlock freedom during and after reconfiguration can be proved [1].

One possible scenario on how this could work is the following. The routing function before reconfiguration, R_{old}, allows packets to be injected into a connected set of routing resources (designated as C_{old}) supplied by R_{old}. Once the need for a reconfiguration event is determined and the new routing function R_{new} is computed, a connected set of routing resources (designated as C_{new}) is required to become available for use by newly injected packets routed under the influence of R_{new} which supplies those resources. This could be done by allowing packets in a subset of C_{old} resources to be delivered to their destinations while not injecting new packets into that subset, which essentially drains those resources. Non-routable packets encountering the topological disconnectivity which caused the need for reconfiguration can be discarded. As packets are no longer injected into any of the C_{old} resources after C_{new} resources are used, C_{old} resources eventually become free and can be incorporated into the set of C_{new} resources once completely empty, nullifying R_{old} (i.e., R_{old} now supplies the null set).

In order for Double Scheme dynamic reconfiguration to be applied to a network, support for the following must exist: (a) support for subjecting some packets to one routing function (or routing subfunction) and other packets to a different routing (sub)function throughout packet lifetime in the network; (b) support for initiating, detecting and notifying drainage of resource sets in the network and network interfaces; and (c) support for changing (updating) the prevailing routing function across the network and network interfaces. For optimization purposes, there should also be support for segregating and re-integrating connected subsets of resources from a unified set so that resources can be used efficiently during the common state of no network reconfiguration. Below, many of the inherent features and mechanisms in InfiniBand Architecture that can be exploited to achieve the above are described. As is shown in Section 3, this allows Double Scheme dynamic reconfiguration to be successfully applied to IBA-compliant networks.

2.2 Exploitable Features of InfiniBand Architecture

InfiniBand is a layered network architecture that employs switch-based, point-to-point links for the interconnect fabric. An IBA network is composed of one or more sub-networks (subnets) through which communication is done using *routers*. A subnet is the smallest functional composition of IBA-compliant components which can operate independently. End-nodes within a subnet are interconnected through *switches*, and each end-node has one or more *channel adapters* (CAs)

attached directly to it. These CAs serve as the source and terminus of IBA packets. Each subnet is managed autonomously by an associated subnet manager (SM). It is responsible for discovery, configuration and maintenance of components associated with a particular subnet. Only one master SM is active at any one time for each subnet, but passive subnet management agents (SMAs) residing in IBA network components are used to communicate with the master SM through a set of well-defined protocols, referred to as the subnet management interface (SMI).

Routing in IBA is source-based but implemented in a distributed manner using forwarding tables residing in each switch. Each CA or switch/router port has a globally unique identifier (GUID)—a physical name—but can have up to 128 local identifiers (LIDs)—a logical name—associated with it. A 3-bit link mask control (LMC) value can be used to map multiple LIDs to the same GUID port. LIDs in the range of BaseLID to BaseLID+$2^{LMC} - 1$ map to the same port to allow destination renaming [11]. For instance, if a CA port has a LMC value of 3 and a hex base address of 0x0010, then addresses 0x0010 to 0x0017 all map to the same physical destination port. Mapping of GUIDs to LIDs allows components in the network to persistently identify other components either logically or physically. How and where this mapping is done is not specified; it is assumed that the SM (possibly with the help of SMAs) can maintain this mapping function. Since there is a unique LID entry corresponding to each address in the forwarding tables as described below, multiple logical addresses pointing to the same physical port can be used to implement source adaptive routing—a technique that allows packets to reach destinations using different paths in the network.

IBA allows packets to be distinguished by service class into one of sixteen different service levels (SLs). The service level of a packet is contained in a 4-bit field in the local routing header (LRH) of the packet used by switches as the packet traverses the subnet. IBA also allows packets to traverse the physical links of a network using different virtual lanes. A virtual lane is a representation of a set of transmit and receive buffers on a link. Up to sixteen virtual lanes (VL0-VL15) are allowed, but a minimum of two (VL0,VL15) are required by all ports. VL15 is used only for control packets, whereas the other virtual lanes (VL0-VL14) are used only for data packets. Virtual lane assignment exists only between ports at each end of a link, and virtual lane assignment on one link is independent of the assignment on another link, given by a 4-bit VL field in the LRH of a packet.

The actual number of data virtual lanes used by ports and how packets of a given service level map to virtual lanes is determined by the SM at the time that the network is configured

or recon£gured. The VL assignment of a packet is not necessarily the same across a subnet, possibly composed of heterogeneous components. Packets of a given service level may need to be switched between different VLs as they traverse the network. Service level to virtual lane (SL-to-VL) mapping is used to change the VL assignment of packets as they traverse a subnet. The SM is responsible for con£guring a 16 entry, 4-bit wide SL-to-VL mapping table associated with each CA or switch port in which each entry indicates the VL to be used for the corresponding SL. The SL-to-VL mapping table can be read and modi£ed by the SM through subnet management methods SubnGet() and SubnSet(). This allows, among other things, the possibility of two packets with the same SL and VL arriving from different input ports yet destined to the same output port to be assigned different VLs.

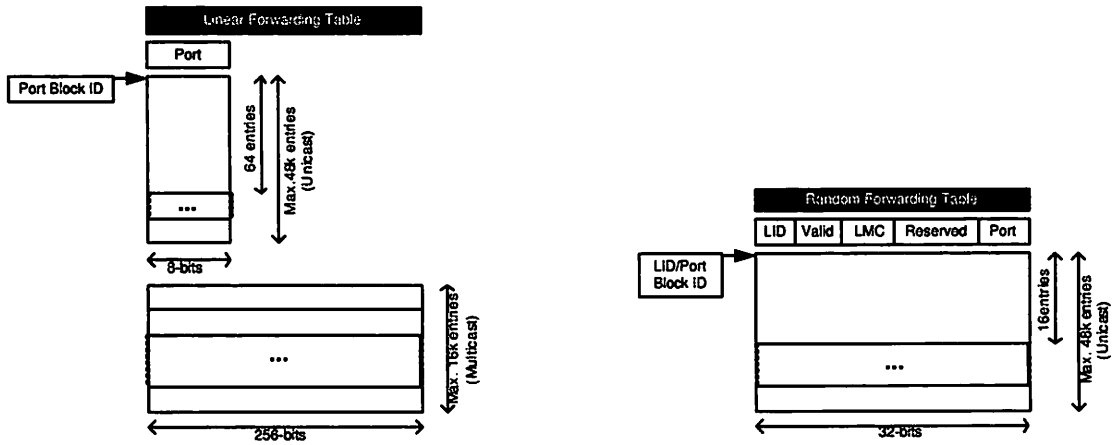

Figure 1. Linear and Random Forwarding Tables in In£niBand.

In addition to the SL-to-VL mapping tables, the SM is also responsible for con£guring forwarding tables in CAs and switches. Routing functions in IBA are implemented explicitly through forwarding tables using LIDs. Alternatively, a special mechanism for exchange of control packets between the SM entities could also be used. This mechanism, called the directed routes, allows the sender to specify the complete path that the packet must take from the source node to the destination node and back. The directed routes mechanism also allows packets to be routed using the normal LID routing on either side of the directed route. Forwarding tables are composed of a set of entries addressed by the LID of a packet's LRH such that a matching entry speci£es the output port that should be used by the packet. They can be organized as linear or as random forwarding tables, as shown in Figure 1.

Linear forwarding table (LFT) entries are con£gured by the SM through an attribute modi£er.

This modifier is a pointer to a list of 64 forwarding table entries or port block elements, where each entry or element is an 8-bit port identifier to which packets with LIDs corresponding to this entry are forwarded. All entries in a LFT are in sequential (linear) order starting from the address specified in the attribute modifier. The level of granularity at which the LFT can be modified is one block, i.e., 64 entries. Assuming the SM can do a read-modify-write on each block, entries within a particular block would be unavailable for lookup only during the time of writing back the block, in the worst case. Random forwarding tables (RFTs) provide greater flexibility since a finer granularity is used for table modification. The attribute modifier for RFTs points to a block of only 16 LID/Port block elements to which this attribute applies. Also, unlike LFTs, consecutive entries in RFTs are not necessarily in sequential addressing order. This flexibility comes at the cost of more complex implementation and higher access time, as RFTs may require implementation using content addressable memories.

The virtual interface between IBA hardware and an IBA consumer process is the send and receive queue pair (QP). Each port can have up to 2^{24} QPs which are operationally independent from one another. Connection-oriented service types bind QPs at the sending and receiving ends whereas datagram (connectionless) service types target QPs at the receiving end by specifying the QP number along with the target CA's port LID. QPs are not directly accessible to consumer processes; instead, consumer processes use "verbs" to submit work requests (WRs) to a send queue or a receive queue. The CA processes this request and places it on the respective queue. QPs can be created by invoking the CreateQP verb with a set of initial attributes. IBA allows CAs to pre-allocate QPs or allocate them in response to a request for communication. For simplicity, this paper assumes that queue pairs are allocated only when a request for communication is received.

When a consumer wants to establish a path, it queries the SM with a "PathRecord" request. The initiator may specify the destination LID (DLID) in the request message, in which case the SM will return only the information related to the path or paths to the specified destination. Alternatively, it may not specify the DLID value, in which case the SM will return information for all the ports that are reachable from the source. Having received path record information, the initiator sends a request message to the target. The requested service type's service level is placed in the request message. Should the request be accepted, the target responds with a response message containing the QP number (for connection oriented services) or end-to-end context (for reliable and unreliable datagram services). The target may reject the request by sending a reject message or the target may specify a different set of variables on which communication can be done through

a response message. The communication establishment sequence is completed when the initiator sends a "Ready to Use" message to the target.

3 Applying the Double Scheme to InfiniBand Architecture

The support necessary to implement the Double Scheme over InfiniBand was mentioned at the end of Section 2.1. Here, specific mechanisms introduced in Section 2.2 that can be used to provide that support and how those mechanism should be used to implement the Double Scheme is described.

The first requirement can be supported by assigning two sets of LID addresses for each GUID. As routing in IBA is source-based and dependent on LID addresses, this allows two different routing functions to route packets, one using one set of LIDs and the other using the other set. This, in effect, means that only half of the total possible number of LIDs and routing table entries are useable during normal operation, which should not typically be a problem. It is not necessary to divide LIDs equally among the routing functions, but this may be preferred to allow source adaptivity in both the routing functions. In the extreme case, 127 out of the maximum of 128 allowed LIDs per port may be used by one routing function and only one by the other. In that case, the second routing function which has only one available destination LID will not have any source adaptivity.

The second requirement can be supported by allowing only half (or any restricted number of) the SLs to be available to packets at any given time outside of reconfiguration. During reconfiguration, when the both routing functions exist in the network simultaneously, packets under the influence of one routing function use one set of SLs while packets under the influence of the other use the other set of SLs.[1] During normal operation, these SLs can be mapped to all the available VLs, allowing the optimization mentioned in Section 2.1 to be supported as well. Given the continuing reductions in integration costs, it is likely that most IBA-compliant devices will support multiple data virtual lanes. During reconfiguration, the SM can modify the SL-to-VL mapping to allow a set of VLs to be drained. The SM can also initiate a "Send Queue Drain" to drain QPs [3]. The drainage state of VLs and QPs can be tracked and notified by the SM.

Finally, the third requirement can be supported by having the SM perform forwarding table updates and PathRecord modifications to allow packets to be sent using the alternative set of LIDs.

[1] It is expected that this would not cause any significant QoS degradation as most implementations are likely not to use all sixteen service levels at once.

By exploiting these IBA features and mechanisms, Double Scheme dynamic reconfiguration can be accomplished with a sequence of steps, as presented below.

3.1 Proposed Reconfiguration Procedure

1. **Initiate reconfiguration:** The need for reconfiguration is established by the SM. Reconfiguration could be triggered by a physical link being down, which is either detected the SMA in a neighbor switch and notified to the SM or is identified directly by the SM during network exploration. Exactly how this is done is beyond the scope of this study. We, therefore, assume that the SM is notified for the need to reconfigure by some IBA supported mechanism. Packets which cannot be routed due to physical disconnectivity are discarded using, for instance, IBA's packet timeout mechanism.

2. **Modify SL-to-VL Mapping and Update Forwarding Tables:** The SM (possibly with the assistance of SMAs) reads the SL-to-VL mapping tables from each port of a CA or switch and modifies them such that the set of SLs that is currently being used by the packets map to only half the VLs (or any restricted number of VLs between 1 and 14). The basic idea is to drain at least one VL for packets that would be using the new routing function. Subnet management packets continue to use VL15.

   In parallel with this, the SM updates forwarding table entries at the switches and CAs that correspond to the LID addresses used for the new routing function. This can be done using a process similar to that used during network initialization. If forwarding is implemented using RFTs, updates can be done without obstructing current routing operations since the old and the new routing functions can be implemented on two independent sets of LID/port blocks, as shown in Figure 1. If, however, LFTs are implemented, the SM will have to do a read-modify-write on each port block that needs to be modified. Packets may not be able to be forwarded concurrently with the update if their destination LIDs lie in the block being written back. This presents a tradeoff between using more complex RFTs that can be modified transparently and using simpler to implement LFTs whose port blocks may become unavailable for a short period of time during reconfiguration.

3. **Detect VL Drainage:** Before the routing information at the injection nodes can be updated, the VLs to be used by the new routing function have to be drained. The drainage algorithm we propose is described in Section 3.2. That algorithm is generally applicable to any deter-

ministic routing function as drainage is based solely on channel dependency properties of the new routing function.

4. **Modify PathRecord and GUID-to-LID Mapping Information:** Once all the forwarding tables have been updated and VLs for the new routing function have been drained, the SM modifies the PathRecord information for each node such that it now supplies the set of SLs that were previously unused. By doing this, the SM will ensure that any new packets being injected into the network use only the VLs that are reserved for them (i.e., VLs that are not being used by packets already in the network and using the old routing function). Notice that by changing the PathRecord information, the SM will force all newly formed QPs to comply with the new set of SLs; QPs which had been formed earlier and contained messages with old SLs will have to be drained using the "Send Queue Drain" mechanism invoked by the SM.

   In parallel with this is the modification of the GUID-to-LID mapping by the SM. The addresses which were previously unused (but within the valid range of Base LID+2^{LMC} for each port) are now supplied. Recall that the new routing function is implemented in the forwarding tables on this set of LIDs. So, supplying these addresses as destination addresses essentially means that packets will now be routed using the new routing function. It is important that the modification of the PathRecord and GUID-to-LID information be performed in synchronism at a particular node so that newly injected packets with the LIDs corresponding to the new routing function are placed in Send queues with the appropriate set of SL. However, these modifications may be performed asynchronously over the network.

5. **Detect VL Drainage and Restore the SL-to-VL mapping:** Once the network has been drained of packets using the old routing function, the SM can restore the SL-to-VL mapping tables at all nodes such that they now provide all available VLs to packets using the new routing function. The same drainage algorithm described in Step 3 can be used, and a similar process as described in Step 2 can be used to modify SL-to-VL mapping.

3.2 Algorithm to Detect VL Drainage

By modifying the SL-to-VL mapping such that the SL-to-VL mapping tables allow use of only a restricted set of VLs for packets with the SLs associated with the old routing functions, VLs for the new routing function can be drained in a single hop. However, before packets using the new

routing function can be allowed to use these VLs, complete drainage of these VLs across the entire subnet must be guaranteed. This is because the actions of the steps given previously need not be carried out synchronously across the entire network. A particular node can maintain the state of buffers at its input and output ports and, thus, detect local drainage of VLs, but it has no way of knowing whether or not it will receive more packets on these VLs from its neighboring nodes. There needs to be some form of synchronization between the nodes in order to detect drainage across the entire subnet.

Presented here is a simple yet general algorithm that can be used to detect virtual lane (or channel) drainage across the network. The algorithm uses the channel dependency information available in the deadlock-free routing function to be implemented in order to determine which channels must be drained. This information is represented in the form of a directed graph, which encodes the dependencies between the channels as allowed by the routing function. By systematically collecting channel drainage information at individual nodes along this dependency graph, channel drainage across the entire network for that particular routing function can be guaranteed.

The key data structure in this algorithm is the channel dependency graph (CDG) [16], which gives dependency relations between different channels. A CDG is simply a directed graph in which vertices (or nodes) of the graph are channels connecting the various routing nodes of the network. Each bidirectional channel is represented as two independent nodes in the graph. Arcs in the CDG represent the dependencies between channels. For example, an arc from channel c_i to c_j indicates that a packet can request c_j while holding resources associated with c_i. Also, the properties of the CDG assert that in order for a deterministic routing function to be deadlock free, the CDG must be acyclic [16].

The drainage algorithm can be implemented in an IBA subnet with the following steps:

1. The SM computes the CDG for the routing function to be implemented. IBA's source-based routing is deterministic; therefore, the CDG must be cycle-free in order for the routing function being implemented to be deadlock-free.

2. Having computed the CDG, the SM sends control packets to the switches connected to all the source channels (i.e., source *nodes*, from the standpoint of the CDG).

3. Every switch forwards control packets along all the directions indicated by arcs in the CDG, when no more packets remain in the VLs that are to be drained. These control packets are transmitted through VL15, not through the data VLs to be drained.

4. When a leaf node of the CDG receives this control packet, it forwards it to the SM.

5. When the SM has received control packets that have traversed all the legal routes as determined by the CDG, the set of channels is guaranteed to be drained.

The SM has knowledge of the routing function to be implemented, and, therefore, it can compute the CDG. From an implementation standpoint, the key issue is that of sending the control packets by the SM to the hosts connected to switches at the originating end of the source channels in the CDG and then collecting them from the leaf channels. These packets have to traverse all the legal paths in the CDG before they reach the leaf channels, which forward them to the SM. Since, only the SM is assumed to have knowledge of the CDG, a combination of LID-based and directed routing could used be to accomplish this task. The number of control packets sent to these source channels and gathered by the SM is equal to the total number of possible paths to leaf channels accessible by source channels.

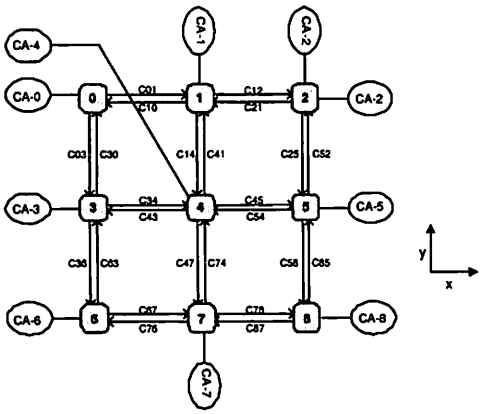

Figure 2. An example IBA subnet comprising a 2D mesh topology.

4 Discussion

As an example, let us consider an IBA subnet with nine switches connected in a 2-D mesh topology. Each switch connects to only one channel adaptor, as shown in Figure 2. For simplicity, let us assume that SL1 through SL8 are allocated to the current routing function, while the remaining SLs are reserved for the new one. Also, let the number of data VLs per physical channel across the subnet be equal to four. To illustrate the various steps of the recon£guration process, we will assume that the source-based deterministic routing function implemented on this network

has to be reconfigured from XY-routing to YX-routing. Both these routing functions are deadlock free independently.

Notice that both routing algorithms are defined on a $C \times N$ domain [16], i.e., these routing functions take into account the input channel and the destination node to compute the output channel of a packet in the network. In IBA, forwarding is defined on an $N \times N$ domain [9] because the forwarding tables consider only the current and destination nodes of a packet to determine its output port (recall that forwarding tables in IBA are implemented on a per node basis rather than per port basis). In a $C \times N$ based routing algorithm, if the incoming port is not considered while making the routing decision, routing rules cannot be enforced, and, hence, deadlock-freedom cannot be guaranteed. However, previous work reported in [12] has shown that $C \times N$ based routing algorithms can be implemented on IBA by use of destination renaming. The basic idea is the following. Given any $C \times N \rightarrow C$ routing table, when there is some switch that supplies two different paths for the packets arriving at different input channels but destined for the same node/host, the destination of one of them is modified, selecting an unused address within the valid range of addresses assigned to that destination node/host. As the destination addresses of these packets are now different from the point of view of the switches within the subnet, they can be routed along the different paths without considering the input channel. This technique undoubtedly will have an impact on the size of the forwarding tables (and consequently, the time for the table lookup), and the maximum number of renamings that can be done is limited by the maximum available number of addresses for each host. For the implementation of the Double Scheme that we are proposing, each host must have at least one LID address reserved for the new routing function during the time reconfiguration of the network is in progress. From this point onwards, we shall refer to all the addresses associated with a host that are being used by the old or the new routing function as including the addresses required for renaming. Note that this reservation of addresses for renaming is necessary only for those cases where the routing function to be implemented in defined on $C \times N$ domain.

At the start of the reconfiguration process, the SM residing on CA-6, for instance, establishes a need for reconfiguration of the subnet. The SM reads in the SL-to-VL mapping table from each CA and switch, and modifies it such that SL1 through SL8, which are the SLs being used by current routing function, map to only two of the four available data virtual lanes at each channel, i.e., VL0 and VL1. Once the modification has been done, the tables are written back to their respective ports. Concurrently, the SM starts updating the forwarding tables at the switches. In the case of the

RFTs, the SM reads in the LID/Port blocks that are not being used by the current routing function, modifies them to include entries corresponding to the new routing function, validates these entries and then writes back the modified block to their respective tables, as illustrated in Figure 3.

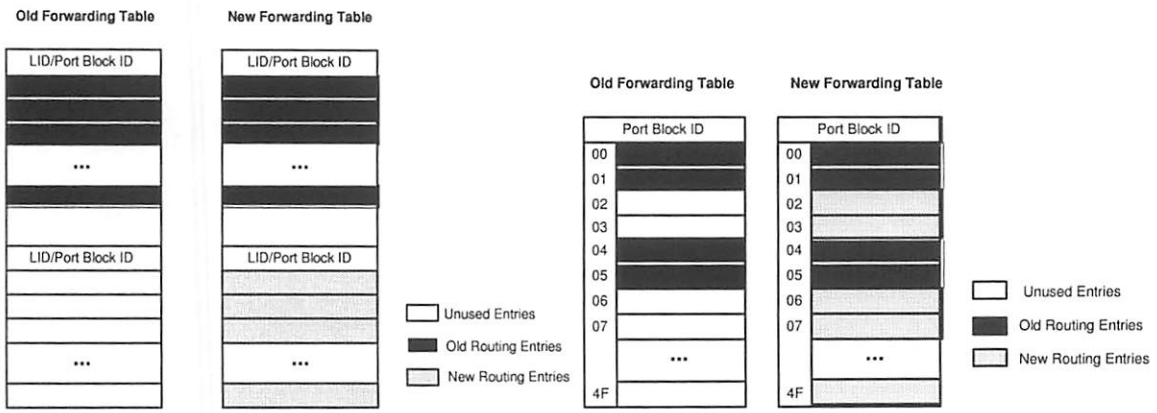

Figure 3. Old and new instantiations of a Random Forwarding Table and a Linear Forwarding Table.

In the case of LFTs, each 64-entry port block may contain addresses corresponding to both the current and the new routing functions, and thus has to be modified. A port block is unavailable at most during the time that the SM writes back the modified block to the forwarding table. Structure of a LFT before and after reconfiguration is also illustrated in Figure 3.

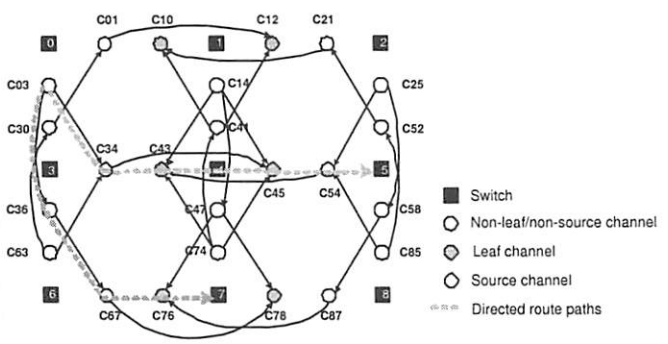

Figure 4. Channel Dependency Graph (CDG) for the YX-routing function. Nodes in the CDG are channels in the network.

Next, the SM starts the process of detecting the drainage of VL2 and VL3 across the subnet. Following the algorithm detailed in Section 3.2, the SM begins by computing the CDG for the

new routing function, i.e., the YX-routing function, shown in the Figure 4. The SM sends control packets to the CAs connected to the switches at the originating ends of the source channels in the CDG. The forward path for each of these packets could be determined by LID routing, however, the return path is set by the SM using directed routes. The SM sends a control packet to the source channels for each path to leaf channels that can be established from that particular source channel. For instance, CA-0 will receive two packets, each of which will be destined for CA-5 and CA-8 using directed routes and forwarded on to the SM. The CAs in the path of these directed route control packets will forward each packet to the next node once VL2 and VL3 are drained at their input and output ports. Finally, the SM will receive control packets from each leaf channel's destination CA corresponding to each path that terminates at this leaf channel. This is to ensure that all the channels accessible through CA-0 are drained of packets using the old routing function when the SM receives the two control packets back. Figure 4 shows the path followed by the two control packets sent by the SM to CA-0. VL2 and VL3 at each of the channels on this path are drained of old packets when the SM receives the appropriate number of packets back.

Once the VL2 and VL3 have been drained and forwarding tables have been updated, the SM *atomically* updates the PathRecord and GUID-to-LID mapping information corresponding to each CA in the subnet although multiple ports may be modified in any order. This information primarily resides with the SM, however, the CAs may have 'cached' the information depending on the particular implementation. In that case, the update has to be done at each CA in the network. At this point, the CAs start injecting packets which are routed using the new routing function. Old packets may still be routing in the network using the old routing function (i.e., old LIDs) but are spatially separated from packets using the new routing function (i.e., new LIDs).

Finally, the SM uses the same drainage detection algorithm as it did before in order to detect drainage of VL0 and VL1. Notice that the SM will use the CDG for XY-routing function, as shown in Figure 5, to detect this drainage. Once complete drainage of old packets has been detected, SL-to-VL mapping at each switch is updated to allow packets with SL9 through SL14 to use all the four VLs. At this point there are no more old packets present in the network, and reconfiguration of the subnet is completed.

5 Conclusion

This paper prescribes a systematic method for applying the Double Scheme to IBA networks which allows deadlock-free dynamic reconfiguration of the network. Three key challenges for

Figure 5. Channel Dependency Graph (CDG) for the XY-routing function. Nodes in the CDG are channels in the network.

implementing the Double Scheme over IBA networks were identified. A number of IBA features and mechanisms that address these challenges and how they should be used are also described. It is shown that spatial and/or temporal separation of resources—which is the basic idea behind the Double Scheme—can be accomplished in an IBA subnet by distinguishing sets of service levels and destination local identifiers used to route packets in the network. Drainage of resources can be accomplished under the direction of the subnet management using various methods and attributes. An algorithm is proposed that uses only IBA mechanisms to accomplish selective resource drainage. It is also shown that dynamic update of forwarding tables and destination names is also supported by IBA in a manner consistent with that needed for the Double Scheme. As a result, this work enables InfiniBand networks to better support applications requiring certain QoS guarantees that would not well tolerate intermittant performance drops-offs as would be the case without deadlock-free dynamic reconfigurabilty. Interesting future work could consist of performing more detailed cost/performance analysis of the Double Scheme applied to InfiniBand Architecture, such as through simulation or through analytical modeling.

References

[1] Ruoming Pang, Timothy Mark Pinkston, and Jose Duato. The Double Scheme: Deadlock-free Dynamic Reconfiguration of Cut-Through Networks. In *The 2000 International Conference on Parallel Processing*, pages 439–448. IEEE Computer Society, August 2000.

[2] R. Buyya. *High performance cluster computing.* Prentice-Hall, 1999.

[3] Gregory F. Pfister. An introduction to the infiniband architecture. In *Proceedings of the Cluster Computing Conference (Cluster00) Chapter 57*, November 2000.

[4] Charles Spurgeon. *Quick Reference Guide to the Ethernet System* . http://wwwhost.ots.utexas.edu/ethernet/descript-100quickref.html, 1995.

[5] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. Autonet: A High-Speed, Self-Configuring Local Area Network Unsing Point-to-Point Links. *IEEE Journal on Selected Areas in Communication*, 9(8):1318–1335, October 1991.

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

[7] et al. Kumar Malavalli. Fibre Channel Switch Fabric-2 (FC-SW-2). *NCITS 321-200x T11/Project 1305-D/Rev 4.3 Specification*, pages 57–74, March 2000.

[8] *InfiniBand^{TM} Architecture Specification Volume 1, Release 1.0.* InfiniBand Trande Association, October 24, 2000.

[9] J. Duato. A New Theory of Deadlock-free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[10] J. Duato. A Necessary and Sufficient Condition for Deadlock-free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, October 1995.

[11] Jose Carlos Sancho, Antonio Robles, and Jose Duato. Effective Strategy to Compute Forwarding Tables for InfiniBand Networks. In *Proceedings of the International Conference on Parallel Processing*, pages 48–57. IEEE Computer Society Press, September 2001.

[12] Pedro Lopez, Jose Flich, and Jose Duato. Deadlock-free Routing in InfiniBand through Destination Renaming. In *Proceedings of the International Conference on Parallel Processing*, pages 427–434. IEEE Computer Society Press, September 2001.

[13] F.J. Quiles J.L.Sanchez R. Casado, A. Bermudez and J.Duato. Performance evaluation of Dynamic reconfiguration in High Speed Local Area Networks. In *Proceedings of the 6th Symposium on High-Performance Computer Architecture*, pages 85–96, Jan 2000.

[14] Dimiter Avresky. *Dependable Network Computing,* Chapter 10. Kluwer Academic Publishers, 2000.

[15] F.J.Quiles J.L. Sanchez R.Casado, A.Bermudez and J.Duato. A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks. *Special Issue on Dependable Network Computing. IEEE Transactions on Parallel and Distributed Systesm*, 12(2):115–132, February 2001.

[16] W. Dally and C. Seitz. Deadlock-free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.