Deadlock-free Routing in InfiniBand™ through Destination Renaming*

P. López, J. Flich and J. Duato
Dept. of Computing Engineering (DISCA)
Universidad Politécnica de Valencia, Valencia, Spain
plopez@gap.upv.es

Abstract

The InfiniBand Architecture (IBA) defines a switch-based network with point-to-point links that supports any topology defined by the user, including irregular ones, in order to provide flexibility and incremental expansion capability. Routing in IBA is distributed, based on forwarding tables, and only considers the packet destination ID for routing within subnets in order to drastically reduce forwarding table size. Unfortunately, the forwarding tables for most of the previously proposed routing algorithms for irregular topologies consider both the destination ID and the input channel. Therefore, these popular routing algorithms for irregular topologies may not be usable in InfiniBand networks because they do not conform to the IBA specifications.

In this paper, we propose an easy-to-implement strategy to adapt the forwarding tables already computed following any routing algorithm that considers the destination ID and the input channel into the required IBA forwarding table format. The resulting routing algorithm is deadlock-free on IBA. Indeed, the originally computed paths are not modified at all. Hence, the proposed strategy does not degrade performance with respect to the original routing scheme.

1. Introduction

InfiniBand [10] has been recently proposed as a standard for communication between processing nodes and I/O devices as well for interprocessor communication. More than 180 companies, including the leading computer manufacturers, support the InfiniBand initiative. The InfiniBand Architecture (IBA) is designed around a switch-based interconnect technology with high-speed point-to-point links. An IBA network is composed of several subnets interconnected by routers, each subnet consisting of one or more switches, processing nodes and I/O devices. Nodes are directly attached to a switch through a Channel Adapter (CA).

Routing in IBA subnets is distributed, based on forwarding tables (routing tables) stored in each switch, and only considers the packet destination node for routing [11]. Neither the input link nor the input buffer (virtual lane) containing the packet header are considered when addressing the forwarding table. By considering only the packet destination node, the size of the forwarding table is drastically reduced. It should be noted that high-degree switches (e.g., 64-port switches) are likely to appear in the near future, and IBA supports up to 16 virtual lanes. If both the input link and the virtual lane through which the packet arrived were considered when addressing the forwarding table, its size would increase by a factor of 1024, also increasing table lookup time. This latter kind of routing is known as $C \times N \rightarrow C$[1] [3] as opposed to the IBA one, which considers both the current and the packet destination nodes, and is known as $N \times N \rightarrow C$ [4].

IBA supports any topology defined by the user, in order to provide flexibility and incremental expansion capability. Thus, the obtained topology can be completely irregular, being necessary to determine the network topology and compute the forwarding tables for all the switches. In particular, a generic routing algorithm, suitable for any topology that can be implemented by the user, is required. Up*/down* [13, 1] is the most popular routing algorithm satisfying this property. However, up*/down* routing can not be used in IBA subnets because it is a $C \times N \rightarrow C$ routing algorithm. If the incoming port number is not considered, the up*/down* rule can not be locally enforced at each switch and deadlock-freedom can not be guaranteed.

In [5], a solution to this problem has been proposed. It is based on removing, on those switches where there exist paths starting with both "up" and "down" channels to reach a given destination, all the routing options starting with an "up" channel. Hence, the possibility of a "down" $\rightarrow$ "up" transition is eliminated. However, this simple approach has two drawbacks. First, it is only intended for up*/down*-based routing. However, there are other routing algorithms, like smart-routing [2], that are neither based on spanning

*This work was supported by the Spanish CICYT under Grants TIC2000-1151-C07 and 1FD97-2129.

[1] C and N are the sets of network links and nodes, respectively.

trees nor on assigning direction labels to channels but also rely on the input channel to route packets through the network. Indeed, smart-routing outperforms up*/down* routing and its variants [7]. Second, the approach proposed in [5] sacrifices some valid shortest routes, increasing the average network distance, thus potentially degrading network performance.

In this paper, we present another approach to implement any $C \times N \rightarrow C$ routing algorithm on IBA networks. The methodology proposed in this paper is not restricted to a particular routing algorithm. Moreover, it does not affect the performance achieved by the original routing algorithm.

The rest of the paper is organized as follows. Section 2 presents in detail the required background and the motivation for the paper. Section 3 describes the technique to support $C \times N \rightarrow C$ routing algorithms on IBA networks, including some implementation issues. In Section 4 we analyze the feasibility of the proposal through some experiments. Finally, some conclusions will be drawn.

2. Background and Motivation

Routing in irregular networks can be based on source or distributed routing. In the former, the packet header contains the sequence of ports to be crossed along the path to the destination [1]. In the latter, each switch has a forwarding table that stores the output ports that can be taken by the incoming packet. In both cases, a routing algorithm must determine the path to be followed. Several deadlock-free routing schemes have been proposed for irregular networks [13, 1, 8, 2, 6, 14, 12], up*/down* routing being the most popular one.

2.1. Up*/down* and its Implementation on IBA

Up*/down* routing is a generic routing algorithm suitable for switch-based networks with regular or irregular topology, and can be implemented using both source or distributed routing.

We will describe here the original up*/down* routing algorithm proposed for DEC Autonet [13]. It is based on an assignment of direction labels to links. To do so, a breadth-first spanning tree (BFS) is computed and then, the "up" end of each link is defined as: (1) the end whose switch is closer to the root in the spanning tree; (2) the end whose switch has the lower ID, if both ends are at switches at the same tree level. As a result, each cycle in the network has at least one link in the "up" direction and one link in the "down" direction. Thus, cycles in the channel dependence graph (CDG) [3] are avoided by prohibiting packets to traverse links in the "up" direction after having traversed one in the "down" direction. Figure 1 shows an example network and the direction assigned

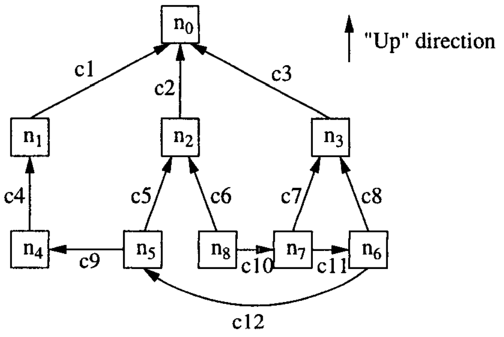

Figure 1. Up*/down* routing on an example network

to its channels. For the sake of simplicity, we have only shown the switches, removing processing nodes that are attached to them. Valid routes are the ones that do not cross "up" after "down" channels. For instance, to forward packets between hosts connected to switches $n4$ and $n2$, the non-minimal path through channels $c4(up) \rightarrow c1(up) \rightarrow c2(down)$ will be used; also between hosts connected to switches $n4$ and $n8$, the non-minimal path through channels $c9(down) \rightarrow c12(down) \rightarrow c11(down) \rightarrow c10(down)$ will be used. Note that the minimal path $c9(down) \rightarrow c5(up) \rightarrow c6(down)$ is not valid because it contains a "down" to "up" transition. However, $c5(up) \rightarrow c6(down)$ is a valid path to forward packets generated at switch $n5$ destined for a host attached to $n8$.

When a packet arrives at a switch, the destination address stored in its header is used, concatenated with the incoming port number, to index the forwarding table of the switch. This table returns the outgoing port number the packet must be routed through. In case of several suitable outgoing ports, the forwarding table may return all of them, and the arbiter decides which one to use. We will not consider this issue. Table 1 shows some entries of the forwarding tables for the network shown in Figure 1, for the path examples given above. Actually, the tables contain processing node IDs rather than switch IDs. However, for the sake of simplicity, assume that there is exactly one host attached to each switch, whose ID is the same as the one for the switch it is connected to. *Local* represents the port used by the host attached to that switch.

As we can see, the forwarding tables use the input channel to distinguish the path that must be used. For instance, at switch $n5$, for the same destination $n8$, there are two alternative output channels $c12$ and $c5$. To select the correct one, the input channel must be considered.

However, IBA does not consider the input channel used by the message to access the forwarding tables. As stated in [5], the trivial solution consisting of removing all the entries in the forwarding table for a given destination but the ones corresponding to packets injected at the current switch (i.e.,

$n0$			$n1$			$n2$			$n4$		
IC	D	OC	IC	D	OC	IC	D	OC	IC	D	OC
$c1$	$n2$	$c2$	$c4$	$n2$	$c1$	$c2$	$n2$	*local*	*local*	$n2$	$c4$
...	...	...	...	...	...	$c5$	$n8$	$c6$	*local*	$n8$	$c9$
...	...	...	...	...	...	...	...	...	...	...	...

$n5$			$n6$			$n7$			$n8$		
IC	D	OC	IC	D	OC	IC	D	OC	IC	D	OC
$c9$	$n8$	$c12$	$c12$	$n8$	$c11$	$c11$	$n8$	$c10$	$c10$	$n8$	*local*
local	$n8$	$c5$	...	...	...	...	...	...	$c6$	$n8$	*local*
...	...	...	...	...	...	...	...	...	...	...	...

Table 1. Forwarding tables for the network shown in Figure 1. IC: Input channel; D: Destination; OC: Output channel.

in switch $n5$, all packets destined for $n8$ will choose $c5$) does not work at all. For example, a packet coming from $n4$ through $c9(down)$ would leave $n5$ using $c5(up)$, breaking the up*/down* rule.

The solution proposed in [5] fills in the forwarding table at each switch with the original up*/down* forwarding table for packets injected at that switch without removing any legal up*/down* path (i.e., including shortest and non-shortest paths). Additionally, when the switch offers alternative paths to reach a given destination and some of them start with an "up" channel while other paths start with a "down" channel, it removes all the routing options starting with an "up" channel. Hence, in the routes that cross that switch, only $local \rightarrow down$ or $up \rightarrow down$ transitions are allowed, which are deadlock-free. In our example, at switch $n5$, all packets destined for $n8$ must use the path through $c12$. The main drawback of this approach is that some paths are penalized. In our example, packets generated at switch $n5$ destined for $n8$ must now use the non-minimal path through $c12$, despite there is a minimal path through $c5$.

This problem will occur on every switch connected to two or more "up" channels and, at least, one "down" channel. If, for any destination, there are several paths and some of them use $up \rightarrow up$ or $local \rightarrow up$ while others use $down \rightarrow down$ at this switch, the problem will appear. This is the case for $n5$ in Figure 1.

Although we have used up*/down* routing to illustrate the problem, it may happen when generating the IBA forwarding tables corresponding to any $C \times N \rightarrow C$ routing algorithm. When, at some switches, there are alternative paths to reach some destination that must be selected according to the input channel, the problem will arise.

2.2. A Particular Feature of IBA Addressing

In this section, we will briefly describe a feature of IBA addressing which our proposal is based on.

An IBA network is designed around a point-to-point switched I/O fabric, whereby end node devices (from inexpensive I/O devices to complex host computers) are interconnected by cascaded switch devices. Each end node contains one or more channel adapters (referred to as *HCA* in the case of processor nodes). Each channel adapter contains one or more ports. Each port has a local identifier (LID) assigned by the local subnet manager, which is unique within the subnet. The LID is used by the subnet (i.e., the forwarding table at each switch) to forward packets towards their destination (the destination local identifier or $DLID$). However, IBA provides multiple virtual ports within a physical port by defining a *LID Mask Control* or LMC [11]. The LMC specifies the number of least significant bits of the LID that a physical port masks (ignores) when it validates that a packet $DLID$ matches its assigned LID. As these bits are not ignored by the switches, from the subnet point of view, each *HCA* port has been assigned not a single address but a valid range of addresses (up to 2^{LMC} consecutive addresses). Moreover, each *HCA* port will accept all packets destined for any valid address within its range. For instance, Table 2 shows the valid address ranges assigned to each *HCA* port for $LMC = 3$. It should be noticed that each *HCA* port can be programmed with its own *LMC* value. For the sake of simplicity, in what follows we will use either the term host or *HCA* to refer to a *HCA* port.

This IBA feature is originally intended for providing alternative routing paths within a subnet. Effectively, as switches do not mask any bit of the packet $DLID$ when forwarding that packet, the subnet may supply different paths for the different addresses of a destination host. The sender host selects one of them by choosing the appropriate $DLID$ address when injecting that packet into the network.

3. Destination Renaming

Our proposal is based on using the virtual addressing feature of IBA to support $C \times N \rightarrow C$ routing algorithms.

Address		*HCA* port
Binary	Dec	
00..0000	0	0 (0.0)
00..0001	1	0 (0.1)
...	...	...
00..0111	7	0 (0.7)
00..1000	8	1 (1.0)
00..1001	9	1 (1.1)
...	...	...
00..1111	15	1 (1.7)
...	...	...

Table 2. Addresses assigned to a given *HCA* port for $LMC = 3$.

$n0$			$n1$			$n2$			$n4$		
IC	D	OC	IC	D	OC	IC	D	OC	IC	D	OC
$c1$	$n2$	$c2$	$c4$	$n2$	$c1$	$c2$	$n2$	*local*	*local*	$n2$	$c4$
...	...	...	...	...	...	$c5$	$n8$	$c6$	*local*	$n8.1$	$c9$
...	...	...	...	...	...	...	...	...	...	...	...

$n5$			$n6$			$n7$			$n8$		
IC	D	OC	IC	D	OC	IC	D	OC	IC	D	OC
$c9$	$n8.1$	$c12$	$c12$	$n8.1$	$c11$	$c11$	$n8.1$	$c10$	$c10$	$n8.1$	*local*
local	$n8$	$c5$	...	...	...	...	...	...	$c6$	$n8$	*local*
...	...	...	...	...	...	...	...	...	...	...	...

Table 3. Forwarding tables after renaming destinations. IC: Input channel; D: Destination; OC: Output channel.

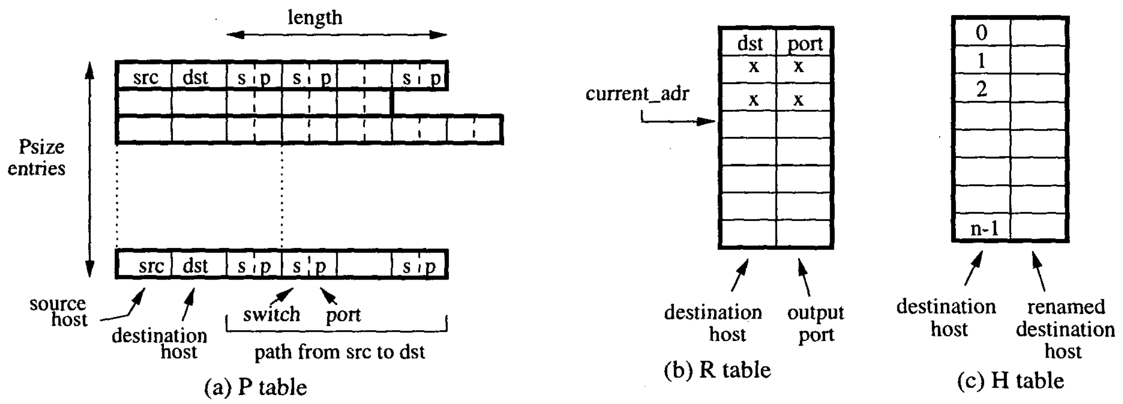

Figure 2. Required data structures to implement the destination renaming technique. P table (a) contains all paths between any pair of hosts, as computed by the $C \times N \to C$ routing algorithm. R table (b) is the forwarding table for each switch. H table (c) is the renaming table for each *HCA*.

The basic idea is the following. Given any deadlock-free $C \times N \to C$ routing tables, when there is some switch that supplies two different paths for packets destined for the same host that arrive at different input channels, the destination of one of them is modified, selecting a valid address in the range assigned to the destination host that is not in use. In other words, we rename the destination for one of the paths. As the destination of both paths is different from the subnet (and the switches) point of view, the correct path can now be selected based only on the *DLID*, without considering the input channel. Of course, the sender host that uses the modified path must now send packets to the renamed destination identifier. Hence, by renaming the destination, we can somehow store some input channel information in the encoding of the destination. Note that destination renaming will have some impact on forwarding table size. Also, there is a maximum number of valid addresses available, which may limit the number of renamings. We will address these issues in Section 4.

Consider again the network example shown in Figure 1 and the forwarding tables shown in Table 1. After applying destination renaming at switch $n5$, the destination $n8$ (actually the host connected to $n8$) in one of the entries will be renamed to $n8.1$. Also, all the forwarding tables that contain the modified path should also be updated. Note that this includes not only the switches traversed before $n5$ but also those traversed after it. Table 3 shows the updated forwarding tables after the renaming. A node ID without suffix (say n) corresponds to its base port address, where the least significant *LMC* bits are zero ($n.0$).

This procedure can be easily generalized to the case where more than two paths attempt to use different output channels for different input links. All we have to do is to check the paths sequentially, comparing at each switch if the path being considered is in conflict with any other one already processed. If it is, then the renaming is performed for the one that it is being examined. Hence, if more than two paths (say x) use different output channels for the same destination and different input channels, several destination renamings may be performed (at most, $x - 1$).

```
procedure compute-forwarding-tables
begin
 for i := 0 to Psize-1 do
   src := P[i].src;
   dst := P[i].dst;
   valid_dst := ValidId(i);

   for j := 0 to P[i].length-1 do
     sw := P[i].s[j];
     port := P[i].p[j];
     R[sw][R[sw].current_adr].dst := valid_dst;
     R[sw][R[sw].current_adr].port := port;
     R[sw].current_adr := R[sw].current_adr + 1;
   end;

   H[src][dst] := valid_dst;
 end;
end.
```

Figure 3. Implementation of the destination renaming technique.

Figure 2 shows the required data structures to implement this strategy. We have assumed that the paths computed by the $C \times N \to C$ routing algorithm are stored in a table (P). If there are n hosts in the network, this table has $n * (n-1)$ entries. Each table entry contains the source and destination hosts and the path between them. This path is stored as a set of pairs switch–outport port for the switch. Table R[s] represents the forwarding table for switch s. Note that it only contains the destination host and the output port that must be used, as required in IBA. Finally, Table H[s] contains the destination renaming information that the host s must use to forward a packet towards its destination.

Figures 3 and 4 show a simple implementation of the proposed technique that does not compute the optimal *LMC* value for each port. This computation has not been included here due to lack of space. The renaming mechanism is implemented as part of the algorithm that must be used to distribute the computed paths into the IBA switches. The compute-forwarding-tables procedure distributes the paths contained in Table P among the forwarding tables R. When it is going to distribute a given source-destination path, before generating a new entry in any forwarding table, it selects a destination identifier that will not generate conflicts in any of the forwarding tables of the traversed switches. In other words, it first searches conflicts for the considered destination and, if any, performs the renaming for it. For this, the ValidId function returns the first valid ID that does not have any conflict along the path. It visits all the switches of the path, checking for conflicts for each possible tested ID. The function finishes once it finds a valid ID with no conflicts. The ValidId function checks a range of virtual destination IDs provided by the First and Last functions, which return the first and last IDs suitable for use for the particular destination being tested, respectively[2]. The first ID checked is the one originally used in the P table. Note that it may happen that the original ID does not encounter any conflict and, therefore, a virtual ID is not needed for a particular path. In fact, this will be the more frequent situation.

```
function ValidId (path#)
begin
 dst := P[path#].dst;

 for valid_id:=First(dst) to Last(dst) do
   IsValidId := true;

   for j := 0 to P[path#].length-1 do
     sw := P[path#].s[j];
     port := P[path#].p[j];

     k := 0;
     found := false;
     while not found and (k<=R[sw].current_adr-1) do
       if ((R[sw][k].dst == valid_id) and
           (R[sw][k].port <> port)) then
         IsValidId := false;
         found := true;
       else
         k := k + 1;
       end;
     end;
   end;

   if (IsValidId) then
     return (valid_id);
   end;
 end;

 HALT; (* Error, virtual ids exhausted *)
END ValidId;
```

Figure 4. Implementation of the destination renaming technique (cont).

Once the ValidId function obtains the valid ID, the compute-forwarding-tables procedure generates all the routing entries in R tables along the path using the valid ID. The H table for the source node is also updated.

Finally, note that the proposed implementation does not take into account whether the switch entries in conflict actually use different input channels. Usually, input channels will differ. However, consider a routing algorithm that has computed the routing tables trying to balance network traffic. In this case, two different paths with the same destination host but different source hosts may pass through the same switch using the same input port and taking different output ports. Again, the only way to distinguish both paths in IBA is by using different destination host addresses, as the proposed implementation does. Thus, this case is also supported by the proposed strategy.

[2]The difference between the values returned by Last and First for a given destination is defined by its current *LMC* value.

4. Evaluation Results

In this section, we analyze the impact of destination renaming on different aspects and parameters of the InfiniBand architecture, assuming that the optimal *LMC* value has been computed for each port. First, we measure the overhead introduced by the renaming technique in the forwarding tables. Next, we evaluate the number of required renamings needed in the worst case (i.e., the most renamed destination) to compute the maximum value for *LMC*. Finally, the potential benefits of the strategy in terms of avoided routing restrictions are studied. This latter study will be performed by measuring the percentage of routing restrictions introduced by the $N \times N \rightarrow C$ InfiniBand specifications which, without destination renaming, either leads to an unimplementable routing algorithm or to a degenerated version of the routing algorithm with higher average distance between hosts [5].

We evaluate different network topologies generated randomly with three restrictions. First, each switch in the network has 4 hosts attached to it. Second, each switch has 4 links to connect to other switches. And third, neighboring switches are connected only by a single link. In order to evaluate the effect of network size on the behavior of the proposed strategy, we also evaluate different network sizes (8, 16, 32, and 64 switches). Finally, in order to obtain meaningful results, we evaluate 10 different random topologies for each network size.

Regarding the routing algorithm used, we evaluate not only up*/down* routing but also smart-routing. In the first case, shortest paths are randomly computed for each source-destination pair. Computed paths are from host to host. In the case of smart-routing, optimized host to host paths are computed by using its load balancing algorithm. In both cases, the resulting paths are in the $C \times N \rightarrow C$ form (i.e., the input port has to be considered at each switch to obtain the output port). Obviously, when we try to use these paths on IBA, some conflicts will arise. Therefore, the renaming strategy will be applied to these sets of paths in order to keep the original routing algorithm.

4.1. Overhead in the Forwarding Tables

IBA defines two forms of switch forwarding tables: linear and random [11]. A given switch supports only one of these forwarding types. The linear forwarding table provides a simple map from *LID* to output port. In other words, the table contains only output ports and the *LID* acts as an index into the table. The size of this table is equal to the number of hosts in the network[3]. On the other hand, random tables are content addressable memories. They are loaded with both *LIDs* and output ports. The table is searched for the packet *LID* and the corresponding output port is returned. The actual size of the table depends on the number of hosts that can be accessed from this switch[4]. As we can see, the destination renaming technique requires a larger table size on both types of forwarding table. In the linear case, as some destination hosts have now several addresses, the total number of hosts in the network increases, and so does the size of the forwarding table.

In the random case, as there are now new entries for the renamed destinations, the size of the table is also increased. With the same number of required destination renamings, the impact on table size is most noticeable with random tables because their initial size (i.e., without renaming) is smaller than for the linear case. For this reason, in what follows, we will analyze the overhead in forwarding table space considering random tables.

Table 4 shows the overhead introduced by the renaming strategy in the forwarding tables for up*/down* and smart-routing, respectively. It shows the average total number of entries required (the sum for all switches), the average total number of original entries (the original *LIDs*, without renaming) as well as the average total number of new entries (additional entries with renamed *LIDs*). The tables also show the minimum, maximum, and average percentage of table size increase due to the added entries. As stated earlier, all results are averaged from 10 random topologies.

As can be seen, the destination renaming technique requires some increase in forwarding table size. The average percentage of increase is around 60% for up*/down* and 40% for smart-routing. Note that the optimized routes computed by smart-routing involve a lower increase in forwarding table size. Anyway, both cases seem to be implementable, as the InfiniBand specifications do not strictly limit the number of entries in the forwarding tables.

4.2. Impact on Destination ID's

As we have seen, the renaming strategy does not significantly increase the size of the forwarding tables. However, this is not the only restriction the destination renaming has. Another restriction is that the number of renamed hosts should not exceed the maximum number of virtual identifiers available to a given host. This number is configured separately for each host by programming its *LMC*. As the *LMC* is a 3 bit value, up to 7 bits of the packet *DLID* can be ignored by the destination host. Hence, there are up to $2^7 = 128$ virtual addresses available for each host. Therefore, in this section we evaluate the number of renamings needed for each particular destination. Moreover, we will only focus on those destinations that need the most renam-

[3]The maximum size of the table is bounded by the LinearFDBCap component of the SwitchInfo attribute.

[4]The maximum size of the table is given by the RandomFDBCap component of the SwitchInfo attribute.

	Total entries			Percentage of table increase		
Sw	Total	Original	New	Min	Max	Avg
8	402	256	145	37	70	57
16	1648	1022	616	39	82	60
32	6733	4078	2637	49	85	65
64	26512	16310	10188	57	71	62

(a)

	Total entries			Percentage of table increase		
Sw	Total	Original	New	Min	Max	Avg
32	5191	3896	1294	32	45	39

(b)

Table 4. Forwarding table overhead. (a) Up*/down* (b) Smart-routing

ings. Network topologies will be the same ones used in the previous section.

Table 5 shows the average number of renamings needed for each destination in the network as well as the absolute maximum number of renamings used for the most renamed destination *LID* for up*/down* and smart-routing, respectively. As we can see, the average number of renamings is very low and it increases slightly as network size increases. For 64-switch networks, the average number of renamings per destination is only 2.1. We can also see that the maximum number of renamings needed for a particular destination does not exceed 7, which is clearly lower than 127. As a consequence, the renaming strategy is not limited by the maximum allowed *LMC* value. Finally, smart-routing requires a lower number of renamings. This explains its lower table size requirements found in Section 4.1.

4.3. Number of Avoided Conflicts

Finally, in this section we qualitatively evaluate the potential benefits of the renaming strategy. The benefits will be measured by counting the number of routing conflicts it avoids in InfiniBand. In other words, without the destination renaming technique, some restrictions will arise when generating the switch forwarding tables. These conflicts either prevent the use of the routing algorithm in IBA or must be avoided (for instance by using the methodology proposed in [5]), which may penalize some paths. However, by using the destination renaming technique, routing restrictions are not introduced and network performance is not degraded.

On the other hand, it must be taken into account that the more connectivity in the network the more alternative routing paths we will find. With more alternatives, more paths can be used for the same destination and different sources. If two or more of these paths meet at the same switch and take different outputs (as they are alternative paths), the InfiniBand $N \times N \rightarrow C$ restriction arises. Hence, as the network becomes better connected, the number of switches with conflicts will increase. To analyze this issue, we evaluate the impact of routing restrictions for a set of 64-switch networks with different number of links connecting neighboring switches. As in previous experiments, we use up*/down* routing and we select a random shortest path for each source-destination pair.

Table 6 shows the number of forwarding table entries with conflicts found in the network when the computed paths are used in an InfiniBand network. We can see that, as more links are used to connect switches, the number of conflicts also increases. In particular, for a highly connected network (8 links per switch to connect to other switches) the conflicts increase up to 8% on average. Therefore, the more connected the network, the more restrictions will arise and the destination renaming will obtain higher benefits.

It should be noted that if these entries with conflicts are avoided with a methodology that restricts routing by increasing the average path length (as in [5]), the network performance may be degraded. With the destination renaming technique, all the conflicts are solved, avoiding routing restrictions and applying the routing algorithm exactly as originally computed.

This is particularly important when implementing a high performance routing algorithm like smart-routing. This routing algorithm obtains high network throughput due to its effort on traffic balancing. In order to obtain a good traffic balance, the smart-routing algorithm uses a linear-programming solver that selects the best set of paths among all the possible paths. The final set of paths can lead to the use of alternative paths to reach the same destinations (in order to get a good traffic balance). Obviously this can not be directly supported on InfiniBand. Moreover, the original paths should not be modified at all, because they have been carefully computed to achieve high performance.

When the smart paths are applied to an InfiniBand network, the percentage of forwarding table conflicts for 32-switch networks with 4 links connecting switches is, on average, 1.86%. With the renaming strategy, the original smart paths can be used in an InfiniBand network without any performance loss.

5. Conclusions

In this paper, we have presented a technique (destination renaming) to implement any routing algorithm designed for irregular networks on IBA. This allows a rapid generation of IBA forwarding tables starting from the ones already computed for other popular environments. We have also proposed a simple implementation of the mechanism as part of the algorithm that is used to distribute the paths into the IBA switches.

Sw	Avg	Maximum
8	1.1	3
16	1.5	4
32	1.8	6
64	2.1	7

(a)

Sw	Avg	Maximum
32	1.2	3

(b)

Table 5. Average and maximum renamings per host (from 10 topologies for each network size). (a) Up*/down* (b) Smart-routing.

Links	Min	Max	Avg
3	0.91	1.44	1.26
4	2.40	3.07	2.71
5	3.90	4.87	4.23
8	7.33	8.71	8.08

Table 6. Percentage of forwarding table entries with conflicts. 64 switches. 4 hosts per switch.

The main problem when implementing the most popular routing algorithms in IBA networks is that, in each switch, both the input port and the destination ID are required to compute the output port. However, IBA only considers the destination ID for routing. As a consequence, they are not directly implementable, because some paths may require different output ports for the same destination and for different input ports. Although one solution has been proposed [5], it has two drawbacks. First, it is intended only for up*/down* and its variants. Second, it may increase the average distance traversed by messages which, in turn, may reduce network performance.

However, the destination renaming technique proposed in this paper can be used with any valid routing algorithm and does not modify at all the originally computed paths. Hence, it does not affect network performance. This technique is based on the virtual addressing feature of IBA that allows a given physical port to be addressed as multiple virtual ports. As the virtual ports are different from the subnet point of view, they can be used to distinguish those paths that require different output ports for the same destination. The technique assigns a different virtual destination identifier for each path with conflicts.

We have analyzed some routing tables generated with the up*/down* and smart routings. We have found that the percentage of entries in forwarding tables with conflicts due to not considering the input port for routing can be as high as 8% for a 256-node network with up*/down* and around 2% for a 128-node network with smart. Hence, these routing tables are not implementable without the destination renaming technique. On the other hand, when using the renaming technique, forwarding table size is increased. We have obtained that for optimized routing algorithms, like smart, this increase is around 40%. For unoptimized routing schemes, the increase does not exceed 65 %. Indeed, the maximum number of renamings required at a given switch is as low as 7 for a 256-node network, which is implementable on IBA.

References

[1] N. J. Boden et al., "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29–36, Feb. 1995.

[2] L. Cherkasova, V. Kotov, and T. Rokicki, "Fibre channel fabrics: Evaluation and design," in *Proc. of 29th Int. Conf. on System Sciences*, Feb. 1995.

[3] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessors interconnection networks," in *IEEE Trans. on Computers*, vol C-36, no. 5, pp. 547-553, May 1987.

[4] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320-1331, Dec. 1993.

[5] J. Duato, "An Efficient Methodology for the Implementation of Up*/Down* Routing on InfiniBand™ Networks," *Tech. Report*, DISCA, UPV, Jan. 2001.

[6] J. Flich et al., "Performance Evaluation of a New Routing Strategy for Irregular Networks with Source Routing," in *Proc. of Int. Conf. on Supercomputing*, May 2000.

[7] J. Flich et al., "Combining In-Transit Buffers with Optimized Routing Schemes to Boost the Performance of Networks with Source Routing," *Proc. of Int. Symp. on High Performance Computing*, Oct. 2000.

[8] D. Garcia, "ServerNet II," in *Proc. of the 1997 Parallel Computing, Routing, and Communication Workshop*, June 1997.

[9] R. W. Horst, "ServerNet deadlock avoidance and fractahedral topologies," in *Proc. of the Int. Parallel Processing Symp.*, April 1996.

[10] InfiniBand™ Trade Association. `http://www.infinibandta.com`.

[11] InfiniBand™ Trade Association, *InfiniBand^{TM} Architecture. Specification Volume 1. Release 1.0.* Available at `http://www.infinibandta.com`.

[12] J.C. Sancho, A. Robles, and J. Duato, "New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks," in *Proc. of Workshop on Communications and Architectural Support for Network-based Parallel Computing*, Jan. 2000.

[13] M. D. Schroeder et al., "Autonet: A high-speed, self-configuring local area network using point-to-point links," *Tech. Report* SRC research report 59, DEC, April 1990.

[14] F. Silla et al., "Efficient Adaptive Routing in Networks of Workstations with Irregular Topology," in *Proc. of Workshop on Communications and Architectural Support for Network-based Parallel Computing*, Feb. 1997.