LID Assignment In InfiniBand Networks

Wickus Nienaber, Xin Yuan, *Member, IEEE* and Zhenhai Duan, *Member, IEEE*

***Abstract*— To realize a path in an InfiniBand network, an address, known as Local IDentifier (LID) in the InfiniBand specification, must be assigned to the destination of the path and used in the forwarding tables of intermediate switches to direct the traffic following the path. Hence, routing in InfiniBand has two components: (1) computing all paths, and (2) assigning LIDs to destinations and using them in intermediate switches to realize the paths. We refer to the task of computing paths as *path computation* and the task of assigning LIDs as *LID assignment*. This paper focuses on the LID assignment component, whose major issue is to minimize the number of LIDs required to support a given set of paths. We prove that the problem of realizing a given set of paths with a minimum number of LIDs is NP-complete, develop an integer linear programming formulation for this problem, design a number of heuristics that are effective and efficient in practical cases, and evaluate the performance of the heuristics through simulation. The experimental results indicate that the performance of our best performing heuristic is very close to optimal. We further demonstrate that by separating path computation from LID assignment and using the schemes that are known to achieve good performance for path computation and LID assignment separately, more effective routing schemes than existing ones can be developed.**

***Index Terms*— InfiniBand, LID Assignment, NP-Complete**

I. Introduction

The InfiniBand architecture (IBA) is an industry standard architecture for interconnecting processing nodes and I/O devices [10]. It is designed around a switch-based interconnect technology with high-speed links. IBA offers high bandwidth and low latency communication and can be used to build many different types of networks including I/O interconnects, system area networks, storage area networks, and local area networks.

An InfiniBand network is composed of one or more subnets connected by InfiniBand routers. Each subnet consists of processing nodes and I/O devices connected by InfiniBand switches. We will use the general term *machines* to refer to processing nodes and I/O devices at the edge of a network. This paper considers the communications within a subnet. A subnet is managed by a subnet manager (SM). By exchanging subnet management packets (SMPs) with the subnet management agents (SMAs) that reside in every InfiniBand device in a subnet, the SM discovers the subnet topology (and topology changes), computes the paths between each pair of machines based on the topology information, configures the network devices, and maintains the subnet.

InfiniBand requires the paths between all pairs of machines to be dead-lock free and deterministic. These paths are realized with a destination based routing scheme. Specifically, machines are addressed by local identifiers (LIDs). Each InfiniBand packet contains in its header the source LID (SLID) and destination LID (DLID) fields. Each switch maintains a forwarding table that maps the DLID to one output port. When a switch receives a packet, it parses the packet header and performs a table lookup using the DLID field to find the output port for this packet. The fact that one DLID is associated with one output port in the forwarding table implies that (1) the routing is deterministic; and (2) each DLID can only direct traffic in one direction in a switch.

Destination based routing limits the paths that can be realized. Consider the paths from nodes 4 and 5 to node 0 in Fig. 1. Assuming that node 0 is associated with only one LID, the paths $4 \rightarrow 3 \rightarrow 1 \rightarrow 0$ and $5 \rightarrow 3 \rightarrow 2 \rightarrow 0$ cannot be supported simultaneously: with one LID for node 0, the traffic toward node 0 in node 3 can only follow one direction. To overcome this problem and allow more flexible routes, IBA introduces a concept called LID Mask Control (LMC) [10], which allows multiple LIDs to be associated with each machine. Using LMC, each machine can be assigned a range of LIDs (from $BASELID$ to $BASELID + 2^{LMC} - 1$). Since LMC is represented by three bits, at most $2^{LMC} = 2^7 = 128$ LIDs can be assigned to each machine. By associating multiple LIDs with one machine, the paths that can be supported by the network are more flexible. For example, the two paths in Fig. 1 can be realized by having two LIDs associated with node 0, one for each path. Nonetheless, since the number of LIDs that can be allocated (to a node or in a subnet) is limited, the paths that can be used in a subnet are still constrained, especially for medium or large sized subnets.

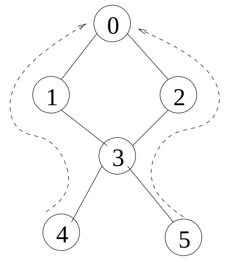


Fig. 1. An example

The use of destination based routing with multiple LIDs for each machine complicates the routing in InfiniBand networks. In addition to finding the paths between machines, the SM must assign LIDs to machines and compute the forwarding tables that realize the paths. Hence, the routing in an InfiniBand network is logically composed of two tasks: the first task is to compute the dead-lock free deterministic paths for each pair of machines; and the second task is to assign LIDs to machines and compute the forwarding tables for realizing the paths determined in the first task. We will use the terms *path computation* and *LID assignment* to refer to these two tasks. The performance of a path computation scheme is commonly evaluated by the link load and load balancing characteristics; the performance of a LID assignment scheme can be evaluated by the number of LIDs needed for a given set of paths; and the performance of a routing scheme, which consists of the two components, can be evaluated with a combination of the two metrics.

W. Nienaber, X. Yuan, and Z. Duan are with the Department of Computer Science, Florida State University, Tallahassee, FL 32306. Email: {nienaber, xyuan, duan}@cs.fsu.edu

Since the IBA specification [10] does not specify the routing algorithms, this area is open to research and many routing schemes have been proposed. Existing routing schemes [1], [2], [3], [8], [13], [14], [15] are all based on the Up*/Down* routing [16], which is originally an adaptive dead-lock free routing scheme. Moreover, all of these schemes integrate the Up*/Down* routing, path selection (selecting deterministic paths among potential paths allowed by the Up*/Down* routing), and LID assignment in one phase. While these schemes provide practical solutions, there are some notable limitations. First, since Up*/Down* routing, path selection, and LID assignment are integrated, these schemes cannot be directly applied to other dead-lock free routing schemes, such as L-turn [6], that may have better load balance properties. Second, the quality of the paths selected by these schemes may not be the best. In fact, the load balancing property of the paths is often compromised by the LID assignment requirement. For example, the fully explicit routing [3] restricts the paths to each destination such that all paths to a destination can be realized by one LID (avoiding the LID assignment problem). Notice that load balancing is one of the most important parameters that determine the performance of a routing system and is extremely critical for achieving high performance in an InfiniBand network. Third, the performance of LID assignment in these schemes is not clear. Since LID assignment is integrated with routing and path selection, the LID assignment problem itself is not well understood.

We propose to separate path computation from LID assignment, which may alleviate the limitations discussed in the previous paragraph: the separation allows path computation to focus on finding paths with good load balancing properties and LID assignment to focus on its own issues. Among the two tasks, path computation in system area networks that require dead-lock free and deterministic paths has been extensively studied and is fairly well understood. There exist dead-lock free adaptive routing schemes, such as Up*/Down* routing [16] and L-turn routing [6], that can be used to identify a set of candidate paths. Path selection algorithms that can select dead-lock free deterministic paths with good load balancing properties from candidate paths have also been developed [12]. Applying these algorithms in InfiniBand networks can potentially result in better paths being selected than those selected by the existing routing schemes developed for InfiniBand. However, in order to apply these path computation schemes, LID assignment, which has not been studied independently from other routing components before, must be investigated. This is the focus in this paper. Note that both path computation and LID assignment are still performed in the topology discovery phase: separating path computation and LID assignment does not mean that LID assignment is done in a later time.

LIDs are limited resources. The number of LIDs that can be assigned to each node must be no more than 128. In addition, the 16-bit SLID and DLID fields in the packet header limit the total number of LIDs in a subnet to be no more than $2^{16} = 64K$. For a large cluster with a few thousand machines, the number of LIDs that can be assigned to each machine is small. For a given set of paths, one can always use a different LID to realize each path. Hence, the number of LIDs needed to realize a routing is no more than the number of paths. However, using this simple LID assignment approach, a system with more than 130 machines cannot be built: it would require more than 129 LIDs to be assigned to a machine in order to realize the (more than 129) paths from other machines to this machine. Hence, the major issue in LID assignment is to minimize the number of LIDs required to realize a given set of paths. Minimizing the number of LIDs enables (1) larger subnets to be built, and/or (2) more paths to be supported in a subnet. Supporting more paths is particularly important when multi-path routing [18] or randomized routing is used. In the rest of this paper, we use the term *LID assignment problem* to refer to the problem of realizing a set of paths with a minimum number of LIDs.

We prove that the LID assignment problem is NP-complete, develop an integer linear programming (ILP) formulation for this problem so that existing highly optimized ILP solvers can be used to obtain solutions for reasonably large systems. We also design various heuristics that are effective in practical cases. These heuristics allow existing methods for finding load balance dead-lock free deterministic paths to be applied to InfiniBand networks. We evaluate the proposed heuristics through simulation. The results indicate that our best performing heuristic achieves near optimal performance: the optimal solution is less than 3% better in all cases that we studied. We further demonstrate that by separating path computation from LID assignment and using the schemes that are known to achieve good performance for path computation and LID assignment separately, more effective routing methods than existing ones can be developed.

The rest of the paper is organized as follows. In Section II, we introduce the notations and formally define the LID assignment problem. The NP-completeness of the LID assignment problem is proven in Section III. Section IV gives the integer linear programming formulation for this problem. Section V describes the proposed heuristics. The performance of the heuristics is study in Section VI. Finally, Section VII concludes the paper.

II. Problem definition

An InfiniBand subnet consists of *machines* connected by *switches*. A *node* refers to either a switch or a machine. InfiniBand allows both regular and irregular topologies. The techniques developed in this paper are mainly for irregular topologies. The links are bidirectional; a machine can have multiple ports connecting to one or more switches; and multiple links are allowed between two nodes. We model an InfiniBand network as a directed multigraph, $G = (V, E)$, where E is the set of directed *edges* and V is the set of switches and machines. Let M be the set of machines and S be the set of switches. $V = M \cup S$. Let there exist n links between two nodes u and v. The links are numbered from 1 to n. The n links are modeled by $2n$ direct edges $((u, v), i)$ (or $u \xrightarrow{i} v$) and $((v, u), i)$ (or $v \xrightarrow{i} u$), $1 \leq i \leq n$. The i-th link between nodes u and v is modeled by two direct edges $((u, v), i)$ and $((v, u), i)$. An example InfiniBand topology is shown in Fig. 2. In this example, switches $s0$ and $s1$ are connected by two links; machine $m3$ is connected to two switches $s1$ and $s2$.

A *path* $p = u \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} ... \xrightarrow{i_{n-1}} a_n \xrightarrow{i_n} v$ consists of a set of directed edges $\{u \xrightarrow{i_0} a_1, a_1 \xrightarrow{i_1} a_2, ..., a_n \xrightarrow{i_n} v\}$. $NODE(p) = \{u, a_1, a_2, ..., a_n, v\}$ is the set of nodes that the path $p = u \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} ... \xrightarrow{i_{n-1}} a_n \xrightarrow{i_n} v$ goes through. $SRC(p) = u$ is the source of path p and $DST(p) = v$ is the destination of path p. A path $p = u \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} ... \xrightarrow{i_{n-1}} a_n \xrightarrow{i_n} v$ is *end-to-end* when $SRC(p) = u \in M$ and $DST(p) = v \in M$. In this case, path p is said to be an *end-to-end* path. For example, the dark line in Fig. 2 shows an end-to-end path $m0 \xrightarrow{1} s0 \xrightarrow{2} s1 \xrightarrow{1} s2 \xrightarrow{1} m4$.

The path computation of a routing scheme determines a set of end-to-end paths, $R = \{p_1, p_2, ...\}$, that must be realized by LID assignment.

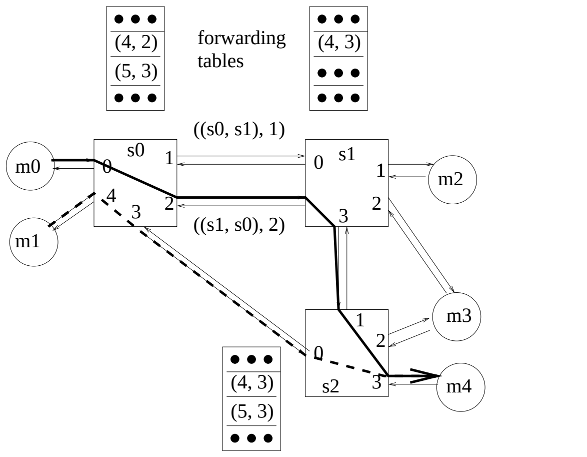


Fig. 2. An InfiniBand network topology (LIDs 4 and 5 are assigned to m4)

InfiniBand realizes each path through destination based routing. In Fig. 2, we show the entries in the forwarding tables that realize two paths $m0 \xrightarrow{1} s0 \xrightarrow{2} s1 \xrightarrow{1} s2 \xrightarrow{1} m4$ (the solid dark line) and $m1 \xrightarrow{1} s0 \xrightarrow{1} s2 \xrightarrow{1} m4$ (the dotted dark line). This example assumes that LIDs 4 and 5 are assigned to machine $m4$ and the entries are illustrated with a random forwarding table format: each forwarding table entry is of the form $(DLID, output\_port)$, forwarding packets with destination address $DLID$ to port $output\_port$. As shown in the example, path $m0 \xrightarrow{1} s0 \xrightarrow{2} s1 \xrightarrow{1} s2 \xrightarrow{1} m4$ is realized by having entry $(DLID = 4, output\_port = 2)$ in the forwarding table in switch $s0$, $(DLID = 4, output\_port = 3)$ in $s1$, and $(DLID = 4, output\_port = 3)$ in $s2$. Once the forwarding tables are installed, machine $m0$ can send packets to $m4$ following this path by making $DLID = 4$ in the packet header. Note that the physical installation of the forwarding table in different switches is performed by the SM in the path distribution phase, which is beyond the scope of this paper.

To realize a path p towards a destination v, a LID LID_v that is associated with the node v must be used and an entry in the form of $(LID_v, output\_port)$ must be installed in each of the intermediate switches along the path. Once LID_v is associated with one $output\_port$ in a switch, it cannot be used to realize other paths that use different output ports in the same switch. We will use the term *assigning LID_v to path p* to denote the use of LID_v to realize path p. In the example in Fig. 2, LID 4 is assigned to path $m0 \xrightarrow{1} s0 \xrightarrow{2} s1 \xrightarrow{1} s2 \xrightarrow{1} m4$ and LID 5 is assigned to path $m1 \xrightarrow{1} s0 \xrightarrow{1} s2 \xrightarrow{1} m4$.

Since different destinations are assigned non-overlapping ranges of LIDs in InfiniBand networks, the number of LIDs required for realizing a routing is equal to the sum of the number of LIDs required for each destination. In other words, the **LID assignment problem for realizing a set of end-to-end paths can be reduced to the LID assignment problem for each individual destination in the set of paths**. Let $R = \{p_1, p_2, ...\}$ be the set of end-to-end paths and $D = \{d | \exists p_i \in R, DST(p_i) = d\}$ be the set of destinations in R. Let $d \in D$ be a destination node in some paths in R, R_d be the set of all paths with destination d in R, $\{p | p \in R \text{ } and \text{ } DST(p) = d\}$. We have $R = \cup_{d \in D} R_d$. Let the minimum number of LIDs needed for realizing R_d be L_d and the minimum number of LIDs needed for realizing R be L. Since LIDs for different destination nodes are independent of one another,

$$L = \sum_{d \in D} L_d.$$

We will call LID assignment for each R_d the *single destination LID assignment problem*. In the rest of the paper, we will focus on the single destination problem. Unless specified otherwise, all paths are assumed to have the same destination. Next, we will introduce concepts and lemmas that lead to the formal definition of the single destination LID assignment problem.

Definition 1: Two paths p_1 and p_2 (with the same destination) are said to have a *split* if there exists a node $a \in NODE(p_1) \cap NODE(p_2)$, $a \xrightarrow{i} b \in p_1$ and $a \xrightarrow{j} c \in p_2$, such that either $i \neq j$ or $b \neq c$.

Basically, two paths have a split when (1) both paths share an intermediate node, and (2) the outgoing links from the intermediate node are different. Fig. 3 (a) shows the case when two paths have a split.

Lemma 1: When two paths p_1 and p_2 have a split, they must be assigned different LIDs. When p_1 and p_2 do not have any split, they can share the same LID (be assigned the same LID).

Proof: We will first prove the first proposition in this lemma: when two paths p_1 and p_2 have a split, they must be assigned different LIDs. Let p_1 and p_2 be the two paths that have a split. From Definition 1, there exists a node $a \in NODE(p_1) \cap NODE(p_2)$, $a \xrightarrow{i} b \in p_1$ and $a \xrightarrow{j} c \in p_2$, such that either $i \neq j$ or $b \neq c$. Consider the forwarding table in node a. When either $i \neq j$ or $b \neq c$, $a \xrightarrow{i} b \in p_1$ uses a different port from $a \xrightarrow{j} c \in p_2$. Since one LID can only be associated with one output port in the forwarding table, two LIDs are needed in switch a to realize the two directions. Hence, p_1 and p_2 must be assigned different LIDs.

Now consider the second proposition: when two paths p_1 and p_2 do not have any split, they can share the same LID (be assigned the same LID). Let p_1 and p_2 be the two paths that do not have a split. There are two cases. The first case, shown in Fig. 3 (b) (1), is when the two paths do not share any intermediate nodes. The second case, shown in Fig. 3 (b) (2), is when two paths share intermediate nodes, but do not split after they join. In both cases, each switch in the network needs to identify at most one outgoing port to realize both paths. Hence, at most one LID is needed in all switches to realize both paths. In other words, The two paths can be assigned the same LIDs. □

It must be noted that the statements "p_1 can share a LID with p_2" and "p_1 can share a LID with p_3" do not imply that "p_2 can share a LID with p_3". Consider paths $p_1 = m2 \rightarrow s1 \rightarrow s2 \rightarrow m4$, $p_2 = m0 \rightarrow s0 \rightarrow s1 \rightarrow s2 \rightarrow m4$, and $p_3 = m_1 \rightarrow s0 \rightarrow s2 \rightarrow m4$ in Fig. 2. Clearly, p_1 can share a LID with p_2 and p_1 can share a LID with p_3, but p_2 and p_3 have a split at switch $s0$ and cannot share a LID. The following concept of *configuration* defines a set of paths that can share one LID.

Definition 2: A *configuration* is a set of paths (with the same destination) $C = \{p_1, p_2, ...\}$ such that no two paths in the set have a split.

Lemma 2: All paths in a configuration can be realized by one LID.

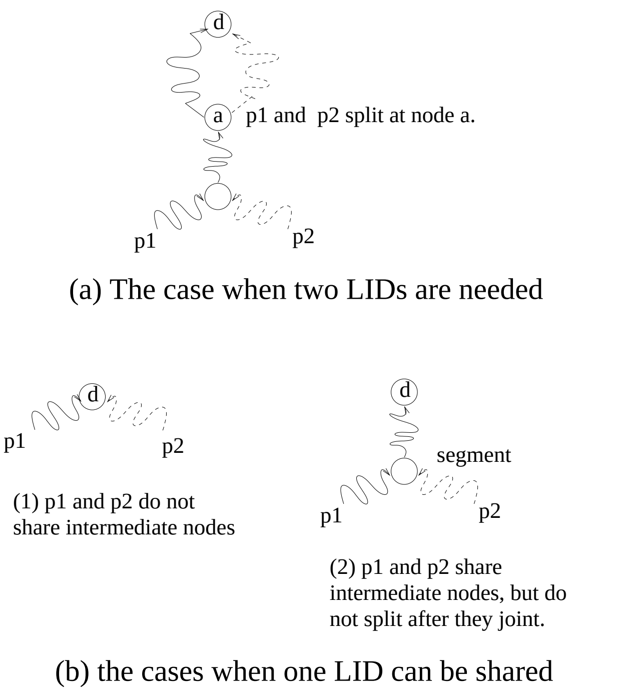

d
a
p1 and p2 split at node a.
p1
p2
(a) The case when two LIDs are needed
d
p1
p2
(1) p1 and p2 do not share intermediate nodes
d
segment
p1
p2
(2) p1 and p2 share intermediate nodes, but do not split after they joint.
(b) the cases when one LID can be shared

Fig. 3. The cases when a LID can and cannot be shared between two paths

Proof: Let l be a LID. Consider any switch, s, in the system. This switch can either be used by some paths in the configuration or not used by any path. If s is used by some paths, by the definition of configuration, all paths that pass through s must follow one outgoing port, $port$, (otherwise, the paths have a split at s and the set of paths is not a configuration). Hence, the entry $(DLID = l, output\_port = port)$ can be shared by all paths using s. If s is not used by any paths in the configuration, no entry is needed in the forwarding table to realize the paths in the configuration. Hence, LID l can be used in the switches along all paths in configuration to realize all of the paths. □

Definition 3 (Single destination LID assignment problem ($SD(G, d, R_d)$)): Let the network be modeled by the multi-graph G, d be a node in G, $R_d = \{p_1, p_2, ...\}$ be a single destination routing (for all $p_i \in R_d$, $DST(p_i) = d$). The single destination LID assignment problem is to find a function $c : R_d \rightarrow \{1, 2, ..., k\}$ such that (1) $c(p_i) \neq c(p_j)$ for every pair of paths p_i and p_j that have a split, and (2) k is minimum.

Let $c : R_d \rightarrow \{1, 2, ..., k\}$ be a solution to $SD(G, d, R_d)$. Let $R_d^i = \{p_j | c(p_j) = i\}$, $1 \leq i \leq k$. By definition, R_d^i is a configuration; $R_d = \cup_{i=1}^{k} R_d^i$; and $R_d^i \cap R_d^j = \phi$, $i \neq j$. Thus, $SD(G, d, R_d)$ is equivalent to the problem of partitioning R_d into k disjoint sets R_d^1, R_d^2, ..., R_d^k such that (1) each R_d^i is a configuration, and (2) k is minimum. When the disjoint configurations R_d^1, R_d^2, ..., R_d^k are found, the routing R_d can be realized by k LIDs with one LID assigned to all paths in R_d^i, $1 \leq i \leq k$.

$SD(G, d, R_d)$ states the optimization version of this problem. The corresponding decision problem, denoted as $SD(G, d, R_d, k)$, decides whether there exists a function $c : R_d \rightarrow \{1, 2, ..., k\}$ such that $c(p_i) \neq c(p_j)$ for every pair of paths p_i and p_j that have a split.

Since InfiniBand realizes multiple LIDs for each destination using the LID Mask Control (LMC) mechanism, the actual number of LIDs assigned to each destination must be a power of two, up to 128. Hence, if the solution to $SD(G, d, R_d)$ is k, the actual number of LIDs assigned to d is $2^{\lceil lg(k) \rceil}$. For example, when $k = 4$, $2^{\lceil lg(k) \rceil} = 4$; when $k = 5$, $2^{\lceil lg(k) \rceil} = 8$.

III. NP-COMPLETENESS

Theorem 1: $SD(G, d, R_d, k)$ is NP-complete.

Proof: We first show that $SD(G, d, R_d, k)$ belongs to NP problems. Suppose that we have a solution for $SD(G, d, R_d, k)$, the verification algorithm first affirms the solution function $c : R_d \rightarrow \{1, 2, ..., k\}$. It then checks for each pair of paths $p1$ and $p2$, $c(p1) = c(p2)$, that they do not have a split. It is straightforward to perform this verification in polynomial time. Thus, $SD(G, d, R_d, k)$ is an NP problem.

We prove that $SD(G, d, R_d, k)$ is NP-complete by showing that the graph coloring problem, which is a known NP-complete problem, can be reduced to this problem in polynomial time. The graph-coloring problem is to determine the minimum number of colors needed to color a graph. The k-coloring problem is the decision version of the graph coloring problem. A k-coloring of an undirected graph $G = (V, E)$ is a function $c : V \rightarrow \{1, 2, ..., k\}$ such that $c(u) \neq c(v)$ for every edge $(u, v) \in E$. In other words, the numbers 1, 2, ..., k represent the k colors, and adjacent vertexes must have different colors.

The reduction algorithm takes an instance $< G, k >$ of the k-coloring problem as input. It computes the instance $SD(G', d, R_d, k)$ as follows. Let $G = (V, E)$ and $G' = (V', E')$. The following vertexes are in V'.

- The destination node $d \in V'$.
- For each $u \in V$, two nodes $n_u, n_{u'} \in V'$.
- For each $(u, v) \in E$, a node $n_{u,v} \in V'$. Since G is an undirected graph, (u, v) is the same as (v, u) and there is only one node for each $(u, v) \in E$ (node $n_{u,v}$ is the same as node $n_{v,u}$).

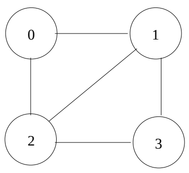
The edges in G' are as follows. For each n_u, let nodes n_{u,i_1}, n_{u,i_2}, ..., n_{u,i_m} be the nodes corresponding to all node u's adjacent edges in G. The following edges: (n_u, n_{u,i_1}), (n_{u,i_1}, n_{u,i_2}), ..., $(n_{u,i_{m-1}}, n_{u,i_m})$, $(n_{u,i_m}, n_{u'})$, $(n_{u'}, d)$ are in E'. Basically, for each node $u \in G$, there is a path in G' that goes from n_u, through each of the nodes in corresponding to the edges adjacent to u in G, then through $n_{u'}$ to node d.

Each node $u \in V$ corresponds to a path p_u in R_d. p_u starts from node n_u, it goes through every node in G' that corresponds to an edge adjacent to u in G, and then goes to node $n_{u'}$, and then d. Specifically, let n_{u,i_1}, n_{u,i_2}, ..., n_{u,i_m} be the nodes corresponding to all node u's adjacent edges in G, $p_u = n_u \xrightarrow{1} n_{u,i_1} \xrightarrow{1} n_{u,i_2} ... \xrightarrow{1} n_{u,i_m} \xrightarrow{1} n_{u'} \xrightarrow{1} d$.

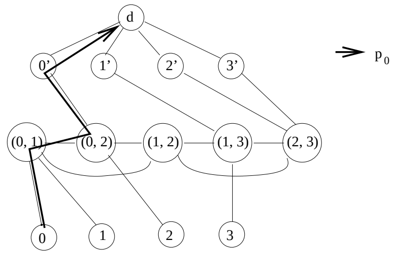
From the construction of p_u, we can see that if nodes u and v are adjacent in G ($(u, v) \in E$), both p_u and p_v go through node $n_{u,v}$ and have a split at this node. If u and v are not adjacent, p_u and p_v do not share any intermediate node, and thus, do not have a split. Hence, $p_u, p_v \in R_d$ *have a split if and only if* u *and* v *are adjacent nodes.*

Fig. 4 shows an example of the construction of G', d and R_d. For the example G in Fig. 4 (a), we first create the destination node d in G'. The second and fourth rows of nodes in Fig. 4 (b) correspond to the two nodes $n_{u'}$ and n_u for each node $u \in V$. The third row of nodes corresponds to the edges in G. Each node u in G corresponds to a path p_u in R_d, $R_d = \{p_0, p_1, p_2, p_3\}$, where

$p_0 = n_0 \xrightarrow{1} n_{0,1} \xrightarrow{1} n_{0,2} \xrightarrow{1} n_{0'} \xrightarrow{1} d$, $p_1 = n_1 \xrightarrow{1} n_{0,1} \xrightarrow{1} n_{1,2} \xrightarrow{1} n_{1,3} \xrightarrow{1} n_{1'} \xrightarrow{1} d$, $p_2 = n_2 \xrightarrow{1} n_{0,2} \xrightarrow{1} n_{1,2} \xrightarrow{1} n_{2,3} \xrightarrow{1} n_{2'} \xrightarrow{1} d$, and $p_3 = n_3 \xrightarrow{1} n_{1,3} \xrightarrow{1} n_{2,3} \xrightarrow{1} n_{3'} \xrightarrow{1} d$. The path p_0 that corresponds to node 0 in Fig. 4 (a) is depicted in Fig. 4 (b). It can easily see that in this example, $p_u, p_v \in R_d$ have a split if and only if u and v are adjacent nodes.



(a) An example graph for graph coloring



(b) The corresponding graph for LID assignment

Fig. 4. An example of mapping G to G'

To complete the proof, we must show that this transformation is indeed a reduction: the graph G can be k-colored if and only if $SD(G', d, R_d, k)$ has a solution.

First, we will show the sufficient condition: if G can be k-colored, $SD(G', d, R_d, k)$ has a solution. Let $c : V \rightarrow \{1, 2, ..., k\}$ be the solution to the k-coloring problem. We can partition R_d into $R_d^i = \{p_u | c(u) = i\}$. Let $p_u, p_v \in R_d^i$. Since $c(u) = c(v)$, nodes u and v are not adjacent in G. From the construction of G', d, and R_d, p_u and p_v do not have split. By definition, R_d^i is a configuration. Hence, R_d can be partitioned into k configurations R_d^1, R_d^2, ..., R_d^k and $SD(G', d, R_d, k)$ has a solution.

Now, we will show the necessary condition: if $SD(G', d, R_d, k)$ has a solution, G can be k-colored. Since $SD(G', d, R_d, k)$ has a solution, R_d can be partitioned into k configurations R_d^1, R_d^2, ..., and R_d^k. Let $p_u, p_v \in R_d^i$, $1 \leq i \leq k$. Since R_d^i is a configuration, p_u does not have split with p_v in G'. From the construction of G', d, R_d, u and v are not adjacent in G. Hence, all nodes in each configuration can be colored with the same color and the mapping function $c : V \rightarrow \{1, 2, ..., k\}$ can be defined as $c(u) = i$ if $p_u \in R_d^i$, $1 \leq i \leq k$. Hence, if $SD(G', d, R_d, k)$ has a solution, G can be k-colored. □

IV. Integer Linear Programming Formulation

Since some highly optimized ILP solvers have been developed, a common approach to handle an NP-complete problem is to develop an ILP formulation for this problem so that existing ILP solvers can be used to obtain solutions for reasonably sized problems. In this section, we will give a 0-1 ILP formulation for the single destination LID assignment problem.

Let $SD(G, d, R_d, k)$ be the decision version of the single destination LID assignment problem. The 0-1 ILP formulation for this problem is as follows. For each $p \in R_d$ and i, $1 \leq i \leq k$, a variable $X_{p,i}$ is created. The value of the solution for $X_{p,i}$ is either 0 or 1. $X_{p,i} = 1$ indicates that p is in configuration R_d^i and $X_{p,i} = 0$ indicates that p is not in configuration R_d^i. The ILP formulation does not have an optimization objective function, instead, it tries to determine whether there exists any solution under the following constraints.

First, the values for $X_{p,i}$ must be either 0 or 1:

For any $p \in R_d$ and $1 \leq i \leq k$, $0 \leq X_{p,i} \leq 1$ and $X_{p,i}$ is an integer.

Second, each $p \in R_d$ must be assigned to exactly one configuration:

$$\text{For any } p \in R_d,\ \sum_{i=1}^{k} X_{p,i} = 1.$$

Third, for any two paths $p, q \in R_d$ that have a split, they cannot be assigned to the same configuration:

For any $p, q \in R_d$ that have a split, $X_{p,i} + X_{q,i} \leq 1$, $1 \leq i \leq k$.

A solution to this formulation for a given k indicates that at most k LIDs are needed for the problem. To solve the optimization version of the problem (finding the minimum k), one can first use a heuristic (e.g. any one described in the next section) to find an initial k, and then repeatedly solve the ILPs for $SD(G, d, R_d, k-1)$, $SD(G, d, R_d, k-2)$, and so on. Let m be the value for the first instance that $SD(G, d, R_d, m)$ does not have a solution, the minimum k is $m + 1$.

Consider the ILP formulation for realizing $R_{m0} = \{p_1, p_2, p_3, p_4\}$ in Fig. 5, where $p_1 = m1 \xrightarrow{1} s4 \xrightarrow{1} s1 \xrightarrow{1} s0 \xrightarrow{1} m0$, $p_2 = m2 \xrightarrow{1} s4 \xrightarrow{1} s3 \xrightarrow{1} s2 \xrightarrow{1} s0 \xrightarrow{1} m0$, $p_3 = m4 \xrightarrow{1} s5 \xrightarrow{1} s2 \xrightarrow{1} s0 \xrightarrow{1} m0$, and $p_4 = m3 \xrightarrow{1} s5 \xrightarrow{1} s3 \xrightarrow{1} s1 \xrightarrow{1} s0 \xrightarrow{1} m0$. Assuming $k = 2$, there are eight variables in the ILP: $X_{p_1,1}$, $X_{p_1,2}$, $X_{p_2,1}$, $X_{p_2,2}$, $X_{p_3,1}$, $X_{p_3,2}$, $X_{p_4,1}$, and $X_{p_4,2}$. The constraints are as follows.

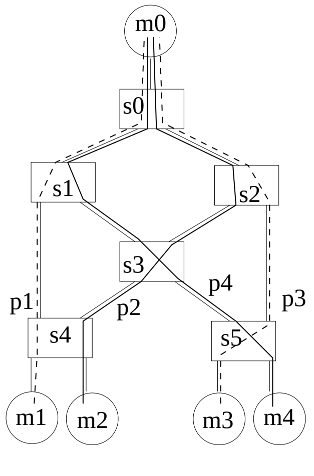


Fig. 5. An example of LID assignment

First, the solutions for the eight variables must be 0 and 1, which is enforced by the following in-equations and requiring the solutions to be integers:

$$0 \leq X_{p_1,1} \leq 1,\ 0 \leq X_{p_1,2} \leq 1,$$
$$0 \leq X_{p_2,1} \leq 1,\ 0 \leq X_{p_2,2} \leq 1,$$
$$0 \leq X_{p_3,1} \leq 1,\ 0 \leq X_{p_3,2} \leq 1,$$
$$0 \leq X_{p_4,1} \leq 1,\ 0 \leq X_{p_4,2} \leq 1$$

Second, the following constraints enforce that each path is assigned one LID:

$$X_{p_1,1} + X_{p_1,2} = 1,\ X_{p_2,1} + X_{p_2,2} = 1,$$
$$X_{p_3,1} + X_{p_3,2} = 1,\ X_{p_4,1} + X_{p_4,2} = 1$$

Finally, among the four paths, p_1 has a split with p_2, p_2 has a split with p_4, and p_4 has split with p_3. To ensure that paths that have splits are not assigned the same LID, the following constraints are added.

$$X_{p_1,1} + X_{p_2,1} \leq 1,\ X_{p_1,2} + X_{p_2,2} \leq 1,$$
$$X_{p_2,1} + X_{p_4,1} \leq 1,\ X_{p_2,2} + X_{p_4,2} \leq 1,$$
$$X_{p_3,1} + X_{p_4,1} \leq 1,\ X_{p_3,2} + X_{p_4,2} \leq 1$$

This ILP formulation has the solution: $X_{p_1,1} = 1$, $X_{p_1,2} = 0$, $X_{p_2,1} = 0$, $X_{p_2,2} = 1$, $X_{p_3,1} = 0$, $X_{p_3,2} = 1$, $X_{p_4,1} = 1$, $X_{p_4,2} = 0$. This means that p_1 and p_4 are realized with one LID and that p_2 and p_3 are realized with another LID.

V. LID ASSIGNMENT HEURISTICS

The ILP formulation allows the problem for a reasonable sized system to be solved with existing ILP solvers. We develop heuristic algorithms so that problems for large networks can be solved. All of the proposed heuristics are oblivious to routing. They do not make any assumption about the routes to be assigned LIDs. As a result, they can be applied to any routing scheme, including multipath routing and schemes that can yield duplicate routes. All of our heuristics are based on the concept of minimal configuration set, which is defined next.

Definition 4: Given a single destination routing $R_d = \{p_1, p_2, ...\}$, the set of configurations $MC = \{C_1, C_2, ..., C_k\}$ is a **minimal configuration set** for R_d if and only if all of the following conditions are met:

- each $C_i \in MC$, $1 \leq i \leq k$, is a configuration;
- each $p_i \in R_d$ is in exactly one configuration in MC;
- for each pair of configuration C_i and $C_j \in MC$, $i \neq j$, there exist $p_x \in C_i$ and $p_y \in C_j$ such that p_x and p_y have a split.

The configuration set is minimal in that there do not exist two configurations in the set that can be further merged. From Lemma 2, all paths in one configuration can be realized by 1 LID. Hence, assuming that $MC = \{C_1, C_2, ..., C_k\}$ is a **minimal configuration set** for routing R_d, the routing R_d can be realized by k LIDs. All of the heuristics attempt to minimize the number of LIDs needed by finding a minimal configuration set.

A. Greedy heuristic

For a given R_d, the greedy LID assignment algorithm creates configurations one by one, trying to put as many paths into each configuration as possible to minimize the number of configurations needed. This heuristic repeats the following process until all paths are in the configurations: create an empty configuration (current configuration), check each of the paths in R_d that has not been included in a configuration whether it has a split with the paths in the current configuration, and greedily put the path in the configuration (when the path does not split with any paths in the configuration). The algorithm is shown in Fig. 6. Each configuration (or path) can be represented as an array of size $|V|$ that stores for each node the outgoing link from the node (in a configuration or a path, there can be at most *one* outgoing link from each node). Using this data structure, checking whether a path has a split with any path in a configuration takes $O(|V|)$ time (line (5) in Fig. 6); and adding a path in a configuration also takes $O(|V|)$ time (line (6)). The loop at line (4) runs for at most $|R_d|$ iterations and the loop at line (2) runs for at most k iterations, where k is the number of LIDs allocated. Hence, the complexity of the algorithm is $O(k \times |R_d| \times |V|)$, where k is the number of LIDs allocated, R_d is the set of paths, and V is the set of nodes in the network.

```
(1)  MC = $\phi$, k = 1
(2)  repeat
(3)    $C_k = \phi$
(4)    for each $p \in R_d$
(5)        if $p$ does not split with any path in $C_k$ then
(6)            $C_k = C_k \bigcup \{ p \}$, $R_d = R_d - \{ p \}$
(7)        end if
(8)    end for
(9)    MC = MC $\bigcup$ { $C_k$ }, k = k + 1
(10) until $R_d = \phi$
```

Fig. 6. The greedy heuristic

We will use an example to show how the greedy heuristic algorithm works and how its solution may be sub-optimal. Consider realizing $R_{m0} = \{p_1, p_2, p_3, p_4\}$ in Fig. 5 in the previous section. The greedy algorithm first creates a configuration and puts p_1 in the configuration. After that, the algorithm tries to put other paths into this configuration. The algorithm considers p_2 next. Since p_1 and p_2 split at switch $s4$, p_2 cannot be included in this configuration. Now, consider p_3. Since p_3 and p_1 do not have any joint intermediate nodes, p_3 can be included in the configuration. After that, since p_4 splits with p_3 at switch $s5$, it cannot be included in this configuration. Thus, the first configuration will contain paths p_1 and p_3. Since we have two paths p_2 and p_4 left unassigned, new configurations are created for these two paths. Since p_2 and p_4 split at switch $s3$, they cannot be included in one configuration. Hence, the greedy algorithm realizes R_{m0} with three configurations: $C_1 = \{p_1, p_3\}$, $C_2 = \{p_2\}$, and $C_3 = \{p_4\}$. Thus, 3 LIDs are needed to realize the routing with the greedy heuristic. Clearly, this is a sub-optimal solution since solving the ILP formulation in the previous section requires only 2 LIDs.

B. Split-merge heuristics

For a given R_d, the greedy algorithm tries to share LIDs as much as possible by considering each path in R_d: the minimal configuration set is created by merging individual paths into configurations. The split-merge heuristics use a different approach to find the paths that share LIDs. This class of heuristics has two phases: in the first phase, R_d is split into configurations; in the second phase, the greedy heuristic is used to merge the resulting configurations into a minimal configuration set, which is the final LID assignment. In the split phase, the working set initially contains one item R_d. In each iteration, a node is selected. Each item (a set of paths) in the working set is partitioned into a number of items such that each of the resulting items does not contain paths that split in the node selected (the paths that split in the selected node are put in different items). After all nodes are selected, the resulting items in the working set are guaranteed to be configurations: paths in one item do not split in any of the nodes. In the worst case, each resulting configuration contains one path at the end of the split phase and the split-merge heuristic is degenerated into the greedy algorithm. In general cases, however, the split phase will produce configurations that include multiple paths. It is hoped that the split phase will allow a better starting point for merging than individual paths. The heuristic is shown in Fig. 7. Using a linked list to represent a set and the data structure used in the greedy algorithm to represent a path, the operations in the loop from line (4) to (7) can be done in $O(|R_d||V|)$ operations: going through all $|R_d|$ paths and updating the resulting set that contains each path with $O(|V|)$ operations. Hence, the worst case time complexity for the whole algorithm is $O(|V|^2|R_d| + k|V||R_d|)$.

```
     /* splitting */
(1)  $S = \{R_d\}$, $ND = V$
(2)  repeat
(3)    Select a node, $a$, in ND;
(4)    for each $S_i \in S$ do
(5)      partition paths in $S_i$ that splits at node $a$ into
         multiple sets $S_i^1$, $S_i^2$, ...,$S_i^j$
(6)      $S = (S - \{S_i\}) \cup S_i^1 \cup ... \cup S_i^j$;$ND = ND - \{a\}$
(7)    end for
(8)  until $ND = \phi$
     /* merging */
(9)  apply the greedy heuristic on $S$.
```

Fig. 7. The split-merge heuristic.

Depending on the order of the nodes selected in the split phase, there are variations of this split-merge heuristic. We consider two heuristics in our evaluation, the *split-merge/S* heuristic that selects the node used by the smallest number of paths first, and the *split-merge/L* heuristic that selects the node used by the largest number of paths first.

C. Graph coloring heuristics

This heuristic converts the LID assignment problem into a graph coloring problem. First, a split graph is built. For all paths $p_i \in R_d$, there exists a node n_{p_i} in the split graph. If p_i and p_j have a split with each other, an edge (n_{p_i}, n_{p_j}) is added in the split graph. It can be easily shown that if the split graph can be colored with k colors, R_d can be realized with k LIDs: the nodes assigned the same color correspond to the nodes assigned the same LID. This conversion allows heuristics that are designed for graph coloring to be applied to solve the LID assignment problem. Consider the example in Fig. 5. The corresponding split graph is shown in Fig. 8. Node $p1$ has an edge with node $p2$ as they split at $s4$, node $p2$ has an additional edge with $p4$ as they split at $s3$. Finally, $p3$ has an edge with $p4$ as they split with each other at $s5$.

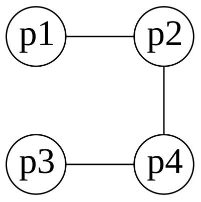


Fig. 8. The split graph for Fig. 5

While many graph coloring algorithms can be applied to color the split graph, we use a simple coloring heuristic in this paper. In our heuristic, the graph is colored by applying the colors one-by-one. Fig. 9 shows one of the graph coloring algorithms that we consider: the *most split path first heuristic* (color/L). The heuristic works as follows. Starting from the split graph, a node with the largest degree is selected and assigned a color. After that, this node and all other nodes that are adjacent to it are removed from the working graph: all remaining nodes can be assigned the same color. Hence, we can use the same heuristic to select the next node to assign the same color. This process is repeated until the working graph becomes empty. In one such iteration, all nodes that are assigned the one color are found. In the next iteration, the same process is applied to all nodes that are not colored. This process repeats until each node is assigned a color.

```
(1)   Compute the split graph for the problem.
(2)   $workinggraph$ = split graph;$color$=1;
(3)   repeat
(4)     repeat
(5)       Select a node in $workinggraph$ with the
            largest degree
(6)       Assign $color$ to the node
(7)       Remove the node and all adjacent nodes
            from $workinggraph$
(8)     until no more nodes in $workinggraph$
(9)     $workinggraph$ = all nodes without colors plus the
            edges between them
(10)    $color$ ++;
(11)  until $workinggraph$ is empty (all nodes are colored)
```

Fig. 9. Most split path first heuristic (color/L).

Consider the example in Fig. 8. In the first iteration (assigning color 1), nodes $p2$ and $p4$ have the largest degree and could be chosen. Let us assume that $p2$ is selected first and assigned color 1. After that, nodes $p1$ and $p4$ are removed since they are adjacent to $p2$. Thus, node $p3$ is selected and assigned color 1. In the second iteration (assigning color 2), the working graph contains node $p1$ and $p4$ with no edges. Hence, both nodes are assigned color 2. The algorithm results in two configurations: $C_1 = \{p2, p3\}$ and $C_2 = \{p1, p4\}$.

The heuristic is embedded in the selection of a node to color in Line (5). We consider two coloring based heuristics in this paper: the *most split path first heuristic* (color/L) showed in Fig. 9 and the *least split path first heuristic* (color/S) when the node in the split graph with the smallest nodal degree is selected (node $p1$ or node $p3$ in Fig. 8). The worst case time complexity for computing the split graph is $O(|R_d|^2|V|)$: it takes $O(|V|)$ time to decide whether an edge exists between two nodes; and there are at most $O(|R_d|^2)$ edges in the split graph. After the graph is created, in all iterations of the loop in Lines (4) to (8) in Fig. 9, $O(|R_d|^2)$ edges are removed from $workinggraph$ in the worst case. Using a priority queue to maintain the sorted order for the nodes based on the nodal degree, each remove operation requires $O(lg(|R_d|^2) = O(lg(|R_d|)$ operations in the worst case. The outer loop (lines (3) to (11)) runs k times. Hence, the complexity for coloring is $O(k \times |R_d|^2 lg(|R_d|))$ and the total time for this heuristic is $O(|R_d|^2|V| + k \times |R_d|^2 lg(|R_d|))$.

VI. Performance study

We carry out simulations to investigate various aspects of the proposed heuristics and different routing schemes. The study consists of four parts: (1) investigating the relative performance of the proposed LID assignment heuristics and identifying the most effective heuristic, (2) investigating the absolute performance of the heuristics by comparing their solutions with the optimal solutions obtained using the ILP formulation, (3) probing the performance on regular and near regular topologies, and (4) studying the performance of various routing schemes.

For random irregular topologies, we report results on systems with 16, 32, and 64 switches and 64, 128, 192, 256, and 512 machines. We will use the notion X/Y to represent the system configuration with X machines and Y switches. For example, $128/16$ denotes the configuration with 128 machines and 16 switches. Each random irregular topology is generated as follows. First, a random switch topology is generated using the *Georgia Tech Internetwork Topology Models (GT-ITM)* [19]. The average nodal degree is 8 for all three cases (16, 32, and 64 switches). After the switch topology is generated, the machines are randomly distributed among the switches with a uniform probability distribution. Note that the topologies generated by GT-ITM are not limited to Internet-like topologies, this package can generate random topologies whose connectivity follows many different probability distribution. Note also that the average number of ports in a switch in the evaluated configurations ranges from 10 to 48, which covers the common types of practical InfiniBand switches. The average nodal degree in the switch topology is 8, and machines are also attached to switches.

Our LID assignment schemes do not make any assumption about path computation and can work with any routing schemes including multi-path routing, non dead-lock free routing, and other path computation schemes such as the recently developed layered routing scheme [11]. However, the paths computed with different schemes may exhibit different characteristics, which may affect the effectiveness of the LID assignment heuristics. In the evaluation, we consider two Up*/Down routing based schemes that guarantee to produce deadlock free routes. The first scheme is called the *Shortest Widest* scheme. In this scheme, the routing between each pair of machines is determined as follows. First, Up*/Down* routing (the root node is randomly selected to build the tree for Up*/Down* routing) is applied to limit the paths that can be used between each pair of machines. After that, a shortest-widest heuristic is used to determine the path between machines. This heuristic determines the paths between machines one by one. At the beginning, all links are assigned a weight of 1. When a path is selected, the weight on each link in the path is increased by 1. For a given graph with weights, the shortest-widest heuristic tries to select the shortest path between two nodes (among all paths allowed by the Up*/Down* routing). When there are multiple such paths, the one with the smallest weight is selected. The second routing scheme is called the *Path Selection* scheme. In this scheme, the paths are determined as follows. First, Up*/Down* routing is applied to limit the paths that can be used between each pair of machines. After that, a k-shortest path routing algorithms [17] is used to find a maximum of 16 shortest paths (following the Up*/Down* routing rules) between each pair of nodes. Note that some pairs may not have 16 different shortest paths. After all paths are computed, a path selection algorithm [12] is applied to select one path for each pair of machines. The path selection algorithm follows the most loaded link first heuristic [12], which repeatedly removing paths that use the most loaded link in the network until only one path for each pair remains. It has been shown in [12] that the most loaded link first heuristic is effective in producing load balancing paths. Both the shortest widest scheme and the path selection scheme compute one path for each pair of machines. Paths computed with these two different schemes exhibit very different characteristics, which allows us to thoroughly investigate the effectiveness of the proposed LID assignment heuristics.

In computing the LIDs allocated for each node, LID mask control is taken into consideration. Each node is assigned a power of 2 LIDs: when k LIDs are required for destination d, the number of LIDs for d is counted as $2^{\lceil lg(k) \rceil}$.

A. Relative performance of LID assignment heuristics

The LID assignment heuristics evaluated include *greedy*, *split-merge/L* where the node used by the largest number of paths is selected first in the split phase, *split-merge/S* where the node used by the smallest number of paths is selected first, *color/L* that is the most split path first heuristic (paths that split with the largest number of other paths are colored first), and *color/S* that is the least split path first heuristic (paths that split with the least number of other paths are colored first). To save space, we will use notion *s-m/L* to represent *split-merge/L* and *s-m/S* to represent *split-merge/S*.

Table I depicts the performance of the heuristics when they are applied to the paths computed using the shortest widest scheme. The table shows the average of the total number of LIDs assigned to all machines. Each number in the table is the average of 32 random instances. We obtain the following observations from the experiments. First, the performance differences among the heuristics for the 16-switch configurations are very small. The performance difference between the best and the worst heuristics is less than 1%. The fact that five different heuristics compute minimal configuration sets in very different ways and yield similar performance suggests that other LID assignment schemes will probably have similar performance for the paths computed by the shortest-widest scheme on networks with a small number of switches. When the network is small, there are not many different shortest paths between each pair of nodes. Paths to the same destination tend to follow the same links with this path computation scheme. Hence, there are not many optimization opportunities in LID assignment. This observation is confirmed in the study of the absolute performance in the next sub-section: the performance of these heuristics for the paths computed with the shortest widest scheme is very close to optimal. Second, as the subnet becomes larger, the performance difference also becomes larger. For example, on the 64-switch configurations, the performance differences between the best and the worst heuristics are 8.4% for 128 machines, 5.5% for 256 machines, and 4.9% for 512 machines. When the network becomes larger, there are more different shortest paths between two nodes, which creates more optimization opportunities in LID assignment.

Among the proposed heuristics, the split-merge approach has a very similar performance to the greedy algorithm. Thus, the higher complexity in the split-merge approach cannot be justified. The most split path first heuristic (color/L) is consistently better than all other heuristics while the least split path first (color/S) is consistently worse than other heuristics. This indicates that color/L is effective for this problem while color/S is not. The trend is also observed when the path selection scheme is used to compute paths.

Table II shows the results for the paths computed by the path selection scheme. Each number in the table is the average (over

Conf.	Greedy	S-m/S	S-m/L	Color/S	Color/L
128/16	478.7	478.9	477.3	479.3	476.4
256/16	1044.3	1045.4	1041.5	1047.7	1039.2
512/16	2218.3	2220.1	2211.8	2220.4	2208.5
128/32	451.5	453.9	452.9	461.3	443.0
256/32	1078.8	1084.7	1079.0	1100.0	1062.4
512/32	2428.7	2440.2	2425.8	2461.0	2392.1
128/64	422.8	427.7	427.0	441.5	407.4
256/64	1015.5	1022.2	1019.3	1044.6	990.6
512/64	2325.8	2338.4	2330.1	2385.1	2274.4

TABLE I
THE AVERAGE OF THE TOTAL NUMBER OF LIDS ALLOCATED (SHORTEST WIDEST)

Conf.	Greedy	S-m/S	S-m/L	Color/S	Color/L
128/16	520.9	524.2	514.0	581.2	466.0
256/16	951.3	952.7	935.0	1062.6	851.2
512/16	1829.2	1852.8	1823.0	2038.7	1653.2
128/32	540.3	546.7	539.3	611.3	466.0
256/32	1006.7	1018.2	1002.2	1130.8	887.2
512/32	1904.0	1920.3	1895.7	2115.8	1688.7
128/64	528.0	541.1	530.5	599.4	460.5
256/64	1054.9	1092.9	1068.1	1197.9	921.4
512/64	2019.9	2075.4	2043.4	2278.6	1786.6

TABLE II
THE AVERAGE OF THE TOTAL NUMBER OF LIDS ALLOCATED (PATH SELECTION)

32 random instances) of the total number of LIDs allocated to all machines for each configuration. There are several interesting observations. First, the performance differences among different heuristics are much larger than the cases with the shortest widest scheme. On the 16-switch configurations, the performance differences between the best and the worst heuristics are 24.7% for 128 machines, 24.8% for 256 machines, and 23.3% for 512 machines. For larger networks, the differences are more significant. On the 64-switch configurations, the performance differences are 30.1% for 128 machines, 30.0% for 256 machines, and 27.5% for 512 machines. This indicates that for the paths computed with the path selection scheme, which are more diverse than those computed by the shortest-widest scheme, a good LID assignment heuristic significantly reduces the number of LIDs needed. The good news is that color/L consistently achieves the best performance in all cases, which indicates that this is a robust heuristic that performs well for different situations. Second, comparing the results for paths computed by the shortest widest routing (Table I) with those computed by path selection (Table II), we can see that when the number of machines is small (128 machines with 32 and 64 switches), the paths computed by the shortest widest scheme requires less LIDs to realize than the paths computed by the path selection scheme assuming the same LID assignment heuristic. However, when the number of machines is larger (256 and 512), the paths computed from the shortest-widest scheme requires more LIDs. This shows that path computation can have a significant impact on the LID requirement.

In summary, depending on the path computation method, LID assignment heuristics can make a significant difference in the number of LIDs required. The color/L heuristic consistently achieves high performance in different situations. The results also indicate that path computation methods have a significant impact on the LID requirement.

B. Absolute performance of the LID assignment heuristics

The previous section shows that the color/L heuristic is more effective than other heuristics. However, it is not clear how well the heuristic performs in comparison to optimal solutions. In this section, we compare the performance of the heuristics with optimal solution. The optimal solutions are obtained using the ILP formulation in Section IV. We use the open source ILP solver *lp_solve* version 5.5.0.11 [9] in the experiments. Due to the complexity in solving the ILPs, we consider smaller networks than those in the previous section: the systems we considered in this section consists of 16, 32, and 64 switches, and 64, 128, and 192 machines. *Lp_solve* is not able to give definitive answers in a reasonable amount of time for some problems. To limit the amount of time in the experiments, we set the time-out parameter for *lp_solve* to four hours for each ILP instance. The experiments are done on a cluster of Dell Poweredge 1950's, each with two 2.33GHz Intel quad-core Xeon E5345 processors and 8GB memory. If *lp_solve* cannot obtain solutions for an ILP in four hours, we ignore the topology and replace it with another random topology. The results for all of the schemes compared are obtained using the same set of topologies. Notice that for a topology with X machines, X solutions for $SD(G, d, R_d)$'s must be obtained to decide the LID requirement for the topology. The solution for each $SD(G, d, R_d)$ determines the LID requirement for one destination d. To obtain the solution for each $SD(G, d, R_d)$, we first use color/L to obtain the initial k and then try to solve $SD(G, d, R_d, k-1)$, $SD(G, d, R_d, k-2)$, and so on, until the solver decides that there is no solution for $SD(G, d, R_d, m)$. The answer to $SD(G, d, R_d)$ is then $m + 1$. Since the performance of $color/L$ is very close to optimal, in the majority of the cases, only one or two ILPs are solved to obtain the solution for $SD(G, d, R_d)$. For each case, the results are the average of 32 random topologies, which are not the same topologies as the ones in the previous subsection since *lp_solve* cannot handle some topologies in the previous subsection. We compare the results for the *greedy* heuristic, the simplest among all proposed heuristics, and *color/L*, our best performing heuristic. Similar to the study in the previous sub-section, we consider paths computed by the shortest-widest scheme and the path selection scheme.

Table III depicts the results of the heuristics and optimal solutions when the shortest widest scheme is used to compute the paths. For all cases, the optimal solutions are on average less than 5% better than the greedy algorithm and less than 1% better than color/L. This confirms our speculation that different heuristics make little difference on the LID assignment performance when the shortest widest routing algorithm is used to compute paths. This is because there are not many shortest paths between two nodes and the paths computed using this method tend to use the same links and do not have too many splits, which limits the optimization opportunities. As the number of switches increases, the performance of the greedy algorithm become worse compared to the optimal solution: for the 128 machine cases, the difference between the greedy heuristic and the optimal algorithm increases from 0.41% to 4.82% when the number of switches increases from 16 to 64. On the other hand, color/L shows much stable relative

Conf.	ILP	Greedy	Color/L	ILP over greedy	ILP over color/L
64/16	210.8	212.1	211.0	0.59%	0.06%
128/16	455.2	457.1	455.5	0.41%	0.06%
192/16	701.0	704.0	715.5	0.44%	0.08%
64/32	177.6	182.7	178.4	2.85%	0.42%
128/32	448.3	457.4	449.0	2.05%	0.17%
192/32	714.9	727.9	716.0	1.82%	0.15%
64/64	159.6	166.5	159.9	4.35%	0.24%
128/64	406.2	425.8	407.7	4.82%	0.37%
192/64	690.3	711.9	692.8	3.13%	0.37%

TABLE III
THE AVERAGE OF TOTAL LIDS ALLOCATED (SHORTEST WIDEST)

Conf.	ILP	Greedy	Color/L	ILP over greedy	ILP over color/L
64/16	215.1	244.5	219.8	13.69%	2.18%
128/16	400.7	451.1	407.0	12.56%	1.56%
192/16	575.7	647.1	582.6	12.41%	1.19%
64/32	225.2	263.4	231.1	16.96%	2.61%
128/32	436.2	510.3	447.3	16.98%	2.54%
192/32	617.7	723.7	634.6	17.15%	2.73%
64/64	222.2	256.3	226.6	15.36%	1.97%
128/64	437.5	509.4	448.1	16.43%	2.43%
192/64	646.4	754.8	664.6	16.76%	2.80%

TABLE IV
THE AVERAGE OF TOTAL LIDS ALLOCATED (PATH SELECTION)

performance in different situations. For example, for the 128 machine cases, the difference between color/L and the optimal algorithm increases from 0.06% to 0.37% when the number of switches increases from 16 to 64. This shows that the performance of color/L is very close to optimal.

Table IV shows the results for the heuristics and ILP when the path selection scheme is used to compute the paths. The performance difference between ILP and greedy is considerably larger than the cases with the shortest widest scheme. The optimal solution is much better than the greedy heuristic, the differences range from 12% in smaller configurations (64/16) to 17% for the larger configurations (128/32 and 64/32). Again, color/L shows much better performance. The optimal solution is only less than 3% better than color/L in all the configurations considered. This indicates that there is not much room to further improve color/L.

Color/L consistently performs well with very little room for further improvement: less than 1% when the widest-shortest scheme is used and less than 3% when the path selection scheme is used. Different path computation schemes may affect the effectiveness of different LID assignment heuristics, yet color/L consistently achieves close to optimal results for different system configurations regardless of the path computation method.

Since we discard the topologies that cannot be handled by *lp_solve*, the topologies are not totally random and the evaluation may not be fair. Next, we will examine the impacts of using such topologies. We compare key statistics on the ILP selected random topologies, which are the first 32 topologies that *lp_solve* gives exact solutions, with those on purely random topologies, which are the first 32 random topologies generated. Table V shows the average number of LIDs needed and the improvement of *color/L* over *greedy* with different routing schemes for various configurations. Two important observations are obtained. First, the ILP selected topologies tend to use a smaller number of LIDs. This means that *lp_solve* tends to give exact solutions to cases with a smaller number of LIDs. Second, the ILP selected topologies do not bias against the heuristics, as evidenced by the similar performance improvement ratio between *color/L* and *greedy* for both types of random topologies. For shortest widest routing, the average of the improvement ratio between color/L and greedy over all the nine configurations in Table IV is 2.01% for purely random topologies and 2.05% for ILP selected topologies. For path selection, the ratio is 13.3% for purely random topologies and 12.9% for ILP selected topologies.

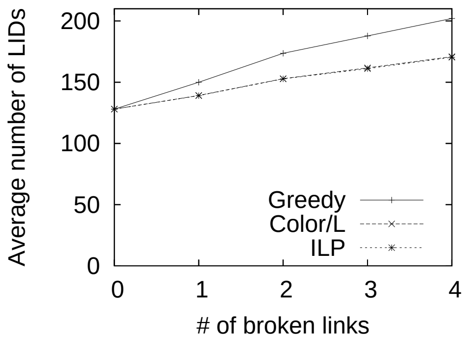
C. Performance of the heuristics on regular and near regular topologies

In this section, we examine the heuristics on regular topologies as well as regular topologies with a few broken links (near regular topologies). Three types of topologies and their variations are considered: two dimensional tori, three dimensional tori, and fat-trees. For the torus topologies, we assume that one machine is attached to each switch. The base routing algorithm is the dimension-order routing [4]. In a torus topology with broken links, all paths that are not affected by the broken links remain the same. For the paths that use one or more of the broken links, alternative paths are computed using the shortest widest algorithm without the Up*/Down* routing constraint. The broken links are randomly selected. We study the cases with 0, 1, 2, 3, and 4 broken links. For the fat-tree topology, we consider m-port n-tree [7]. The base routing algorithm for the fat-tree topology is a recently developed deterministic routing algorithm in [5]. Similar to the torus case, in a fat-tree topology with broken links, all paths computed by the base routing scheme that are not affected by the broken links remain the same. Alternative paths are computed using the shortest widest algorithm for the affected paths.

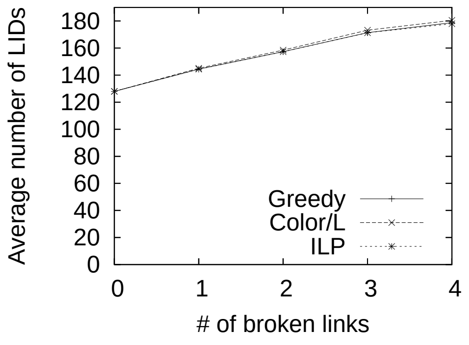
Fig. 10 shows the performance of the heuristics on the torus topology with zero and more broken links. Fig. 10 (a) contains the results for a two dimensional 16×8 torus, and Fig. 10 (b) contains the results for a three dimensional $8 \times 4 \times 4$ torus. Using the dimensional order routing, only one LID is needed to realize all paths to each destination in the tori (without broken links): there is no split in all paths, and all heuristics yield the same optimal results. As more links are broken, alternative paths yield more splits and more LIDs are needed to realize paths. In the 2D torus with broken links, the performance of *color/L* is very close to the optimal while the *greedy* heuristic is significantly worse. For the 3D torus, both greedy and coloring have similar performance as the optimal algorithm in all cases. The main reason for this difference is that in 2D torus, the average path length is much larger. A long path can potentially have many splits with other paths, which increases the optimization opportunities in LID assignment. As a result, *color/L* is much more effective than *greedy*. On the other hand, for the 3D torus, the path lengths are much shorter, which presents fewer optimization opportunities in LID assignment, resulting in both *greedy* and *color/L* performing similarly to the optimal scheme. The performance results for an 8-port 3-tree, which has 128 machines and 80 switches, are shown in Fig. 11: the results are similar to those on the 3D torus for the similar reason: the path length in the fat-tree is also short.

Conf.	Topo. type	Shortest widest			Path selection		
		Greedy	Color/L	Improvement	Greedy	Color/L	Improvement
64/16	random	214.0	213.0	0.50%	246.5	224.4	9.83%
	ILP selected	212.1	211.0	0.53%	244.5	219.8	11.26%
128/32	random	466.6	457.3	2.02%	556.9	483.8	15.13%
	ILP selected	457.4	449.0	1.88%	510.3	447.3	14.09%
192/64	random	739.3	720.3	2.63%	842.9	729.6	15.52%
	ILP selected	711.9	692.8	2.75%	754.8	664.6	13.57%

TABLE V
ILP SELECTED RANDOM TOPOLOGIES VERSUS PURELY RANDOM TOPOLOGIES



(a) two dimensional 16×8 torus



(b) three dimensional $8 \times 4 \times 4$ torus

Fig. 10. Performance on torus topologies with zero or more broken links

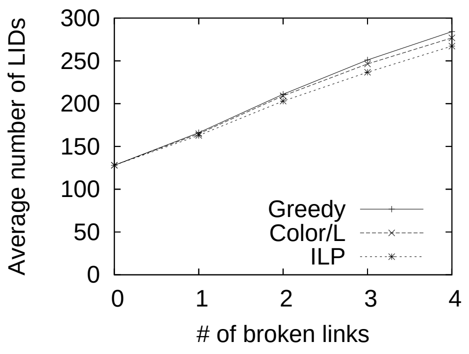


Fig. 11. Performance on a fat-tree topology (8-port 3-tree) with zero or more broken links

We have performed studies on other 2D and 3D tori and fat-trees, the results have a similar trend. These results indicate that the *color/L* heuristic is effective for regular and near regular topologies: its performance is consistently close to the optimal.

D. Overall performance of various path computation methods

We compare a new routing scheme that separates path computation from LID assignment with existing schemes for InfiniBand including destination renaming [8] and the SRD routing scheme [14]. The new routing scheme, called *separate*, uses the path selection scheme for path computation and color/L for LID assignment. SRD [14] selects paths such that one LID is sufficient to realize all paths to a destination. Hence, at the expense of the load balancing property of the paths computed, this method requires the least number of LIDs among all path computation schemes. Note that fully explicit routing [3] is a variation of SRD. The destination renaming [8] scheme uses a shortest path algorithm to select paths that follow Up*/Down* routing rules. It assigns LIDs as the paths are computed. All three schemes compute one path for each pair of machines.

We evaluate the performance of the routing methods with two parameters: (1) the number of LIDs required, and (2) the load balancing property of the paths. We measure the load balancing property as follows. We assume that the traffic between each pair of machines is the same and measure the *maximum link load* under such a traffic condition. In computing the maximum link load, we normalize the amount of data that each machine sends to all other machines to be 1. Under our assumption, the load of a link is proportional to the number of paths using that link. A good load balance routing should distribute traffic among all possible links and should have small maximum link load values in the evaluation.

Table VI shows the results for the three schemes on different configurations. The results are the average of 32 random instances for each configuration. As can be seen from the table, SRD uses one LID for each machine, and thus, it requires a smallest number of LIDs. However, it puts significant constraints on the paths that can be used and the load balancing property is the worst among the three schemes: the maximum link load of the SRD scheme is much higher than other schemes. For example, on 128/16, the maximum link load with SRD is 17% higher than that with *Separate*; on 512/64, it is 19% higher. Destination renaming, which is more comparable to our proposed new scheme, has a better load balancing property than SRD. Our proposed scheme, *Separate*, has the best load balancing property in all cases, which can be attributed to the effectiveness of the path selection algorithm [12]. Moreover, for reasonably large networks (256 and

conf.	SRD		Renaming		Separate	
	load	LIDs	load	LIDs	load	LIDs
128/16	4.34	128	3.84	477.8	3.70	466.0
256/16	8.65	256	7.52	1044.9	7.35	851.2
512/16	17.95	512	15.46	2213.3	14.91	1653.2
128/32	3.29	128	3.01	448.1	2.75	466.0
256/32	6.89	256	6.26	1079.2	5.8	887.2
512/32	14.71	512	13.24	2422.8	12.37	1688.7
128/64	3.29	128	3.01	420.0	2.75	460.5
256/64	6.15	256	5.72	1011.8	5.13	921.4
512/64	11.36	512	10.55	2323.4	9.54	1786.6

TABLE VI
THE MAXIMUM LINK LOAD AND THE NUMBER OF LIDS REQUIRED

512 machines), *separate* also uses a smaller number of LIDs than destination renaming. For the 512 machines/64 switches case, in comparison to destination renaming, the separate scheme reduces the maximum link load by 10.6% while decreasing the number of LIDs needed by 25.4%. This indicates that *separate* has much better overall performance than destination renaming: it reduces the maximum link load and uses a smaller number of LIDs simultaneously. This demonstrates the effectiveness of separating path computation from LID assignment, as well as the effectiveness of the color/L LID assignment heuristic.

VII. Conclusion

We investigate the LID assignment problem in InfiniBand networks. We show that this problem is NP-complete and develop an integer linear programming formulation for it. We further design LID assignment heuristics and show that the color/L heuristic is consistently the most effective heuristic among all proposed schemes in different situations. Moreover, depending on the path computation method, color/L, whose performance is very close to optimal, can be effective in reducing the number of LIDs required. We demonstrate that the techniques developed in this paper can be used with the existing schemes for finding dead-lock free and deterministic paths with good load balancing properties to obtain efficient routing schemes for InfiniBand networks. We must note that our proposed routing scheme, which separates path computation from LID assignment, is difficult to be applied in an incremental fashion. It is more suitable to be used to compute the initial network configuration than to deal with incremental network changes. In particular, our scheme is not as efficient in dealing with incremental network changes as other schemes designed for that purpose [3].

Acknowledgment

This work is supported in part by National Science Foundation (NSF) grants: CCF-0342540, CCF-0541096, and CCF-0551555.

References

[1] A. Bermudez, R. Casado, F. J. Quiles, T.M. Pinkston, and J. Duato, "Evaluation of a Subnet Management Mechanism for InfiniBand Networks." *Proc. of the 2003 IEEE International Conference on Parallel Processing* (ICPP), pages 117–124, Oct. 2003.

[2] A. Bermudez, R. Casado, F. J. Quiles, and J. Duato, "Use of Provisional Routes to Speed-up Change Assimilation in InfiniBand Netrworks." *Proc. of the 2004 IEEE International Workshop on Communication Architecture for Clusters* (CAC'04), page 186, April 2004.

[3] A. Bermudez, R. Casado, F. J. Quiles, and J. Duato, "Fast Routing Computation on InfiniBand Networks." *IEEE Trans. on Parallel and Distributed Systems*, 17(3):215-226, March 2006.

[4] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks, an Engineering Approach,* Morgan Kaufmann Publishers, 2003.

[5] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," the *IPDPS Workshop on Communication Architecture for Clusters* (CAC), pages 1-8, April 2007.

[6] M.Koibuchi, A. Funahashi, A. Jouraku, and H. Amano, "L-turn Routing: An Adaptive Routing in Irregular Networks." *Proc. of the 2001 International Conference on Parallel Processing* (ICPP), pages 383-392, Sept. 2001.

[7] X. Lin, Y. Chung, and T. Huang, "A Multiple LID Routing Scheme for Fat-Tree-Based Infiniband Networks." *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium* (IPDPS'04), p. 11a, April 2004.

[8] P. Lopez, J. Flich, and J. Duato, "Deadlock-Free Routing in InfiniBand through Destination Renaming." *Proc. of the 2001 International Conference on Parallel Processing* (ICPP), pages 427-434, Sept. 2001.

[9] lp_solve reference guide, http://lpsolve.sourceforge.net/5.5/

[10] InfiniBand^{TM} Trade Association, *Infi niBand^{TM} Architecture Specifi - cation*, Release 1.2, October 2004.

[11] O. Lysne, T. Skeie, S. Reinemo, I. Theiss, "Layered Routing in Irregular Networks," *IEEE Trans. on Parallel and Distributed Systems*, 17(1):51-65, January 2006.

[12] M. Koibuchi, A. Jouraku and H. Amano, "Path Selection Algorithm: The Stretegy for Designing Deterministic Routing from Alternative Paths." *Parallel Computing*, 31(1):117-130, 2005.

[13] J. C. Sancho, A. Robles, and J. Duato, "A New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks." *Proc. of the 4th Workshop on Communication Architecture and Applications for Network-Based Parallel Computing*, Jan. 2000.

[14] J. C. Sancho, A. Robles, and J. Duato, "Effective Strategy to Computing Forwarding Tables for InfiniBand Networks." *Proc. of the International Conference on Parallel Processing* (ICPP), pages 48-57, Sept. 2001.

[15] J. C. Sancho, A. Robles, and J. Duato, "Effective Methodology for Deadlock-Free Minimal Routing in InfiniBand Networks." *Proc. of the 2002 International Conference on Parallel Processing* (ICPP), pages 409-418, 2002.

[16] M. D. Schroeder, A. D. Birrell, M. Burrow, H. Murray, R. M. Needham, T. L. Rodeheffer, "Autonet: a High-speed Self-configuring Local Area Network Using Point-to-Point Links." *IEEE JSAC*, 9(8): 1318-1335, 1991.

[17] J. Y. Yen. "Finding the k shortest loopless paths in a network." *Management Science*, 17(11), July 1971.

[18] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem, "Oblivious Routing for Fat-Tree Based System Area Networks with Uncertain Traffic Demands." *ACM Sigmetrics*, pages 337-348, June 2007.

[19] E. W. Zegura, K. Calvert and S. Bhattacharjee, "How to Model an Internetwork." *IEEE Infocom '96*, pages 594-602, April 1996.

PLACE
PHOTO
HERE

Wickus Nienaber Wickus Nienaber obtained the B.S. and M.S. degrees in Computer Science from Florida State University in 2005 and 2007, respectively. He is currently a PhD student in the Computer Science Department at Florida State University. His research is in the area of parallel computer architectures in general with a focus on system area networks and interconnects.

PLACE PHOTO HERE

Xin Yuan Xin Yuan received his B.S. and M.S degrees in Computer Science from Shanghai Jiaotong University in 1989 and 1992, respectively. He obtained his PH.D degree in Computer Science from the University of Pittsburgh in 1998. He is currently an associate professor at the Department of Computer Science, Florida State University. His research interests include networking and parallel and distributed systems. He is a member of IEEE and ACM.

PLACE PHOTO HERE

Zhenhai Duan Zhenhai Duan received the B.S. degree from Shandong University, China, in 1994, the M.S. degree from Beijing University, China, in 1997, and the Ph.D. degree from the University of Minnesota, in 2003, all in Computer Science. He is currently an Assistant Professor in the Computer Science Department at the Florida State University. His research interests include computer networks and multimedia communications, especially Internet routing protocols and service architectures, the scalable network resource control and management in the Internet, and networking security. Dr. Duan is a co-recipient of the 2002 IEEE International Conference on Network Protocols (ICNP) Best Paper Award. He is a member of IEEE and ACM.