# Fast Routing Computation on InfiniBand Networks

Aurelio Bermúdez, *Member*, *IEEE*, Rafael Casado, *Member*, *IEEE*, Francisco J. Quiles, *Member*, *IEEE*, and José Duato, *Member*, *IEEE*

**Abstract**—The InfiniBand architecture has been proposed as a technology both for communication between processing nodes and I/O devices, and for interprocessor communication. Its specification defines a basic management infrastructure that is responsible for subnet configuration and fault tolerance. Each time a topology change is detected, new forwarding tables have to be computed and uploaded to devices. The time required to compute these tables is a critical issue, due to application traffic is negatively affected by the temporary lack of connectivity. In this paper, we show the way to integrate several routing algorithms, in order to combine their advantages. In particular, we merge a new proposal, characterized by its high computation speed but low efficiency, with a traditional one, slower but more efficient. Our goal is to provide new routes in a short period of time, minimizing the degradation mentioned before, and maintaining, at the same time, high network performance.

**Index Terms**—High-speed LANs, network management, network topology, routing protocols.

## 1 Introduction

The InfiniBand Architecture (IBA) [8], [14] is a new standard for high-speed I/O and interprocessor communication in clusters developed by the InfiniBand Trade Association (IBTA). IBA defines a technology for interconnecting processor nodes (hosts) and I/O nodes to form a system area network. Hosts and I/O units are interconnected using an arbitrary (possibly irregular) switched point-to-point network, instead of using a shared bus. Processor nodes can include several CPUs and memory modules, and they use one or several host channel adapters (HCAs) to connect to the switch fabric. I/O nodes can have any structure, from a simple console to a RAID subsystem. These devices use target channel adapters (TCAs) to connect to the fabric.

IBA subnets are managed in an autonomous way. There is a subnet management mechanism capable of assimilating any topology change without external intervention, guaranteeing service availability. The specification defines various subnet management entities, describing their functions and the structure of the control packets used to exchange information among them. An entity called subnet manager (SM) is in charge of discovering, configuring, activating, and maintaining the subnet. This entity exchanges subnet management packets (SMPs) with the subnet management agents (SMAs) present in every device. Fig. 1 shows an example of irregular subnet including these management entities.

The SM periodically sweeps the subnet searching for topology changes. Optionally, switch SMAs can inform the SM about the occurrence of a change by means of *trap* messages. In both cases, after a change is detected, the SM obtains the current subnet topology by exchanging SMPs with active devices. Once the exploration finishes, it uses the topological information collected to build the routes through the subnet. Finally, new forwarding tables (FTs) are distributed to the subnet switches, concluding the process.

During the entire assimilation process (topology discovery, FT computation, and distribution), there is a lack of network connectivity, negatively affecting to application traffic. Fig. 2 shows that the time required to compute new FTs is the most critical step. The methodology we have used to obtain these results is presented in the evaluation section. More details may be found in [2].

In this work, we present a new routing algorithm for IBA that focuses on reducing paths computation time, even if the resulting network performance is temporarily sacrificed. Also, we proceed to mix several routing algorithms (based on the same up*/down* directed graph) to combine their advantages.

The rest of the paper is organized as follows: First of all, Section 2 describes the IBA forwarding tables and the up*/down* routing algorithm and revises several techniques to obtain up*/down* routes in IBA. In Section 3, we detail our proposal to compute IBA routes, including its formal description and some proofs of convergence, connectivity, and deadlock-freedom. Also, this section describes how we merge routing algorithms in order to improve network performance. Then, in Section 4, we evaluate the performance of several pure and hybrid algorithms through several simulation results. Finally, Section 5 gives some conclusions.

## 2 Background: Up*/Down* Routing on IBA

### 2.1 IBA Forwarding Tables

IBA switches are responsible for relaying packets toward their destinations by using a DLID (destination local

- *A. Bermúdez, R. Casado, and F.J. Quiles are with the Department of Computer Science, Universidad de Castilla-La Mancha, 02071—Albacete, Spain. E-mail: {aurelio.bermudez, rafael.casado, francisco.quiles}@uclm.es.*
- *J. Duato is with the Department of Computer Engineering, Universidad Politécnica de Valencia, 40022—Valencia, Spain. E-mail: jduato@gap.upv.es.*

*Manuscript received 22 Sept. 2004; revised 25 Feb. 2005; accepted 15 May 2005; published online 25 Jan. 2006.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0243-0904.*

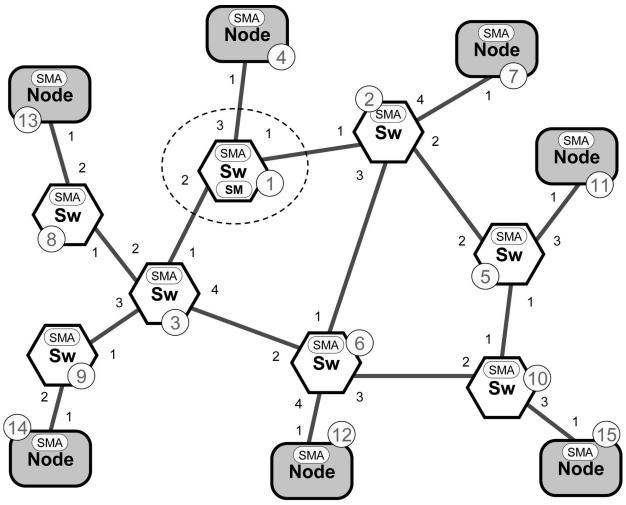


Fig. 1. Example of irregular subnet topology composed of eight switches and seven end nodes. The location of the SM is highlighted with a dashed oval. Circled numbers represent the local identifier (LID) assigned to each subnet device by the SM during the discovery process. Small numbers at the ends of the links represent switch and channel adapter port numbers.

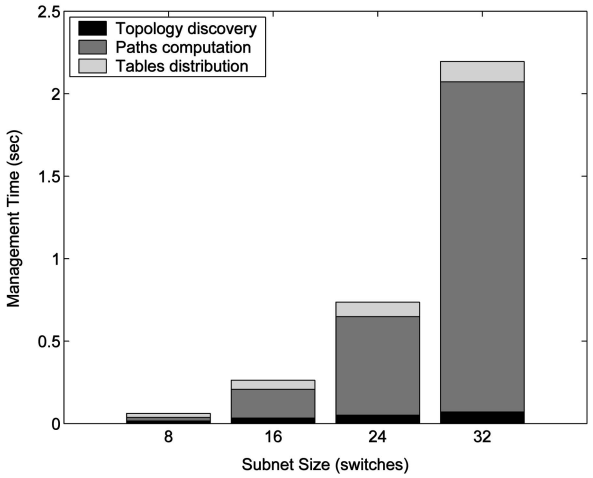


Fig. 2. Time required by the subnet management mechanism to update the subnet routes for a topology change consisting of a switch failure.

identifier) value in the packet header. In order to select the output port to be used, a switch must support one of two types of forwarding tables, the random forwarding table (RFT) and the linear forwarding table (LFT), shown in Fig. 3.

In an RFT, each entry includes an DLID/LMC (LID mask control) pair, which specifies a range of identifiers from DLID to $\mathrm{DLID} + 2^{\mathrm{LMC}} - 1$, and the output port associated to them. When a DLID is not found into the table, then the packet is relayed through a predefined default port. On the other hand, in a LFT, the DLID value is implicit to the position of the entry into the table. Also, this table can include up to 16K entries for multicast routing containing a 256-bits mask which specifies the set of output ports for a multicast packet.

During the distribution process, forwarding table entries are sent by the SM in 64-entry or 16-entry blocks (a block in each SMP), depending on the type of table supported by the destination switch (LFT or RFT).

## 2.2 Up*/Down* Routing

Up*/down* [15] is probably the most popular deadlock-free routing algorithm developed for irregular topologies. This algorithm configures the network as an acyclic directed graph with a single sink node. For each link, a direction is named *up* and the opposite one is named *down*. As an example, Fig. 4 shows a possible assignment of directions for the topology of Fig. 1. In this case, link directions have been assigned according to the distance to the SM of the device at the end of the link. There are several algorithms in the literature to build the directed graph; MDST [15], POST [10], DFS [11], and *partial discovery* [3] are some of them. Their detailed descriptions are out of the scope of this paper.

To avoid deadlocks, up*/down* states that legal routes cannot use a link in the *up* direction after having used one in the *down* direction. In Fig. 4, an example of legal route from node 7 to node 14 could be $7 \rightarrow 2 \rightarrow 1 \leftarrow 3 \leftarrow 9 \leftarrow 14$, but not $7 \rightarrow 2 \leftarrow 6 \rightarrow 3 \leftarrow 9 \leftarrow 14$. The reason is that an unallowed *down-up* transition was used at node 6.

It is relatively easy to compute a set of up*/down* routes without using *down-up* transitions if the technology considers the input port when routing packets. That is the case of Autonet [15] and Myrinet [5]. Autonet explicitly considers the input port into the routing tables. In the case of

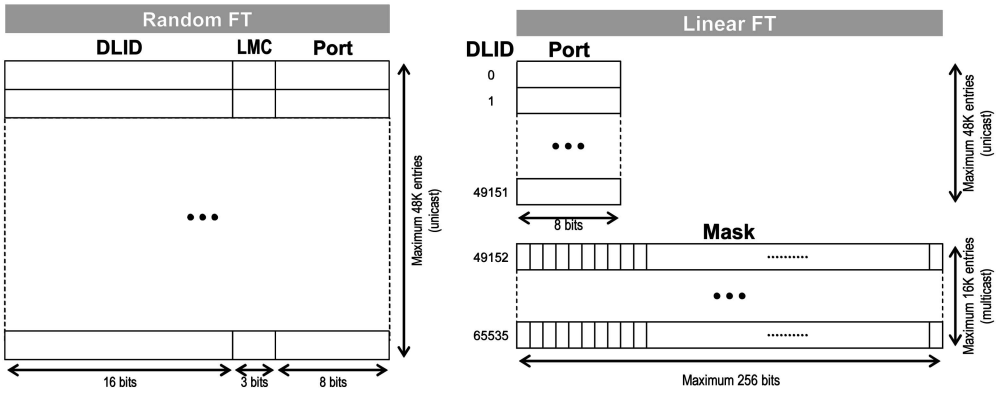


Fig. 3. Structure of IBA forwarding tables.

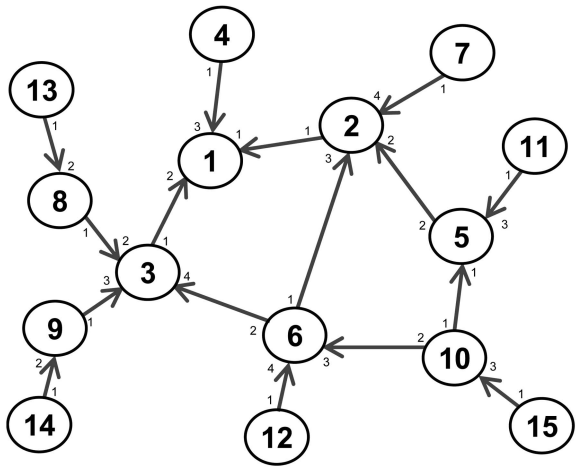


Fig. 4. Directed graph for the subnet shown in Fig. 1.

Myrinet, the input port is implicitly used into the source route. However, IBA does not take into account the input port when routing packets, and *down-up* transitions may be used if we consider routes independently and the directed graph is not balanced. As an example, suppose the situation shown in Fig. 5. The path from node $n2$ to node $n1$ through port 1 requires three hops, and the path through port 3 requires four hops. Therefore, the best choice would be to use the first route. Now, let us consider node $n3$, connected to $n2$ by means of a downward link. This node routes packets to $n1$ through $n2$. The consequence is that an unallowed *down-up* transition appears between ports 2 and 1 at node $n2$, and the network may lead to deadlock.

The solutions to these situations proposed in the literature [2], [4], [6], [7], [12], [13] can be classified into two categories. Some techniques sacrifice performance using paths longer than necessary, in order to guarantee deadlock-freedom. The alternative consists of differentiating paths in conflict by increasing resource utilization (as routing information, virtual channels, etc.). *Fully explicit routing* and *destination renaming* are examples of both categories, and they are described below.

### 2.3 Fully Explicit Routing

The *Fully Explicit Routing algorithm* (FERa) [2] is a Dijkstra-based routing algorithm that fits InfiniBand constrains. It takes its name from the fact that it computes all the FT entries required by each route. As we will see later, another proposal could take advantage of the utilization of default ports, reducing the amount of entries computed.

The algorithm works as follows: For each destination node $n_d$ in the subnet, the rest of subnet nodes are considered. This is done by performing a controlled flooding and preventing those hops that involve a forbidden *down-up* transition. During this flooding, a new table entry is added for each visited node $n_v$ (if it is a switch), considering the port used to reach $n_v$ as output port for $n_d$. Also, the length of the route is stored. Then, if a shorter route is later found, we may discard the first one because IBA routing is deterministic.

To guarantee that the combination of several routes does not generate *down-up* transitions, FERa proceeds as follows:

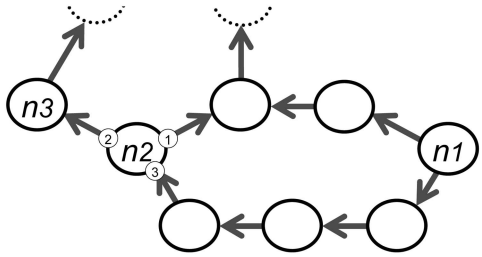


Fig. 5. Example of unallowed *down-up* transition when not considering the input port to route packets.

If, for a given destination, there is any output port in the *down* direction, it ignores all the routing options that imply the use of a link in the *up* direction, even if they lead to shorter paths. Then, in the example of Fig. 5, node $n2$ uses port 3 to reach $n1$.

Fig. 6 shows the sequence of table entries that are computed for the subnet shown in Fig. 1, assuming the directed graph shown in Fig. 4. When FERa is applied, the algorithm computes 120 forwarding table entries (15 entries for each subnet switch).

To conclude, let us analyze FERa complexity. As the well-known Dijkstra algorithm is $O(n^2)$, the complexity of FERa, when it is applied to every destination node, is $O(n^3)$.

### 2.4 Destination Renaming

*Destination Renaming* (DR) [7] encodes the information related to the input port in the destination identifier. Basically, the idea to solve situations as presented in Fig. 5 is to provide several LIDs for the same destination, in order to differentiate among the conflicting routes. Fig. 7 shows the same example, after renaming destination $n1$ at switch $n2$. Then, node $n2$ sends its own packets to $n1$ through the shortest route using DLID 20 and relays packets from node $n3$ to $n1$ without using a *down-up* transition. Destination renaming can be easily implemented on IBA networks due to the possibility of assigning multiple virtual LIDs to the same port (using the LMC value described before).

DR requires that a routing algorithm, which considers the input port (as happens in Autonet and Myrinet), computes all paths between any pair of hosts. After that, every path is sequentially checked, comparing, at each switch, if it presents a conflict with any other path already processed. If there is a conflict, then the destination node is renamed and new forwarding table entries are obtained.

Intuitively, we can see that DR requires much more resources (memory and computation time) than FERa with the goal of achieving better performance. To minimize this effect, DR is only applied to obtain routes between hosts. Routes with switches acting as source and/or destination should be obtained by other way.

## 3 Partially Implicit Routing

As we have described in the previous section, the FER and DR algorithms compute an individual route for each pair of subnet nodes. In this section, we present the *Partially Implicit Routing algorithm* (PIRa) [4], an alternative mechanism that reduces the amount of table entries to compute and, consequently, the global computation time. To achieve this,

| Node (DLID) | New table entries (LID: DLID → $P_{OUT}$) | | Node (DLID) | New table entries (LID: DLID → $P_{OUT}$) | | Node (DLID) | New table entries (LID: DLID → $P_{OUT}$) | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1: 1 → 0 | (0) | 6 | 6: 6 → 0 | (0) | 11 | 5: 11 → 3 | (1) |
| | 2: 1 → 1 | (1) | | 2: 6 → 3 | (1) | | 10: 11 → 1 | (2) |
| | 3: 1 → 1 | (1) | | 3: 6 → 4 | (1) | | 2: 11 → 2 | (2) |
| | 5: 1 → 2 | (2) | | 10: 6 → 2 | (1) | | 1: 11 → 1 | (3) |
| | 6: 1 → 1 | (2) | | 1: 6 → 1 | (2) | | 6: 11 → 1 | (3) |
| | 8: 1 → 1 | (2) | | 5: 6 → 2 | (2) | | 3: 11 → 1 | (4) |
| | 9: 1 → 1 | (2) | | 8: 6 → 1 | (2) | | 8: 11 → 1 | (5) |
| | 10: 1 → 1 | (3) | | 9: 6 → 1 | (2) | | 9: 11 → 1 | (5) |
| 2 | 2: 2 → 0 | (0) | 7 | 2: 7 → 4 | (1) | 12 | 6: 12 → 4 | (1) |
| | 1: 2 → 1 | (1) | | 1: 7 → 1 | (2) | | 2: 12 → 3 | (2) |
| | 5: 2 → 2 | (1) | | 5: 7 → 2 | (2) | | 3: 12 → 4 | (2) |
| | 6: 2 → 1 | (1) | | 6: 7 → 1 | (2) | | 10: 12 → 2 | (2) |
| | 3: 2 → 1 | (2) | | 3: 7 → 1 | (3) | | 1: 12 → 1 | (3) |
| | 10: 2 → 1 | (2) | | 10: 7 → 1 | (3) | | 5: 12 → 2 | (3) |
| | 8: 2 → 1 | (3) | | 8: 7→ 1 | (4) | | 8: 12 → 1 | (3) |
| | 9: 2 → 1 | (3) | | 9: 7 → 1 | (4) | | 9: 12 → 1 | (3) |
| 3 | 3: 3 → 0 | (0) | 8 | 8: 8 → 0 | (0) | 13 | 8: 13 → 2 | (1) |
| | 1: 3 → 2 | (1) | | 3: 8 → 2 | (1) | | 3: 13 → 2 | (2) |
| | 8: 3 → 1 | (1) | | 1: 8 → 2 | (2) | | 1: 13 → 2 | (3) |
| | 9: 3 → 1 | (1) | | 9: 8 → 1 | (2) | | 9: 13 → 1 | (3) |
| | 6: 3 → 2 | (1) | | 6: 8 → 2 | (2) | | 6: 13 → 2 | (3) |
| | 2: 3 → 1 | (2) | | 2: 8 → 1 | (3) | | 2: 13 → 1 | (4) |
| | 10: 3 → 2 | (2) | | 10: 8 → 2 | (3) | | 10: 13 → 2 | (4) |
| | 5: 3 → 2 | (3) | | 5: 8 → 2 | (4) | | 5: 13 → 2 | (5) |
| 4 | 1: 4 → 3 | (1) | 9 | 9: 9 → 0 | (0) | 14 | 9: 14 → 2 | (1) |
| | 2: 4 → 1 | (2) | | 3: 9 → 3 | (1) | | 3: 14 → 3 | (2) |
| | 3: 4 → 1 | (2) | | 1: 9 → 2 | (2) | | 1: 14 → 2 | (3) |
| | 5: 4 → 2 | (3) | | 8: 9 → 1 | (2) | | 8: 14 → 1 | (3) |
| | 6: 4 → 1 | (3) | | 6: 9 → 2 | (2) | | 6: 14 → 2 | (3) |
| | 8: 4 → 1 | (3) | | 2: 9 → 1 | (3) | | 2: 14 → 1 | (4) |
| | 9: 4 → 1 | (3) | | 10: 9 → 2 | (3) | | 10: 14 → 2 | (4) |
| | 10: 4 → 1 | (4) | | 5: 9 → 2 | (4) | | 5: 14 → 2 | (5) |
| 5 | 5: 5 → 0 | (0) | 10 | 10: 10 → 0 | (0) | 15 | 10: 15 → 3 | (1) |
| | 10: 5 → 1 | (1) | | 5: 10 → 1 | (1) | | 5: 15 → 1 | (2) |
| | 2: 5 → 2 | (1) | | 6: 10 → 3 | (1) | | 6: 15 → 3 | (2) |
| | 1: 5 → 1 | (2) | | 2: 10 → 2 | (2) | | 2: 15 → 2 | (3) |
| | 6: 5 → 1 | (2) | | 3: 10 → 4 | (2) | | 3: 15 → 4 | (3) |
| | 3: 5 → 1 | (3) | | 1: 10 → 1 | (3) | | 1: 15 → 1 | (4) |
| | 8: 5 → 1 | (4) | | 8: 10 → 1 | (3) | | 8: 15 → 1 | (4) |
| | 9: 5 → 1 | (4) | | 9: 10 → 1 | (3) | | 9: 15 → 1 | (4) |

Fig. 6. Sequence of steps required to compute a complete set of FERa up*/down* routes for the subnet shown in Fig. 1. Table entries are represented by means of the LID of the switch where the entry is stored, a destination LID, the output port to use for that destination (a value of 0 means that the switch itself is the destination of the packet and, therefore, it must be delivered to the internal management port), and the distance to the destination node (in brackets).

PIRa groups several routes in each switch by taking advantage of the existence of a default port. As we have seen, the default port is used to deliver an incoming packet when there is no information in the switch forwarding table related to its DLID.

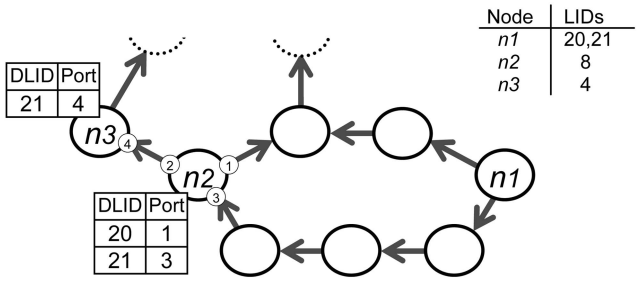


Fig. 7. Example of destination renaming.

### 3.1 Preliminary Definitions and Assumptions

Let $G(N, A)$ be the up*/down* directed graph associated to the subnet topology. Let $N$ be the set of nodes (switches and hosts) in the subnet. Let $A$ be a set of edges that represent the links between nodes. For each edge, its direction corresponds to the up*/down* direction assignment for the corresponding link. An edge in $A$ from node $n_x$ to node $n_y$ will be denoted as $(n_x, n_y)$. We assume that, if $(n_x, n_y) \in A$, then $(n_y, n_x) \notin A$. By definition, $G$ is an acyclic directed graph containing only one sink node.

Each node has a set of ports connecting to other nodes. Let $P$ be the set of ports in the subnet. We will denote as $p_x^y \in P$ the port of node $n_x$ connected to node $n_y$. Then,

$\forall(n_x, n_y) \in A, p_x^y, p_y^x \in P$. Additionally, every node $n_x$ has an internal (management) port denoted as $p_x \in P$. For convenience, we will assign a direction to each port (except for internal ports), which corresponds to the up*/down* direction assignment for the attached link. Let $D(p)$ be the direction of a port $p$. If $(n_x, n_y) \in A$, then we define $D(p_x^y) = DOWN$, and $D(p_y^x) = UP$, that is, a port direction is *down* if it is attached to the *down* end the link, and vice versa.

Let $FT(n_x, n_y)$ be the forwarding table entry at node $n_x$ containing the output port to reach node $n_y$. Initially, $\forall n_x, n_y \in N, FT(n_x, n_y)$ is not assigned. Also, let $DEF(n_x)$ be the default routing port at node $n_x$. In the same way, $\forall n_x \in N, DEF(n_x)$ is not assigned.

### 3.2 The Algorithm

The base of PIRa is the following one: If we consider the directed graph that models the subnet, a valid up*/down* route includes zero or more upward links (i.e., in the *up* direction) followed by zero or more downward links. As all upward segments converge at the sink node of the directed graph, we can define them by using the default port in each switch. On the other hand, downward segments are defined using explicit table entries.

The algorithm requires an exploration of the directed graph, starting from the sink node. In each step, the process selects a node $n1$ for which all the upward links are connected to previously visited nodes. If $n1$ is a switch, we assign as the default port at $n1$ one of the ports that connects it to one of the previously visited nodes (say, $n2$). Next, a new table entry must be added at $n1$ for each additional neighbor that has already been visited (reachable through upward links). Finally, the process considers inserting a table entry at each previously visited switch $n3$, which will be used to reach $n1$. This entry selects as output port the one that $n3$ uses to reach $n2$, except when $n3$ and $n1$ are neighbors. Note that it is not necessary to add a new table entry for a switch when the output port selected is equal to the switch default port. In particular, when $n3$ does not have any entry to reach $n2$ (i.e., it uses the default port), no new entry is added.

Fig. 8 shows the sequence of table entries that are computed by PIRa for the subnet shown in Fig. 1, assuming the directed graph shown in Fig. 4. Fig. 9 shows the set of visited nodes in each step. Note that, in this case, in addition to seven default ports, the algorithm only computes 50 table entries. This is a small amount compared to the 120 entries required by FERa over the same scenario.

Next, we propose a formal definition for the PIR algorithm:

Let $PREV(n_x)$ be the set of neighbors of $n_x$ such that the links connecting those nodes to $n_x$ have their *down* end attached to a port in $n_x$, that is, $\forall n_y \in N, n_y \in PREV(n_x)$ iff $D(p_x^y) = DOWN$. This set contains the neighbor nodes of $n_x$ that have been visited before visiting $n_x$ when computing the routes.
Let $X^i$ be the set of explored nodes after step $i$.
**Step 1**
$X^1 =$ {sink}
**Step *i***
Find $n_x \notin X^{i-1}$ such that $PREV(n_x) \subset X^{i-1}$.

| LID considered | Default port | New table entries (LID: DLID → $P_{OUT}$) | |
|---|---|---|---|
| 1 | - | 1: 1 → 0 | |
| 2 | 1 | 2: 2 → 0 | 1: 2 → 1 |
| 3 | 1 | 3: 3 → 0 | 1: 3 → 2 |
| 4 | n/a | 1: 4 → 3 | |
| 5 | 2 | 5: 5 → 0<br>1: 5 → 1 | 2: 5 → 2 |
| 6 | 2 | 6: 6 → 0<br>6: 2 → 1<br>1: 6 → 2 | 2: 6 → 3<br>3: 6 → 4 |
| 7 | n/a | 1: 7 → 1<br>2: 7 → 4 | 6: 7 → 1 |
| 8 | 1 | 8: 8 → 0<br>1: 8 → 2 | 3: 8 → 2 |
| 9 | 1 | 9: 9 → 0<br>1: 9 → 2 | 3: 9 → 3 |
| 10 | 2 | 10: 10 → 0<br>10: 5 → 1<br>1: 10 → 2<br>2: 10 → 3 | 3: 10 → 4<br>5: 10 → 1<br>6: 10 → 3 |
| 11 | n/a | 1: 11 → 1<br>2: 11 → 2 | 5: 11 → 3<br>10: 11 → 1 |
| 12 | n/a | 1: 12 → 2<br>2: 12 → 3 | 3: 12 → 4<br>6: 12 → 4 |
| 13 | n/a | 1: 13 → 2<br>3: 13 → 2 | 8: 13 → 2 |
| 14 | n/a | 1: 14 → 2<br>3: 14 → 3 | 9: 14 → 2 |
| 15 | n/a | 1: 15 → 2<br>2: 15 → 3<br>3: 15 → 4 | 5: 15 → 1<br>6: 15 → 3<br>10: 15 → 3 |

Fig. 8. Sequence of steps required to compute a complete set of PIRa up*/down* routes for the subnet shown in Fig. 1. Table entries are represented using the criteria applied in Fig. 6.

Add $n_x$ to the set of explored nodes, that is,
$X^i = X^{i-1} \cup \{n_x\}$.
Randomly select a node $n_f \in PREV(n_x)$ and set $DEF(n_x) = p_x^f$. Let us call $n_f$ the *father* of $n_x$.
Update the forwarding table of $n_x$ to reach explored neighbors, $\forall n_y \in PREV(n_x), n_y \neq n_f, FT(n_x, n_y) = p_x^y$.
Update the forwarding tables for the explored neighbors of $n_x, \forall n_y \in PREV(n_x), FT(n_y, n_x) = p_y^x$.
Update forwarding tables for the rest of explored nodes, $\forall n_y \in X^{i-1}$ such that
$n_y \notin PREV(n_x), FT(n_y, n_x) = FT(n_y, n_f)$.

As described above, each step of PIRa selects a node not selected before, which interacts with every previously selected node. Then, the total number of required interactions is $\sum_{i=1}^{n} i - 1 = \frac{n(n-1)}{2}$ and, consequently, the complexity of PIRa is $O(n^2)$.

### 3.3 Formal Proof

In what follows, we are going to prove that the proposed algorithm to compute forwarding table entries converges (i.e., it is able to explore the entire subnet) and delivers a set of routes that are collectively connected (i.e., they provide a path from every node to every other node in the subnet) and deadlock-free.

**Lemma 1 (Convergence).** $X^{card(N)} = N$.

**Proof.** We have to prove that, at each Step $i$, if $X^{i-1} \neq N$, it is possible to find $n_x \notin X^{i-1}$ such that $PREV(n_x) \subset X^{i-1}$.

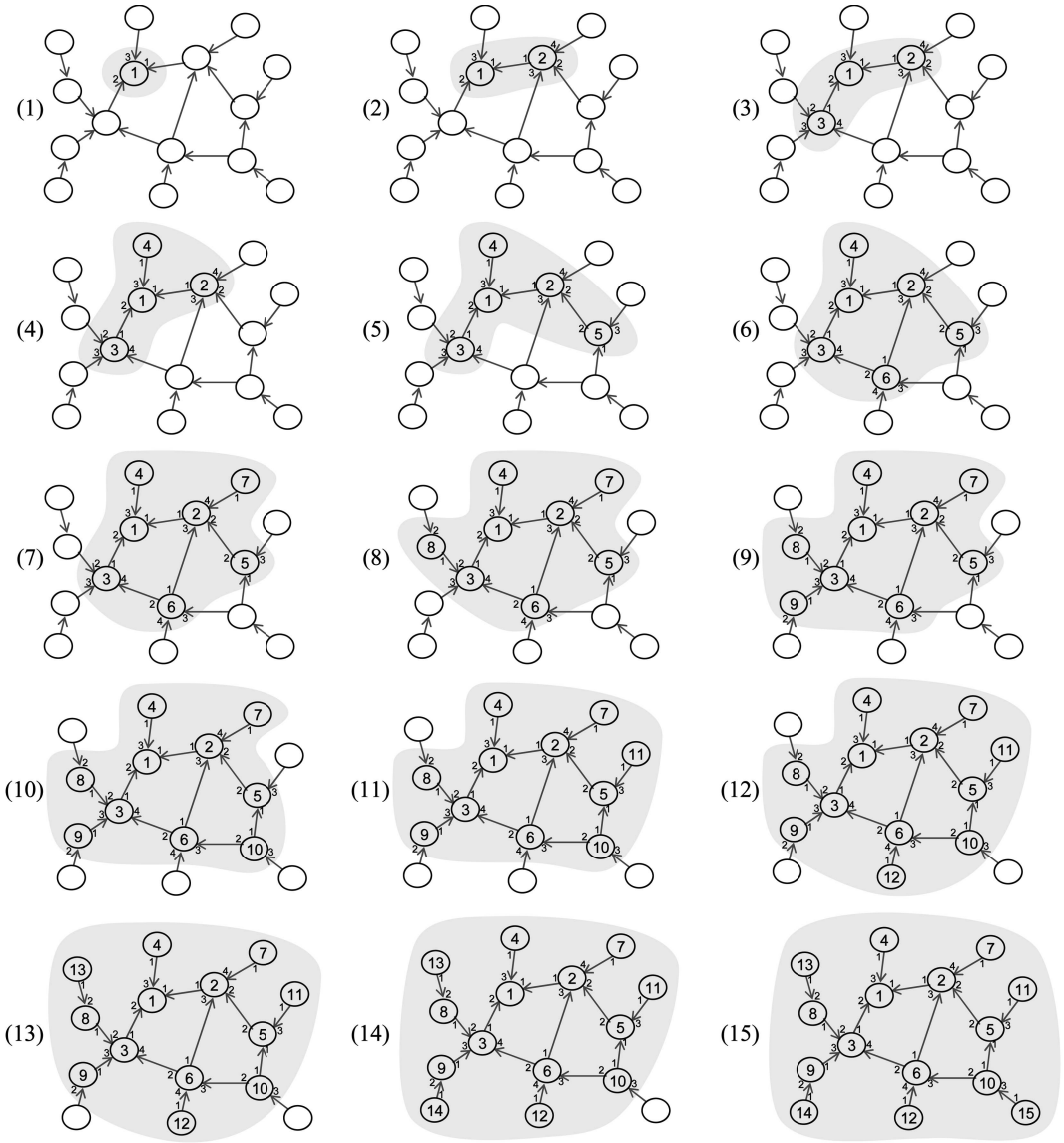


Fig. 9. Example of PIRa flooding. Steps are shown in brackets.

We proceed by contradiction. Assume that $\forall n_x \notin X^{i-1}$, $PREV(n_x) \subset X^{i-1}$. Then, $\exists n_y \in PREV(n_x)$ such that $n_y \notin X^{i-1}$. If $n_y$ satisfies that $PREV(n_y) \subset X^{i-1}$, we have found a node to explore at Step $i$ and we are done. Otherwise, we continue the search considering $n_y$ instead of $n_x$. By proceeding in this iterative way, and taking into account that $G$ is finite, in a finite number of steps we will find:

- A node $n_z$ satisfying that $n_z \notin X^{i-1}$ and $PREV(n_z) \subset X^{i-1}$.
- A sink node not included in $X^{i-1}$. This option contradicts the initial assumption about the existence of only one sink node in $G$, which was included in $X^{i-1}$ in the first iteration.
- A previously analyzed node. This contradicts the initial assumption about the inexistence of cycles in $G$.

Therefore, at each Step $i$, if $X^{i-1} \neq N$, it is possible to find $n_x \notin X^{i-1}$ such that $PREV(n_x) \subset X^{i-1}$ and add $n_x$ to the set of explored nodes, that is, $X^i = X^{i-1} \cup \{n_x\}$. This implies that, after $card(N)$ steps, we will have $X^{\mathrm{card}(N)} = N$. □

**Lemma 2 (Connectivity).** *The PIR algorithm provides a path between every pair of nodes.*

**Proof.** Taking into account Lemma 1, we can prove connectivity by showing that, at every Step $i$, there is connectivity among the already explored nodes. We proceed by induction, proving connectivity to $X^1, X^2 \ldots$

$X^1 = \{sink\}$. Paths are not required in this case.

$X^2 = \{sink, n_1\}$, where $n_1$ is a neighbor of the sink node. The path between both nodes is set by definition using the edge connecting them.

Let us assume the connectivity of $X^{i-1}$. Now, we have to prove the connectivity of $X^i = X^{i-1} \cup \{n_x\}$. For each neighbor of $n_x$ belonging to $X^{i-1}$, a path is established among them through the edge that connects it to $n_x$. For the rest of nodes in $X^{i-1}$, the algorithm establishes connectivity from any of those nodes to $n_x$ by using the already existing path to the father of $n_x$ (regardless of whether such a path uses default ports or not) and adding the edge

between this node and $n_x$. Note that the edge from the father of $n_x$ to $n_x$ is always a *down* link and, thus, it does not contradict the up*/down* routing rules. Similarly, the algorithm establishes connectivity from $n_x$ to the rest of nodes in $X^{i-1}$ by using the already existing path from the father of $n_x$ to the corresponding node (regardless of whether such a path uses default ports or not) and using the default port at $n_x$ to reach its father. □

**Lemma 3 (Deadlock Freedom).** *The set of routes generated by PIRa is deadlock-free.*

**Proof.** Taking into account that $G$ is an acyclic directed graph and that the up*/down* routing algorithm is deadlock-free [10], it is enough to prove that a forbidden *down-up* transition cannot occur.

We proceed by contradiction. Let us assume that a *down-up* transition exists. We denote the implied nodes as $n_x \leftarrow n_y \rightarrow n_z$, where arrows indicate link direction assignment. By definition, for some Step $i$, $n_y$ was added to $X^i$ after adding $n_x$ and $n_z$. Before adding $n_y$ to $X^i$, connectivity between $n_x$ and $n_z$ was guaranteed by Lemma 2. Therefore, $n_x$ and $n_z$ will not use $n_y$ to reach each other, and paths between them will not use that forbidden *down-up* transition. □

### 3.4 Hybrid Routing

As we will see in the evaluation section, PIRa is faster than FERa and DR at the expense of performance. The reason is that this algorithm favors congestion close to the root of the up*/down* directed graph. Of course, it is desirable to maintain both speed and performance at the same time, and mixing these routing algorithms may be an easy way to achieve it.

The basic idea is, first, we use PIRa to quickly obtain a set of valid subnet routes after the occurrence of a topology change. After that, we could add an additional step to the change assimilation process in order to restore network performance. In this way, once a "provisional" set of PIRa routes have been computed and distributed to switches, and subnet traffic has been reactivated, the management mechanism could use either FERa or DR to obtain a better set of "final" routes.

Additionally, final routes could be dynamically distributed (i.e., without stopping user traffic) in a deadlock-free manner. The reason is that provisional and final routes can coexist without producing deadlocks due to both sets of routes use a subset of the routes provided by the original up*/down* routing algorithm.

## 4 Performance Evaluation

In this section, we compare the three routing approaches described before (FERa, DR, and PIRa). We have analyzed their individual properties and the benefits obtained when they are mixed, attending to the methodology described in Section 3.4. We have studied their behavior when the change assimilation process is being performed, as well as the final network performance provided. The evaluation has been performed using simulation techniques. Before showing and analyzing the results we have obtained, we briefly describe the simulation methodology.

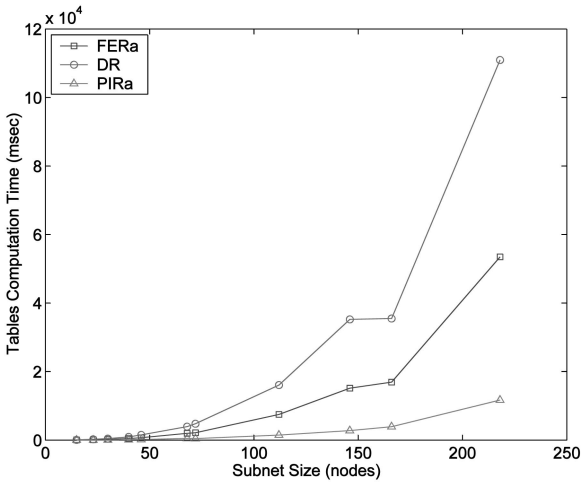


Fig. 10. Time required to compute a set of routes for different subnet sizes. The values have been empirically obtained by executing the computation algorithms on an Intel Pentium III (1.06 GHz) processor.

### 4.1 Simulation Methodology

Our model embodies key physical and link layer features of IBA, allowing the simulation of various IBA-compliant network designs. Also, it incorporates the subnet management entities and packets defined in the specification. To develop it, we have used the *OPNET Modeler* [9] simulation software. The current model is composed of IBA links, 4-port fully demultiplexed switches, and end nodes containing a HCA (hosts). See [1] for more detail.

We have evaluated randomly generated irregular subnets with 8, 16, 24, 32, 48, and 64 switches, assuming that there is at least a host connected to each switch, if a port is available. Also, not all switch ports are connected. All the plots presented here correspond to 1X links; however, results for different link bandwidths are almost identical. Logically, differences are clearly appreciated when we measure packet latency.

In all cases, the amount of operational data virtual lanes (VLs) per subnet port is 2 (VL0 and VL1). The size of the input and output buffers associated to each VL is 4,096 bytes. VL mapping and link arbitration schemes are not relevant for this work. In this case, for service level (SL) to VL mapping, a cyclic assignment of VLs is considered. Also, physical links are assigned to data VLs using a round-robin strategy.

We have considered a packet maximum transfer unit (MTU) of 256 bytes (the minimum MTU value allowed by the IBA specification). The packet generation rate is Poisson, and we have used several packet destination distributions (uniform, bit-reversal, matrix transpose, and perfect-shuffle). The SL value (from 0 to 15) is randomly generated for each packet. The traffic load applied is different for each subnet topology, varying from low loads to saturation.

For each simulation run, after a transient period, we have programmed a topology change, consisting of the failure of a switch or link. The experiment is repeated for each switch and link in the subnet, and average values are shown in the plots. In order to detect the change as quickly as possible, traps support is active in all subnet switches. The simulation is stopped once the topology change has been completely assimilated.

Next, we analyze the time spent in the computation of the routing tables. After that, we show the performance

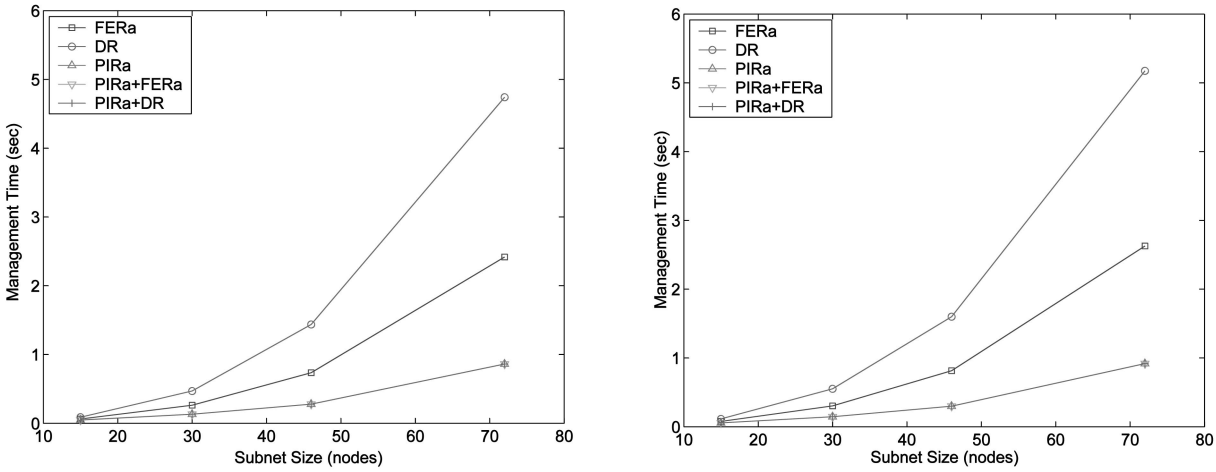


Fig. 11. Time required by the subnet management mechanism to update the subnet routes for a change consisting of a switch or link failure. Uniform traffic. (a) Switch failure and (b) link failure.

degradation produced by the execution of the change assimilation process. Finally, we evaluate network performance after the topology change.

### 4.2 Routing Table Computation Time

Fig. 10 shows the time required by the analyzed computation algorithms to build a complete set of subnet forwarding tables. In general, PIRa works about five times faster than FERa, and 10 times faster than DR. Also, as subnet size increases, disparities become more noticeable due to the significant differences in the complexity of the algorithms.

Next, Fig. 11 shows the time required by the routing algorithms to compute the set of routes, integrated between the subnet exploration task and the forwarding table distribution task. Results correspond to a topology change consisting of a switch (Fig. 11a) and link (Fig. 11b) failure, respectively. Exploration time has been measured from the moment in which the change is detected, and it is exactly the same for all the mechanisms. In the plots, "PIRa + FERa" and "PIRa + DR," legends refer to the hybrid management mechanisms introduced in Section 3.4, in which the distribution of PIRa routes is followed by the computation and distribution of FERa or DR ones.

Comparing this figure with Fig. 10, we can see as the time required to compute new forwarding tables determines the time consumed by the entire management process. Therefore, DR is the algorithm that spends more time managing the subnet, followed by FERa. The time required by PIRa to provide a valid set of new routes is the shortest one, and it is independent of if it is followed by a second computation process or not.

Moreover, we may appreciate in the figure that link failures require slightly longer reconfigurations than switch failures. The reason is that the remaining topology is bigger in the first case and reconfiguration time is high dependent of this parameter (as shown in Fig. 10). In fact, a link failure does not imply the faulty of the two switches connected to

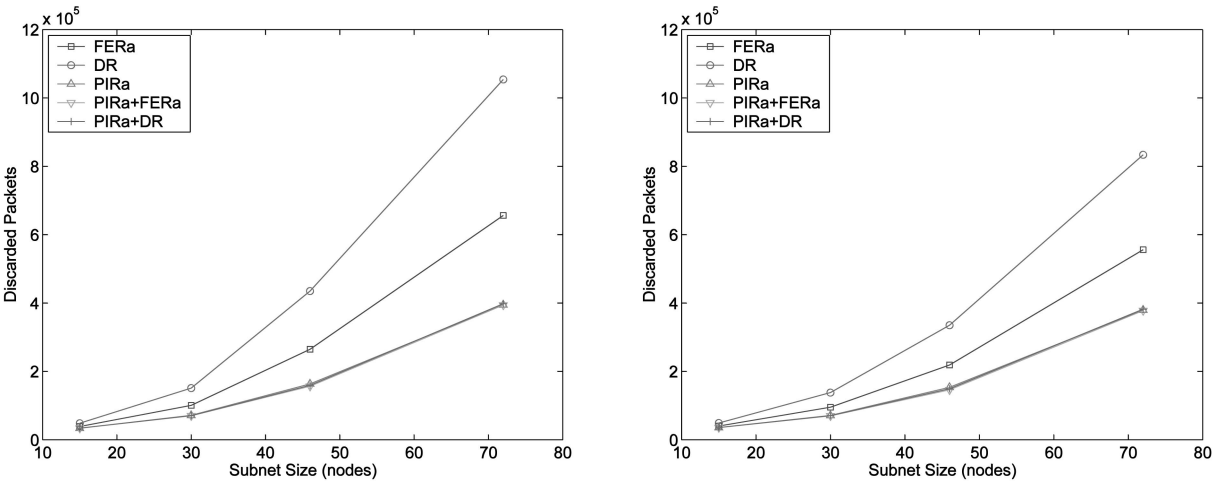


Fig. 12. Amount of packets discarded during the change assimilation. Uniform traffic. (a) Switch failure and (b) link failure.

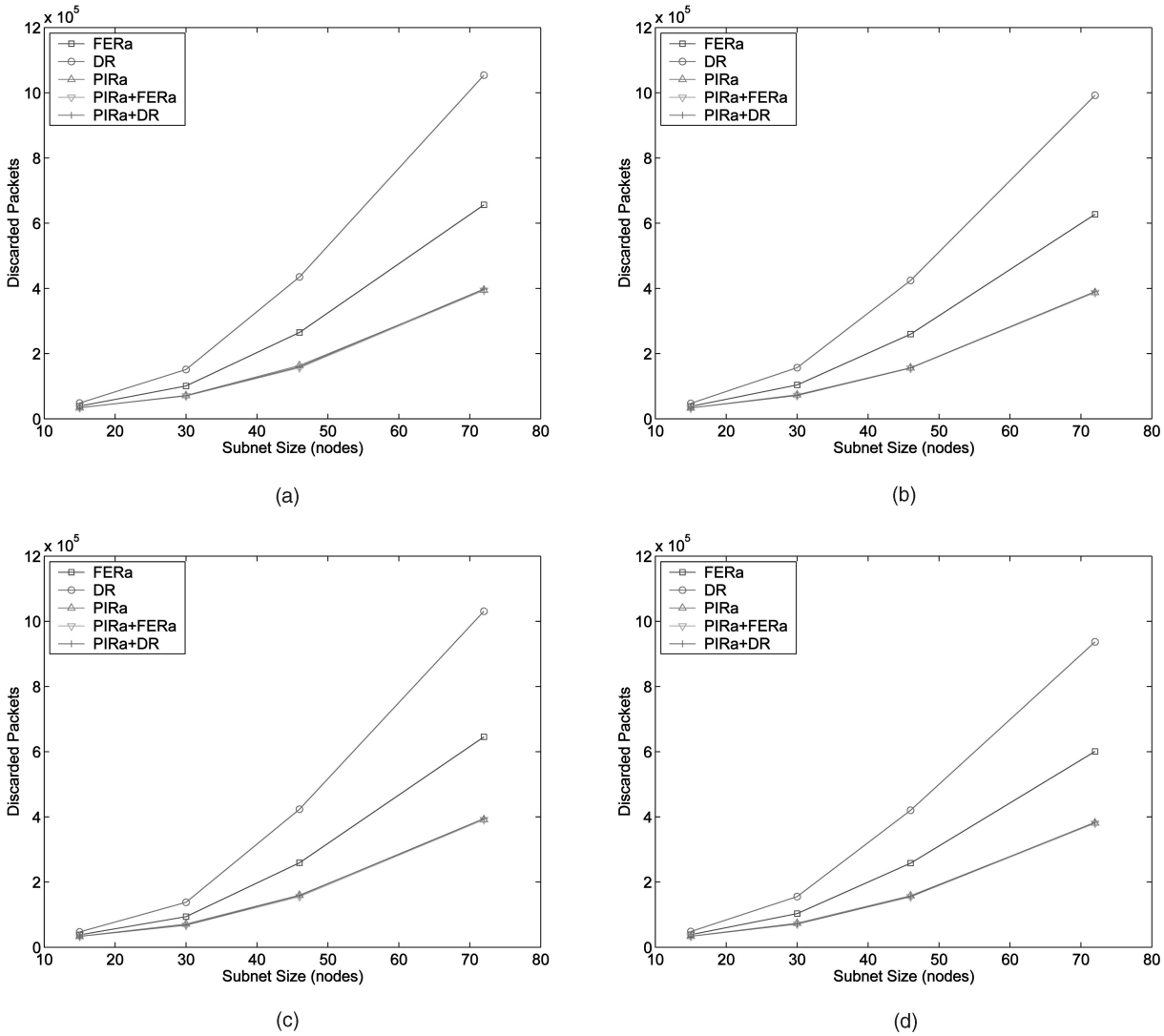


Fig. 13. Number of packets discarded during the change assimilation consisting of a switch failure. (a) Uniform, (b) bit-reversal, (c) matrix transpose, and (d) perfect-shuffle.

it; whether the switch failure implies the fault of all the links connected to it. These differences are more appreciable when considering super-connected topologies, with high switch degrees.

### 4.3 Performance Degradation during the Execution

Fig. 12 shows the amount of (link level) data packets that are discarded during the assimilation of a switch or link failure. In any case, discarded packets are not reinjected into the network, assuming that this function is performed by an upper layer mechanism.

As we can see, the fast computation of PIRa benefits the entire management process independently of if it is followed by FERa or DR, considerably reducing the total amount of discarded packets. Differences are more noticeable for large subnets. Also, we can appreciate as the execution of FERa or DR after PIRa does not imply an important additional packet discarding. The reason is that FERa (or DR) forwarding tables are distributed without stopping application traffic, as mentioned in Section 3.4.

Next, Fig. 13 shows packet discarding due to a switch failure in function of different traffic patterns. Only slight differences are appreciated. We may conclude that the particular traffic pattern applied is not relevant, and packet discarding mainly depends on time spent and network load. For this reason, from now on, only uniform traffic is applied.

### 4.4 Performance Evaluation after the Topology Change

Fig. 14 shows network performance (latency and throughput) provided by PIRa, FERa, DR, and their combinations, in function of subnet size. Curiously, DR provides better performances than FERa only in a few cases, highlighting the importance of obtaining well balanced routes instead of the shortest ones. As we may expect, PIRa provides the lowest performances. Obviously, the final performance for the hybrid mechanisms (PIRa + FERa and PIRa + DR) is very similar to the obtained when FERa and DR are executed alone.

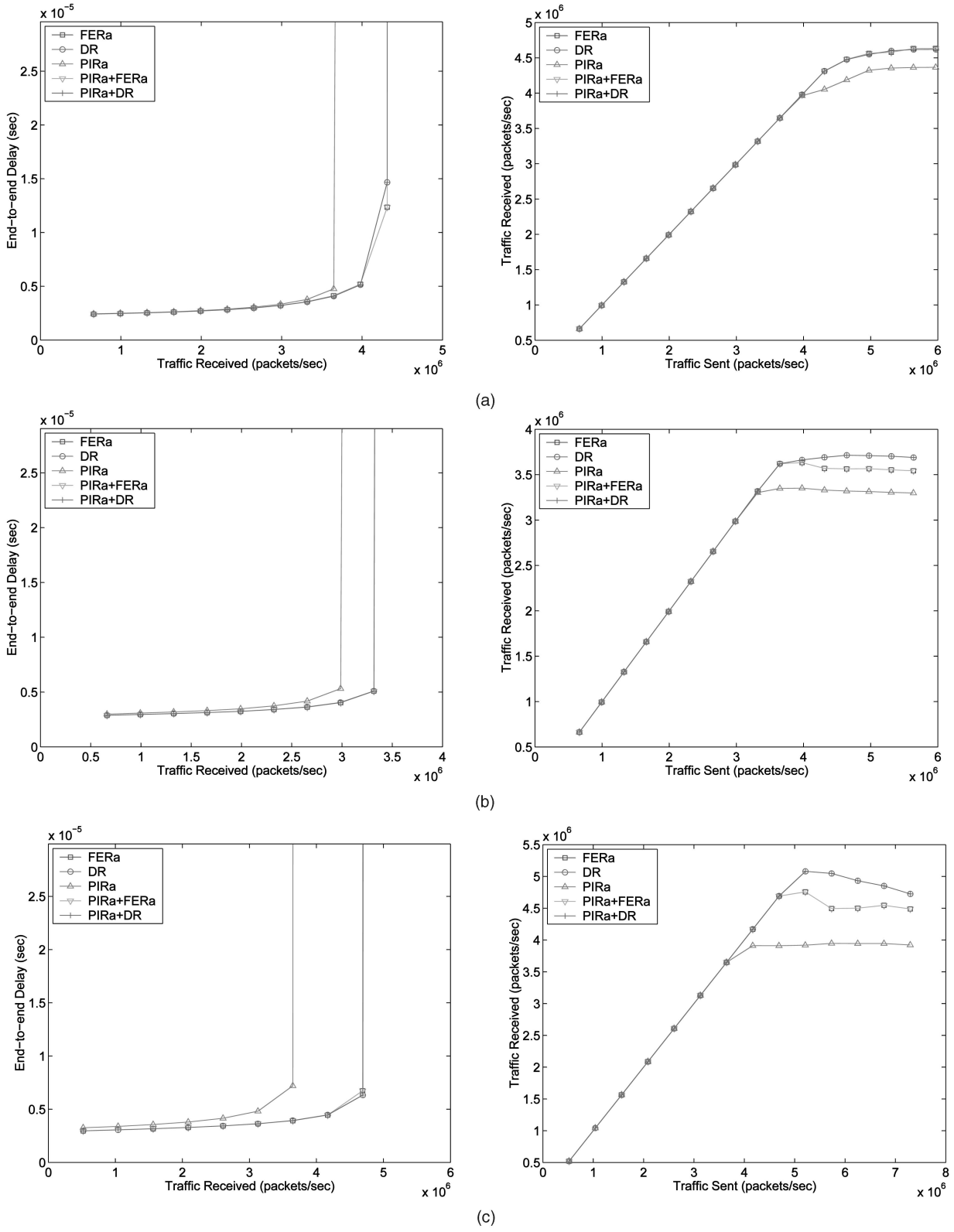


Fig. 14. Network latency and throughput for different topologies. (a) Eight switches and seven hosts (Fig. 1), (b) 16 switches and 14 hosts, and (c) 24 switches and 22 hosts.

## 5 Conclusions

The InfiniBand architecture provides a management mechanism that allows the subnet to autonomously assimilate the occurrence of a topology change. The main bottleneck of this mechanism is the time the subnet manager requires to compute a deadlock-free set of routes. In this paper, we have presented and evaluated a way to quickly compute a valid set of subnet routes. These routes take advantage of

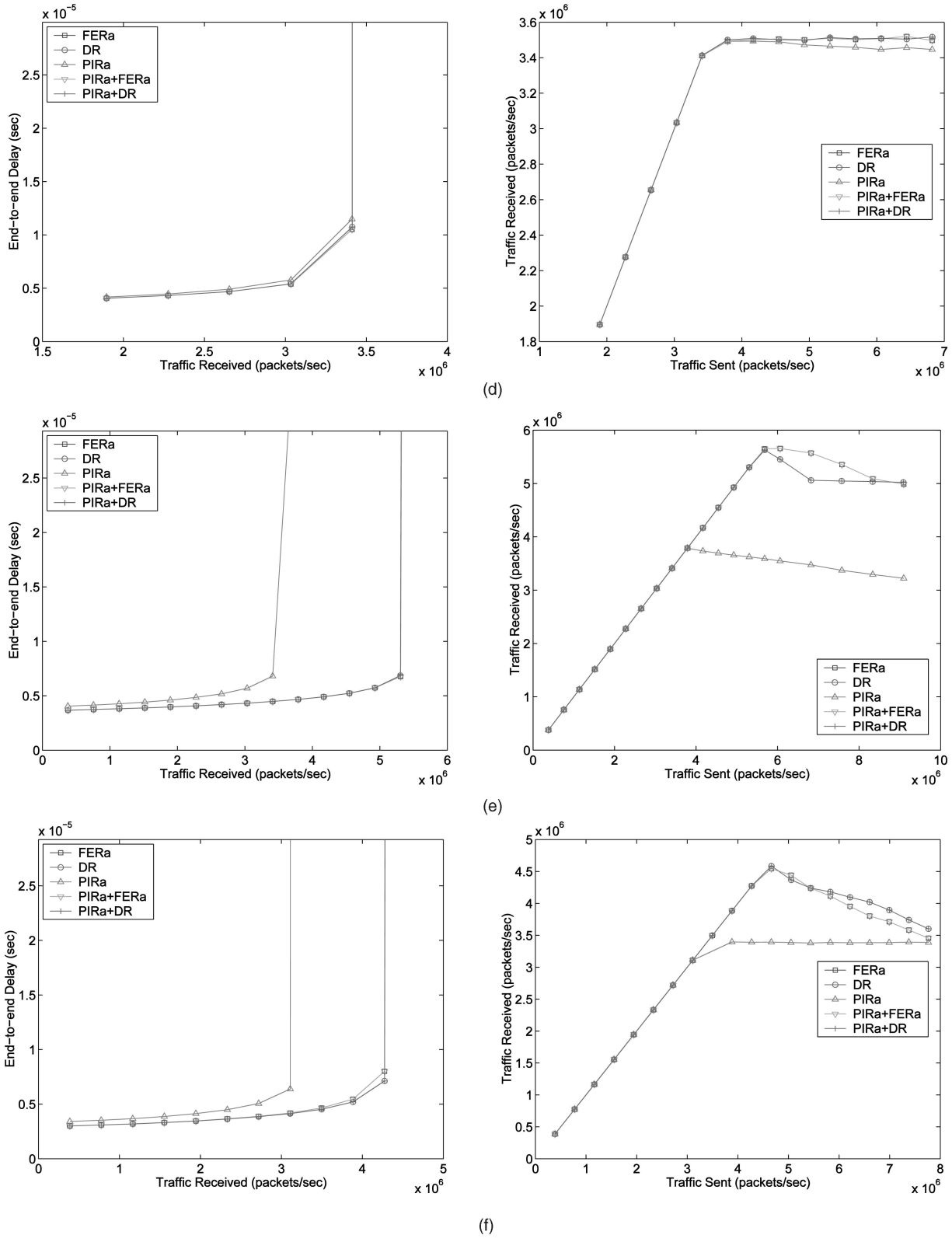


Fig. 14 (continued). (d) 32 switches and 40 hosts, (e) 48 switches and 64 hosts, and (f) 64 switches and 82 hosts.

the existence of a switch default port. The main advantage of the proposed strategy is that forwarding table entries can be distributed to subnet switches much earlier. The consequence is that the negative effects that the change assimilation process produces over application traffic are reduced. Once the change has been assimilated, network performance can be easily restored by computing and distributing a new set of more efficient paths.

## ACKNOWLEDGMENTS

This work was partly supported by the following Spanish projects: TIC2003-08154-C06 (Ministerio de Ciencia y Tecnología) and PBC05-007-1 (Junta de Comunidades de Castilla-La Mancha).

## REFERENCES

[1] A. Bermúdez, R. Casado, F.J. Quiles, T.M. Pinkston, and J. Duato, "Modeling InfiniBand with OPNET," *Proc. Second Ann. Workshop Novel Uses of System Area Networks,* Feb. 2003.
[2] A. Bermúdez, R. Casado, F.J. Quiles, T.M. Pinkston, and J. Duato, "Evaluation of a Subnet Management Mechanism for Infiniband Networks," *Proc. 2003 IEEE Int'l Conf. Parallel Processing,* Oct. 2003.
[3] A. Bermúdez, R. Casado, F.J. Quiles, T.M. Pinkston, and J. Duato, "On the Infiniband Subnet Discovery Process," *Proc. 2003 IEEE Int'l Conf. Cluster Computing,* Dec. 2003.
[4] A. Bermúdez, R. Casado, F.J. Quiles, and J. Duato, "Use of Provisional Routes to Speed-Up Change Assimilation in InfiniBand Networks," *Proc. 2004 IEEE Int'l Workshop Comm. Architecture for Clusters (CAC '04),* Apr. 2004.
[5] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kuawik, C.L. Seitz, J.N. Seizovic, and W. Su, "Myrinet: A Gigabit per Second LAN," *IEEE Micro,* vol. 15, no. 1, pp. 29-36, Feb. 1995.
[6] E.J. Kim, K.H. Yum, C.R. Das, M. Yousif, and J. Duato, "Performance Enhancement Techniques for Infiniband Architecture," *Proc. Ninth Int'l Symp. High-Performance Computer Architecture (HPCA-9),* 2003.
[7] P. López, J. Flich, and J. Duato, "Deadlock-Free Routing in InfiniBand through Destination Renaming," *Proc. 2001 Int'l Conf. Parallel Processing,* Sept. 2001.
[8] InfiniBand Architecture Specification (1.1), InfiniBand Trade Assoc., http://www.infinibandta.com/, Nov. 2002.
[9] OPNET Technologies, Inc., http://www.opnet.com/, 2006.
[10] T.L. Rodeheffer and M.D. Schroeder, "Automatic Reconfiguration in Autonet," SRC Research Report 77, in *Proc. ACM Symp. Operating Systems Principles,* Oct. 1991.
[11] J.C. Sancho, A. Robles, J. Duato, "A New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks," *Proc. Fourth Workshop Comm., Architecture, and Applications for Network-Based Parallel Computing,* Jan. 2000.
[12] J.C. Sancho, A. Robles, and J. Duato, "Effective Strategy to Compute Forwarding Tables for Infiniband Networks," *Proc. Int'l Conf. Parallel Processing,* Sept. 2001.
[13] J.C. Sancho, A. Robles, J. Flich, P. López, and J. Duato, "Effective Methodology for Deadlock-Free Minimal Routing in InfiniBand Networks," *Proc. Int'l Conf. Parallel Processing,* 2002.
[14] T. Shanley InfiniBand Network Architecture, Mindshare, Inc., Oct. 2002.
[15] M.D. Schroeder et al., "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE J. Selected Areas in Comm.,* vol. 9, no. 8, Oct. 1991.

**Aurelio Bermúdez** received the MSC degree in computer science from the University of Murcia in 1997 and the PhD degree from the University of Castilla-La Mancha in 2004. In 2000, he joined the Department of Computer Science at the University of Castilla-La Mancha. He is currently an assistant professor of computer architecture and technology. His research interests include routing, configuration, and fault tolerance in high-performance networks, and network simulation. He is a member of the IEEE.

**Rafael Casado** received the BSC degree in computer science from the University of Castilla-La Mancha in 1993, the MSC degree in computer science from the University of Murcia in 1995, and the PhD degree in computer science from the University of Castilla-La Mancha in 2001. In 1998, he joined the Department of Computer Science at the University of Castilla-La Mancha and, currently, he is associate professor in this department. His research interests include routing and reconfiguration algorithms for high-speed networks and multicomputers. Dr. Casado has served as a member of the program committee and a reviewer in several conferences and journals, including some of the most prestigious of the area. He is a member of the IEEE.

**Francisco J. Quiles** received the degree in physics (electronics and computer science) and the PhD degree from the University of Valencia, Spain, in 1986 and 1993, respectively. In 1986, he joined the Department of Computer Science at the University of Castilla-La Mancha, where he is currently a full professor of computer architecture and technology and vice-director of research at the University of Castilla-La Mancha. He has developed several courses on computer organization and computer architecture. His research interests include high-performance networks, parallel algorithms for video compression, and video transmission. He has published more than 100 papers in international journals and conferences on performance evaluation of parallel computer and communications systems and on video compression. He is a member of the IEEE and the IEEE Computer Society.

**José Duato** received the MS and PhD degrees in electrical engineering from the Technical University of Valencia, Spain, in 1981 and 1985, respectively. Currently, Dr. Duato is a professor in the Department of Computer Engineering (DISCA) at the same university. He was also an adjunct professor in the Department of Computer and Information Science, The Ohio State University. His current research interests include interconnection networks and multiprocessor architectures. Professor Duato has published more than 300 refereed papers. He proposed a powerful theory of deadlock-free adaptive routing for wormhole networks. Versions of this theory have been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E supercomputer, the internal router of the Alpha 21364 microprocessor, the IBM BlueGene/L supercomputer, and the Cray Black Widow supercomputer. He is the first author of the book *Interconnection Networks: An Engineering Approach*. Dr. Duato served as a member of the editorial boards of *IEEE Transactions on Parallel and Distributed Systems* and *IEEE Transactions on Computers*. He has been the general cochair for the 2001 International Conference on Parallel Processing, the program committee chair for the 10th International Symposium on High Performance Computer Architecture (HPCA-10), and the program cochair for the 2005 International Conference on Parallel Processing. Also, he has served as cochair, member of the steering committee, vice-chair, or member of the program committee in more than 50 conferences, including the most prestigious conferences in his area (HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, Europar, and HiPC). He is a member of the IEEE and the IEEE Computer Society.