DIMACS Technical Report 99-19
April 1999

Deadlock-Free Routing in Irregular Networks Using Prefix Routing

by

Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

Li Sheng
Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, Bellcore and NEC Research Institute.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

In this paper, we propose a deadlock-free routing in irregular networks using prefix routing. Prefix routing is a special type of routing with a compact routing table associated with each node (processor). Basically, each outgoing channel of a node is assigned a special label and an outgoing channel is selected if its label is a prefix of the label of the destination node. Node and channel labeling in an irregular network is done through constructing a spanning tree. The routing process follows a two-phase process of going up and then down along the spanning tree, with a possible cross channel (shortcut) between two branches of the tree between two phases. We show that the proposed routing scheme is deadlock- and livelock- free. Possible extensions are also discussed.

Index terms: Deadlock-freedom, irregular networks, livelock-freedom, routing, spanning trees.

1 Introduction

Switch-based networks are becoming more and more popular to meet the ever increasing demand for high performance. Many switching hubs have been used in switched LANs, such as Fast Ethernet, FDDI, Myrinet, and ATM. In general, switch-based networks provide virtual point-to-point communication, and hence, offer better throughput and lower latency for many applications.

Networks of workstations (NOWs) with underling switch-based networks have been considered as a cost-effective alternative to massively parallel computers. Workstations and switches can be interconnected to form various topologies, mostly irregular ones. Routing is the process of transmitting data from the *source* node to the *destination* node in a given system. Many deadlock-free routing algorithms have been proposed for regular topologies such as meshes, tori, and hypercubes by taking the advantage of convenient addressing schemes offered in most regular topologies. A *deadlock* occurs when several routing processes are in a circular waiting state and cannot advance toward their destination because the channels required by them are not available. A *livelock* occurs when a routing process travels around its destination node, never reachs it. Irregular topologies pose some new challenges for the routing process:

- It is difficult, if not impossible, to derive a routing scheme without using a complete routing table.
- Irregular routing paths (because of irregular topologies) pose an additional dimension of difficulty to ensure deadlock- and livelock-freedom.

Most deadlock-free routing algorithms for irregular networks are based on a spanning tree [5], [12], [13] or a Eulerian path [8]. By restricting the routing path along branches in the spanning tree (with limited and controlled *jumps*, also called *shortcuts*, between tree branches), deadlock-free routing is derived. In a path-based approach, a Eulerian path is first constructed to ensure a feasible path between any two nodes. Shortcuts are allowed to generate shorter paths. In general, the tree-based approach is more favorable than the path-based approach in unicasting (which includes one source and one destination) for generating shorter routing paths on average. However, to ensure deadlock-freedom in multicasting (which includes one source and multiple destinations), the path-based approach is more involved [5].

In [9], an up*/down* unicast routing algorithm is proposed aiming to better utilize all the available channels in the network. First of all, an arbitrary node is selected as a special node and then the network is partitioned into two subnetworks: up subnetwork and down subnetwork. The up subnetwork consists of unidirectional channels directed towards the special node and the down network consists of unidirectional channels directed away from the special node. In case of a tie, tie breaker is made by comparing the ids of two end nodes connected by the channel. A routing process always selects a sequence of up channels (if any) followed by a sequence of down channels (if any). Based on the definition of the up (down) subnetwork, no cycle exists among up (down) channels. A cycle involves up and down channels is impossible, since a transition from a down channel to an up channel is forbidden. To determine the status of each channel (up/down), a minimum spanning tree is constructed from the special node (root node) that connects each node through a shortest path. The routing in Autonet [9] was built based on this up*/down* unicast routing algorithm. However, a routing table is still maintained to ensure only the legal routes with the *minimum hop counts* (shortest) are allowed. The up*/down* unicast routing algorithm can be also applied in regular networks such as 2-D meshes, however, its performance cannot match the traditional XY routing [10].

Both path-based and tree-based approaches do not address adequately the addressing scheme, making them difficult to be implemented in an actual system. Without a simple addressing scheme in a given network, source and destination nodes in a unicast do not know their relative locations. The traditional table lookup approach works but it cancels out the elegant feature of these approaches. In [5] and [10], each header of the routing message is associated with a bit-string of length n, where n is the number of nodes in the irregular network. Similarly, every node has an n-bit string (called *reachability string*) associated with every one of its outgoing channels that lead to channels in the down direction, where n is the number of nodes in the network. These reachability strings can be constructed during the formation of the spanning tree.

Our approach is based on *compact routing* that uses a reduced size of a routing table [4]. Two commonly used compact routing schemes are *interval routing* [1], [3] and *prefix routing* [2]. Both schemes are based on assigning special labels to each unidirectional channel. At each routing step, a particular neighbor is selected (as the next forwarding node), if the label in the corresponding channel meets a certain condition. In interval routing, each channel is associated with an interval of integers. A channel is selected if the destination address (an integer) is within the interval. In prefix routing, each channel is associated with a label of a binary string and each node is also labeled with a binary string. A channel is selected, and hence, the corresponding neighbor is selected, if the channel label is a prefix of a destination node label .

In this paper, we extend a prefix routing algorithm proposed in [2] and prove it to be both deadlock- and livelock-free. Like other tree-based approaches, routes are not necessarily the shortest. Since currently most LAN switches are built using cut-through switching, which can forward partially received data as soon as the packet header is received, low-latency delivery is obtainable and it is relatively insensitive to the hop count.

This paper is organized as follows: Section 2 proposes a prefix routing algorithm, followed by an example, and the proofs of deadlock- and livelock-freedom. Section 3 discusses possible extensions. Finally, Section 4 concludes this paper. In the subsequent discussion, we refer "routing" to "unicasting routing" without causing confusion.

2 Prefix Routing

Suppose $G = (V, E)$ is a graph representing an irregular network, where V is the vertex set and E is the edge set. $v \in V$ represents a node in the network and uv represents a link (also called a *channel*) between nodes u and v. Note that v can be either a switch or a workstation in NOWs. Two switches or one switch and one workstation can be connected, but not two workstations. To simplify our discussion, we do not distinguish a switch from a workstation and will simply call each of them a node. Each link uv has two unidirectional channels: (u, v) and (u, v). Prefix routing is based on a labeling scheme that assigns a label to each node and channel. We use $L(v)$ and $L(u, v)$ as labels for node v and channel (u, v), respectively. In the following discussion, a "link" and a "channel" represent an undirected edge and a directed edge, respectively.

Our approach consists of two phases: preprocessing and routing.

1. (Preprocessing): Build a spanning tree and assign labels to nodes and channels.
   - Build a spanning tree of a given graph rooted at a selected node.
   - Assign labels to nodes and channels of the spanning tree during its formation.
   - Complete labels to all the remaining channels in the graph.
2. (Routing): Construct a distributed routing algorithm.
   - Construct a distributed routing scheme based on the following: Suppose d is the destination and v is the current node, node u will be the forwarding node if $L(v, u)$ is a prefix of $L(d)$.

2.1 Building a spanning tree

The spanning tree can be built using one of the traditional methods: Initially, all nodes are unmarked. The process starts from a selected node, r, called a root. A signal is sent from root r to all its adjacent nodes $adj(r)$. Once node v receives a signal from node w and node v is unmarked, the parent-child relation is established between w and v. Node v continues the same process by broadcasting its signal to its adjacent nodes $adj(v)$. A marked node will ignore any signal received. This process normally generates a *minimum spanning tree*, i.e., each node is reached from the root (through a branch in the spanning tree) through a shortest path. This occurs when latency of transmitting a signal between two adjacent nodes is uniform and the underlying switch has all-port capability; that is, it can simultaneously send (and receive) signals along different channels. However, our approach works for any spanning tree, not limited to the minimum one.

We use the following high-level message passing model [14] to describe the proposed algorithm. Messages are passed to a named receiver node through asynchronous static channels (links). That is, send and receive commands involved in a message exchange do not need to be executed at the same time. A buffer is usually used to hold the message sent from the sender. An output command is of the form: **send** message_list **to** destination, where the destination is a node id. An input command has the form: **receive** message_list **from** source, where the source is a node id. We also use Dijkstra guarded command $G \to C$, where G is a guard consisting of a list of Boolean expressions and C is a regular command. $*[G \to C]$ represents a repetitive statement and when the guard fails the repetitive statement terminates. An alternative statement is expressed in the form:

$$[\ G_1 \to C_1 \ \Box\ G_2 \to C_2 \ \Box ... \Box\ G_n \to C_n\ ]$$

The alternative statement selects the execution of exactly one of the constituent guarded commands. In the case where more than one command list can be selected, the choice is *nondeterministic*.

To construct a spanning tree, two different processes are used: one at root r and the other at non-root node v. Initially, $mark(v) = F$, for all $v \in V$.

At root r:
    $mark(r) := T$;
    $*$ [another node u in $adj(r) \to$
            [**send** *parent_sig* **to** node u;
              receive *child_sig* **from** node $u \to$ place u in $child(r)$]
      ]

At a non-root node v:
  $*$ [**receive** *parent_sig* **from** node w and $mark(v) = F \to$
            [$mark(v) := T$;
              $*$ [another node u in $adj(v) \to$
                        [**send** *parent_sig* **to** node u;
                          receive *child_sig* **from** node $u \to$ place u in $child(v)$]
                ]
            ]
    $\Box$ **receive** *parent_sig* **from** node w and $mark(v) = T \to$ no action
    ]

Note that in the above algorithm, each node will send back $parent\_sig$ to its parent node (since it is an adjacent node). This will not cause any problem, because the parent node has been marked true and will ignore the signal.

We assume that the process of constructing a spanning tree starts at one selected node. This approach can be extended to the case where each node can initiate its own process (and could be at the same time) and end with only one winner (the winner is the root node) [7]. More general ways of constructing a spanning tree can be found in [6].

Based on the definition of spanning trees, we can define three types of channels:

- *Up channel*: a channel in the spanning tree that directs towards the root.
- *Down channel*: a channel in the spanning tree that directs away from the root.
- *Cross channels*: all channels that are not in the spanning tree.

2.2 Assignment of labels to nodes and channels

The labeling assignment is extended from the one in [2]. Assignment of labels to nodes and channels of the spanning tree is done as follows: The label of the root is 1, i.e., $L(r) = 1$. If u is the kth child of v, then assign $L(u) = L(v)||k$, where $||$ represents a concatenation operation. If node v is the father of node u, then $L(v, u) = L(u)$ and $L(u, v) = e$, where e represents an empty string label. In the distributed formation of the labeling scheme, each node v decides its label and labels for channels (v, u), where $u \in adj(v)$. Without loss of generality, we assume the maximum number of children (of a node) is less than the base of the selected number system; otherwise, we can always insert a special character to indicate the beginning of symbol k. Because each message exchange may involve several messages, say k messages, we use the following format in each exchange: $(type\_of\_message, msg_1, ..., msg_k)$

Again, two different processes are used in the node labeling process: one for root r and the other one for non-root node v. Since the labeling process is done during the formation of the spanning tree, these two processes can be combined.

At root r:

```
mark(r) := T; k := 1; L(r) := 1
* [ another node u in adj(v) →
        [ send (parent_sig, L(r), k) to node u; k := k + 1;
          receive (child_sig, child_count) from node u →
                [ place u in child(r); L(r, u) = L(r)||child_count ]
        ]
  ]
```

At a non-root node v:

```
* [ receive  (parent_sig, parent_label, child_count) from  node w and mark(v) = F →
      [ send (child_sig, child_count) to node w;
        mark(v) := T; k := 1;
        L(v) = parent_label||child_count; L(v, w) := e;
        * [ another node u in adj(v) →
                [ send (parent_sig, L(v), k) to node u; k := k + 1;
                  receive (child_sig, child_count1) from node u →
                        [ place u in child(v); L(v, u) = L(v)||child_count1  ]
                ]
          ]
      ]
  □ receive  (parent_sig, parent_label, child_count) from  node w and mark(v) = T →
          no action
  ]
```

The labeling of channels that are outside the spanning tree is done based on the node labels of two end nodes: If there is no parent-child relation between v and u and $uv \in E$, then $L(v, u) = L(u)$ and $L(u, v) = L(v)$.

At any node v:

```
* [ L(v, u) is not assigned, where u ∈ adj(v) →
        [ send ask_neighbor_label to node u;
          receive (neighbor_label, label) from node u → L(v, u) := label;
        ]
```

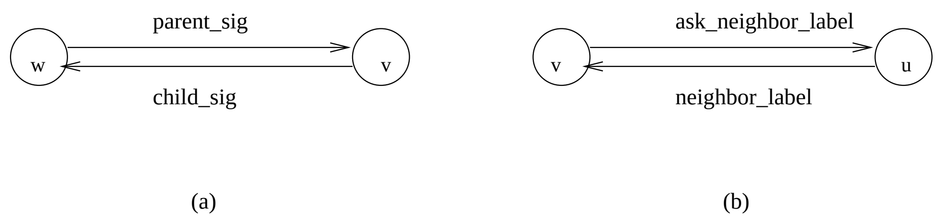

Figure 1: Message exchanges between (a) parent and child nodes, (b) connected nodes with no parent-child relation.

$\Box$ **receive** *ask_neighbor_label* **from** node $u \rightarrow$
  send (*neighbor_label*, $L(v)$) **to** node u
]

Figure 1 shows two different types of message exchange between adjacent nodes in the formation of the spanning tree and labeling scheme.

Note that an up channel has a label e, a down channel carries the label of the corresponding child node, and a cross channel has the label of the corresponding cross neighboring node (at a different branch of the spanning tree).

To estimate the length of each label, we assume that each switch has a bounded *maximum vertex degree*, Δ, where $\Delta = \max\{d_G(v) | v \in V\}$ and $d_G(v)$ is the *vertex degree* of v in G. The length of each label depends on the *level* of the corresponding node (the length of the unique path in the tree from the root to the node), and the number of siblings (i.e., the number of nodes that share a common parent node with the current node). The level of each node is bounded by the *depth* (the length of the longest path in the tree from the root to a leave node) of the minimum spanning tree. The depth of a minimum spanning tree is bounded by the diameter, $diam(G)$, of the graph G representing the irregular network. Therefore, the length of each label is bounded by $diam(G) \log \Delta$. The label of each down channel in the spanning tree can be further reduced by keeping only the difference between the labels of two end nodes. Note that in this case the label of the parent node is a prefix of the label of its child nodes.

2.3 Distributed routing algorithm

The distributed routing algorithm based on the proposed labeling scheme is the following:

- At an intermediate node v (including source node s), neighbor u is selected as the forwarding node if $L(v, u)$ is a prefix of $L(d)$, where d is the destination.
- If such a neighbor does not exist, select a neighbor w such that $L(v, w) = e$.

Basically, this algorithm is based on the label associated with each outgoing channel. A channel is selected if the corresponding channel label is a prefix of the label of the destination (that is the reason the proposed routing is called prefix routing). If there is no outgoing channel that has a label matching the one of the destination, an up channel (with label e) is selected.

In general, a routing process proceeds by visiting a sequence of up channels (if any), followed by at most one cross channel, and ended with a sequence of down channels (if any).

2.4 Example

In the example of Figure 2, an irregular network with six nodes is shown, together with a spanning tree (with its links represented as solid lines). Applying the proposed prefix routing algorithm, the path for $(s, d) = (11, 121)$ is $11 \rightarrow 12 \rightarrow 121$ and the path for $(s, d) = (112, 121)$ is $112 \rightarrow 12 \rightarrow 121$. Note that we use here s to represent both the source node and its label $L(s)$.

We assume that each node can relate the label of a destination to its id (node id). Each node keeps a label table as shown in Table 1 for the irregular network in Figure 2. The table can be derived during the formation of the spanning tree: Each node forwards its node id and label pair up along the tree until reaching root node r. Once r receives all node id and label pairs, a label table is constructed. Finally, r broadcasts the label table down the spanning tree.

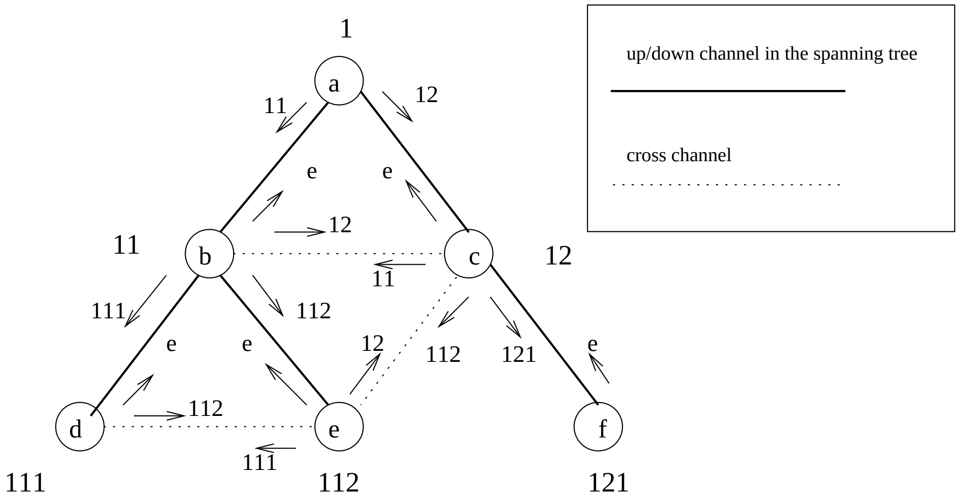

Figure 2: A sample irregular network.

2.5 Proofs of deadlock- and livelock-freedom

We now prove that although the proposed routing algorithm may not generate a shortest path for a given source-destination pair, it is deadlock- and livelock-free.

Theorem 1 *The proposed routing algorithm is deadlock-free.*

Proof. We prove that any routing process proceeds by visiting a sequence of up channels (if any), followed by at most one cross channel, and ended with a sequence of down channels (if any). Since any sequence of up (down) channels is acyclic, the routing process is deadlock-free.

Based on the address labeling scheme, a node label $L(s)$ is a prefix of another node label $L(d)$ if and only if node s is an ancestor of node d in the spanning tree. We consider the following three cases:

Case 1 : source s is an ancestor of destination d.

Based on the channel labeling scheme, destination d can be reached through a unique sequence of down channels. We now show that it is the only possible routing path. Clearly, no up channel will be used, since it is used only when the current node label is not a prefix of the destination node label. Also, no cross channel will be used. For otherwise, suppose at an intermediate node w, a cross channel is used to reach node v at a different branch, based on the property derived from the labeling scheme, both w and v are ancestors of destination d, which brings a contradiction. In summary, the routing process follows a unique sequence of down channels to reach the destination in this case.

Case 2 : source s is a decedent of destination d.

The label of s is not a prefix of d. The routing process follows a sequence of up channels to reach the destination. However, if the spanning tree is nonminimal, i.e., the path (in the spanning tree) from the root to an intermediate node is not necessarily the shortest one in G. The following situation can also occur: A sequence of up channels are used to reach an intermediate node u (including source node s). If there is a cross neighbor v (of u) that is an ancestor of d, then the corresponding cross channel is used to reach node v and the remaining routing process resembles Case 1 from node v to node d which consists of a sequence of down channels. Note that if the spanning tree is a minimal one, i.e., each node is reached from the root (in the spanning tree) through a shortest path, the above case will never occur, since node s can be reached from the root through a shorter path $(... \to v \to u \to s)$ than the current one $(... \to v \to ... \to d \to ... \to u \to s)$. As a summary for the nonminimal spanning tree case: the routing process follows a sequence of up channels to reach the destination or it follows a sequence of up channels (if any) until it reachs an intermediate node u that has a cross neighbor v which is an ancestor of destination d, and then, Case 1 applys to the remaining routing process to reach d from v. Therefore, prefix routing is deadlock-free even if the underlying spanning tree is not a minimum one.

Case 3 : source s and destination d do not have the ancestor/decedent relation.

This case resembles Case 2, where a sequence of up channels are used, unless there is a cross neighbor that is an ancestor of the destination. The general routing process follows a sequence of up channels until reaching either the lowest common ancestor of s and d or the first intermediate node (including source node s) that has a cross neighbor which is an ancestor of the destination. In the later case, the corresponding cross channel is used to reach that neighbor. Finally for both cases, the routing process completes by following a sequence of down

Id	a	b	c	d	e	f
Label	1	11	12	111	112	121

Table 1: A label table associated with each node.

channels to reach the destination.

Therefore, the proposed routing algorithm is deadlock-free. □

Theorem 2 *The proposed algorithm is livelock-free.*

Proof. By the definition of the proposed routing algorithm, the routing process starts with a sequence of up channels (if any). Since up channels do not form a cycle, this sequence is finite. Once it uses a cross channel, the routing processing ends with a sequence of down channels (if any). Again, since down channels do not form a cycle, this sequence is finite. Therefore, any routing process takes a finite number of steps to reach the destination. □

3 Extensions

In this section, we consider possible extensions to the proposed prefix routing. Two directions of extensions are considered: (1) improve channel utilization and (2) reduce the length of a routing path.

3.1 Routing with multiple cross channels

In the prefix routing scheme, shortcuts are only possible between *adjacent* tree branches, i.e, two branches that have a direct link connection. To support shortcuts between branches that are not adjacent, we have to distribute channel labels to nodes in the neighborhood.

For example, one possible extension is for node u to pass label $L(u, v)$ of a cross link uv to node w (via node v) that has a cross link wv with common end node v. In this case, each cross link may have up to Δ labels, where Δ is maximum vertex degree of the graph. The same routing scheme still applies and can be proved to be deadlock- and livelock-free. The difference is that in the extended routing scheme, up to two cross channels may appear in a routing path. Using this approach, the average length of routing path can be reduced due to the possible use of more shortcuts, but the memory needed to store all the labels for cross links are increased.

3.2 Multiple spanning trees

In an n-node irregular network, the number of up and down channels is $2(n-1)$. The remaining channels are cross channels. Based on the definition of the prefix routing algorithm, each routing process uses at most one cross channel. Obviously, cross channels cannot be fully used. In an irregular topology that corresponds to a dense graph with $O(n^2)$ links, the ratio between the number of cross channels and the number of up and down channels is n and a large percent of links are not utilized. However, most switched LANs have a constant number of incoming and outgoing channels, 2Δ, where Δ is a constant c; therefore, such a ratio is bounded by constant $c-1$.

In order to increase channel utilization, multiple spanning trees can be used. At least c multiple spanning trees should be used to fully utilize all the links; that is, to ensure each link (with two channels) appears in at least one spanning tree. To avoid introducing too many *virtual channels* (derived from a physical channel by multiplexing the physical channel into many logical (virtual) channels), edge-disjoint spanning trees should be constructed if possible. Basically, k virtual channels will be employed if the corresponding channel is used in k different spanning trees. However, the problem of minimizing the number of virtual channels while ensuring each link appears in at least one spanning tree is conjectured to be NP-hard in a given irregular network. Various trade-offs can be used.

One simple solution is to introduce two edge-disjoint spanning trees and each node (and link) has two labels, one for each tree. Routing can follow either one of the spanning trees. When it is impossible to construct two edge-disjoint spanning trees, two virtual channels are introduced. The cost-effectiveness of this extension remains to be verified through simulation. We need to compare this approach with the one by Silla and Duato [11], where two virtual channels are used, one channel is used in a deadlock-free routing and the other can be used without any restriction.

3.3 Minimum depth spanning trees

In the proposed routing algorithm, both the length of each label and the routing path depend on the depth of the minimum spanning tree. This depth is between $diam(G)/2$ and $diam(G)$, where $diam(G)$ stands for the diameter of graph G. We can use one of the existing algorithms to construct a *minimum depth spanning tree*. Note that this is based on the selection of the root node. Once a root is selected, we build a minimum spanning tree. Among all the minimum spanning trees (with different roots), we select one with a minimum depth. In this case, the average length of a label and a routing path can be reduced.

4 Conclusions

In this paper, we have extended a prefix-based routing scheme in irregular networks and shown that it is deadlock- and livelock-free. Unlike traditional path-base and tree-based routing, prefix routing is based on a simple labeling scheme and labels to nodes and channels are assigned during the formation of a spanning tree. Both prefix routing and the existing up*/down* routing are based on string matching during the routing process. Prefix routing can be considered as a complement to up*/down* routing. Our future work will focus on comparing the proposed scheme with existing ones, such as the up*/ down* routing algorithm, through simulation.

References

[1] E. M. Bakker and J. van Leeuwen. Linear interval routing. *Algorithms Review*. 1991, 45-61.

[2] E. M. Bakker, J. van Leeuwen, and R. B. Tan. Prefix routing schemes in dynamic networks. *Computer Networks and ISDN Systems.* 26, 1993, 403-421.

[3] P. Fraigniaud and C. Cavoille. Interval routing schemes. *Proc. of the 12th Annual Symposium on Theoretical Aspects of Computer Science.* LNCS 900, Springer-Verlag, 1995, 279-290.

[4] G. N. Frederickson and R. Janardan. Optimal message routing without complete routing tables. *Proc. of the 5th ACM Symp. on Principles of Distributed Computing.* 1986, 88-97.

[5] R. Liberskind-Hadas, D. Mazzoni, and R. Rajagopalan. Tree-based multicasting in wormhole-routed irregular topologoies. *First Merged IPPS/SPDP.* 1998, 244-249.

[6] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann. 1996.

[7] R. Perlman. An algorithm for distributed computation of a spanning tree in an extended lan. *Proc. of the 9th Data Communication Symp.* Sept. 1985, 44-53.

[8] W. Qiao and L. M. Ni. Adaptive routing in irregular networks using cut-through switches. *Proc. of the 1996 International Conference on Parallel Processing.* Vol. 1, Aug. 1996, 46-60.

[9] M. Shroeder, A. D. Birrel, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal of Selected Areas in Communications.* 9, (10), Oct. 1991, 1318-1335.

[10] F. Silla and J. Duato. Is it worth the flexibility provided by irregular topologies in networks of workstations? *Proc. of Workshop on Communications, Architecture, and Applications for Network-based Parallel Computing (CANPC'99).* Jan. 1999.

[11] F. Silla and J. Duato. On the use of virtual channels in networks of workstations with irregular topology. *1997 Parallel Computer Routing and Comm. Workshop.* 1997.

[12] R. Sivaram, R. Kesavan, D. K. Panda, and C. B. Stunkel. Where to provide support for efficient multicasting in irregular networks: network interface or

switch? *Proc. of the 1998 Internatioanl Conference on Parallel Processing.* 1998, 452-459.

[13] R. Sivaram, D. K. Panda, and C. B. Stunkel. Multicasting in irregular networks with cut-through switches using tree-based multidestination worms. *Proc. of the 2nd Parallel Computing, Routing, and Communication Workshop.* June 1997.

[14] J. Wu. *Distributed System Design.* The CRC Press. 1999.