Deadlock-Free Routing in Irregular Networks Using Prefix Routing *

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

Abstract

We propose a deadlock-free routing scheme 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. We also compare prefix routing with the existing up*/down* routing which has been widely used in irregular networks. Possible extensions are also discussed.

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

*This work was supported in part by NSF grants CCR 9900646 and ANI 0093936.

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 underlying 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 [5, 8] by taking 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, but never reachs it. Unlike regular topologies, irregular topologies pose some new challenges to design a deadlock- and livelock-free routing process: (1) It is difficult, if not impossible, to derive an efficient routing scheme without using a complete routing table. (2) 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 [6], [13] or a Eulerian path [10]. 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 tree-based approach is more involved [6].

In [11], 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 while the down network consists of unidirectional channels directed away from the special node. In case of a tie, a 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 that involves up and down channels is impossible, because 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 [11] 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 also be applied in regular networks such as 2-D meshes, however, its performance cannot match the traditional XY routing [12].

Neither the path-based approach nor the tree-based approach supports an efficient addressing scheme. The traditional table lookup approach works but it cancels out the elegant feature of these approaches. In [6] and [12], 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 routing table of reduced size [4]. Two commonly used compact routing schemes are *interval routing* [3] and *prefix routing* [1]. 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 the label of the destination node.

In this paper, we extend a prefix routing algorithm proposed in [1] 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 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 compares prefix routing with up*/down* routing. Section 4 discusses possible extensions. Finally, Section 5

concludes this paper. In the subsequent discussion, we refer "routing" as "unicast 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 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 refer to 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): 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 label assignments to all the remaining channels in the graph.

(Routing): Construct a distributed routing algorithm.

- 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 $L(d)$.

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*; that is, each node is reached from the root 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; it is not limited to the minimum one.

We use the following high-level message passing model [16] to describe the proposed algorithm. Messages are passed to a named receiver node through asynchronous static channels. An output command is of the form: **send** message_list **to** destination. An input command has the form: **receive** message_list **from** source. We also use Dijkstra guarded command $G \rightarrow C$, where G is a guard consisting of a list of Boolean expressions and C is a regular command. $*[G \rightarrow C]$ represents a repetitive statement and when the guard fails the repetitive statement terminates. An alternative statement is expressed in the form: $[\ G_1 \rightarrow C_1\ \Box\ G_2 \rightarrow C_2\ \Box ... \Box\ G_n \rightarrow C_n\ ]$. 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.

Two different processes are used to construct a spanning tree: 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 $adj(r) \rightarrow$
  [**send** *parent_sig* **to** node u;
    receive *child_sig* **from** node $u \rightarrow$ place u in $child(r)$]
]

At a non-root node v:
$*$ [**receive** *parent_sig* **from** node w and $mark(v) = F \rightarrow$
  [$mark(v) := T$;
    $*$ [another node u in $adj(v) \rightarrow$
      [**send** *parent_sig* **to** node u;
        receive *child_sig* **from** node $u \rightarrow$ place u in $child(v)$]
    ]
  ]
$\Box$ **receive** *parent_sig* **from** node w and $mark(v) = T \rightarrow$
  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 multiple nodes can initiate at the same time) and end with only one winner (the winner is the root node) [9]. More general ways of constructing a spanning tree can be found in [7].

Based on the definition of a spanning tree, 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 channel*: a channel that is not in the spanning tree.

If a cross channel connects two nodes in the same branch of the tree, it is called *up-cross* (*down-cross*) if it directs towards (away from) the root. Other cross channels (that connect two nodes in different branches) are simply called cross channels. Note that up-cross and down-cross appear only when the spanning tree is non-minimum.

Assignment of labels to nodes and channels. The labeling assignment is extended from the one in [1]. 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 that 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 m messages, we use the following format in each exchange: $(type\_of\_message, msg_1, ..., msg_m)$.

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 $adj(v)$ $\rightarrow$

[**send** $(parent\_sig, L(r), k)$ **to** node u; $k := k + 1$;
receive $(child\_sig, child\_count)$ **from** node $u \rightarrow$
[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 $mark(v) = F \rightarrow$
[**send** $(child\_sig,\ child\_count)$ **to** node w;
$mark(v) := T$; $k := 1$;
$L(v) = p\_label \| child\_count$; $L(v, w) := e$;
$*$ [another node u in $adj(v) \rightarrow$
[**send** $(parent\_sig, L(v), k)$ **to** node u; $k := k + 1$;
receive $(child\_sig, child\_count1)$ **from** node $u \rightarrow$
[place u in $child(v)$; $L(v, u) = L(v) \| child\_count1$]
]
]
]
$\square$ **receive** $(parent\_sig,\ parent\_label,\ child\_count)$ **from** node w and $mark(v) = T \rightarrow$
no action
]

The labeling of channels that are outside the spanning tree is based on labels of two end nodes: If there is no parent-child relationship 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 $u \in adj(v) \rightarrow$
[**send** $ask\_neighbor\_label$ **to** node u;
receive $(neighbor\_label,\ label)$ **from** node $u \rightarrow$
$L(v, u) := label$;
]
$\square$ **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

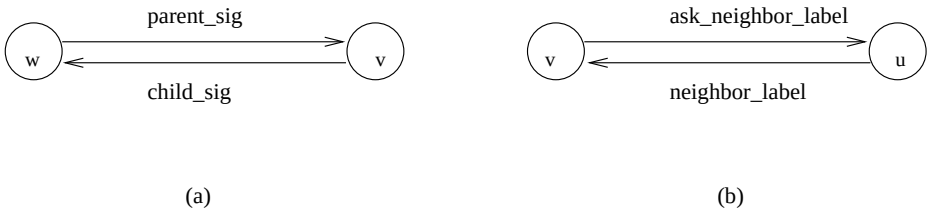

Figure 1: Message exchanges between (a) parent and child nodes, (b) neighbors with no parent-child relation.

formation of the spanning tree and the 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.

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 leaf 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.

Distributed routing algorithm. The routing algorithm in terms of the proposed labeling scheme is as follows:

- 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. 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 ending with a sequence of down channels (if any).

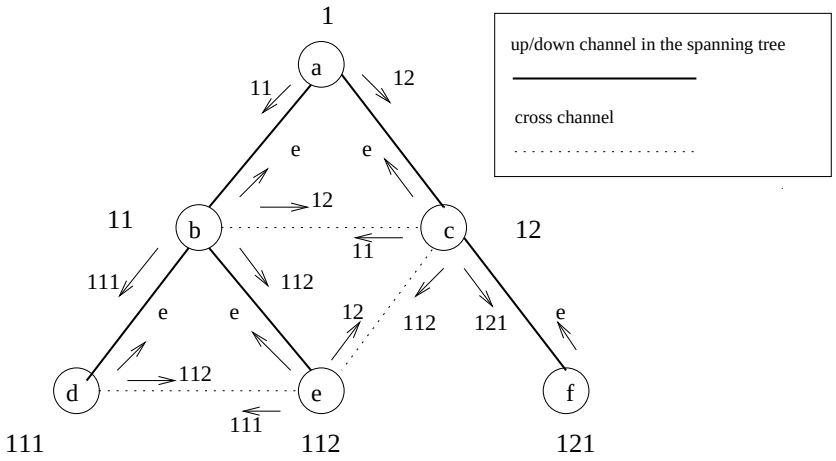

Figure 2: A sample irregular network.

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

Table 1: A label table associated with each node of Figure 2.

In Figure 2, an irregular network with six nodes is shown, along with a spanning tree (with its links represented as solid lines). Applying the proposed prefix routing algorithm, we can derive the path $11 \rightarrow 12 \rightarrow 121$ for $(s, d) = (11, 121)$ and the path $112 \rightarrow 12 \rightarrow 121$ for $(s, d) = (112, 121)$. Note that here we use 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 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.

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 or up-cross channel (if any), and ending with a sequence of down or down-cross channels (if any). Since any sequence of up channels is acyclic and any sequence of down or down-cross 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 there is only one possible routing path if the spanning tree is minimum. Clearly, no up channel can be used, since an up channel is used only when the current node label is not a prefix of the label of the destination node. Also, no cross channel will be used; 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 is a contradiction. If the spanning tree is non-minimum, down-cross channels may be used. A down-cross channel (u, v) is used if v has a longer prefix of destination d than the label of u's child has in the spanning tree. Note that if the spanning tree is minimum, that is, each node is reached from the root (of the spanning tree) through a shortest path, the above case will never occur, because in this case node d can be reached from the root through a shorter path ($... \rightarrow v \rightarrow u... \rightarrow d$) than the current one ($... \rightarrow v... \rightarrow u... \rightarrow$).

(Case 2): Source s is a descendant 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 non-minimum, the following situation can also occur: A sequence of up channels is used to reach an intermediate node u (including source node s). If there is a neighbor v (of u) that is an ancestor of d, then the corresponding up-cross channel is used to reach node v. The remaining routing process resembles Case 1 from node v to node d which consists of a sequence of down or down-cross channels. Note that if the spanning tree is minimum, that is, each node is reached from the root of the spanning tree through a shortest path, the above case will never occur, because node s can be reached from the root through a shorter path ($... \rightarrow v \rightarrow u... \rightarrow s$) than the current one ($... \rightarrow v \rightarrow ... \rightarrow d \rightarrow ... \rightarrow u... \rightarrow s$). As a summary for the non-minimum 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 neighbor v which is an ancestor of destination d, and then, Case 1 applies to the remaining routing process to reach d from v.

(Case 3): Source s and destination d do not have the ancestor/descendant relationship. 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. When the spanning tree is minimum, the routing process follows a sequence of up channels until reaching either the least common ancestor of s and d or the first intermediate node (including source node s) that has a cross neighbor that 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 or down-cross channels to reach the destination. When the spanning tree is non-minimum, the routing process starts with a sequence of up channels (could be zero), followed by a cross, up, or up-cross, and ends with a sequence of down or down-cross channels.

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 or up-cross channel (if any), the routing processing ends with a sequence of down or down-cross channels (if any). Again, since down and down-cross 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 Comparisons

We first review the up*/down* routing algorithm and then compare it with the proposed prefix routing algorithm.

Up*/down* routing. The following summarizes basic steps in up*/down* routing [11]:

1. An arbitrary node is selected as a special node and then the network is partitioned into two subnetworks: *up subnetwork* and *down subnetwork*.
2. 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, a tie breaker is made by comparing the ids of two end nodes connected by the channel.
3. Every node has an n-bit string, called *reachability string*. In addition, each outgoing channel in the down direction (also called outgoing down channel) is associated with a reachability string. The length of the string is n $(= |V|)$, the number of nodes in the network. The reachability string associated with a node is formed by the logical OR of the reachability strings associated with its outgoing down channels and by setting the bit that corresponds to the node id.

4. A routing process always selects a sequence of up channels (if any) followed by a sequence of down channels (if any).

5. The routing process starts by randomly selecting an outgoing channel. This process is repeated at each intermediate node until reaching a node that has a reachability string with the bit that corresponds to the destination set to 1.

To establish the status of up/down channels, a minimum spanning tree algorithm is used at the special node (the root). As a result, each node u is assigned a level, $l(u)$, the distance to the root. A channel from u to v, (u, v), is labeled "down" if $l(u) < l(v)$ and it is labeled "up" if $l(u) > l(v)$. In the case of $l(u) = l(v)$, (u, v) will be labeled "down" if $id(u) < id(v)$; otherwise, it is labeled "up". To construct reachability strings, the following procedure is followed:

1. Assign each node in the network a distinct id, ranging from 1 to n, where n is the number of nodes in the network.

2. The formation of reachability strings starts at a node (or more than one node) that has no outgoing down channel. The reachability string of the node is derived by first defining an n-bit string of all 0's and then by setting the bit whose index in the string matches the node id. This reachability string is sent to all the outgoing up channels of the node.

3. When a node v receives a new string from an incoming up channel (u, v), it copies the string to the corresponding outgoing down channel (v, u) as its reachability string.

4. Once a node receives a string from each of its incoming up channels, these strings are logically ORed together to form a new string. The reachability string associated with the node is derived by setting the bit (whose index in the string matchs the node id) in the new string. The reachability string is then forwarded to all the outgoing up channels of the node.

Figure 3 shows the result of applying up*/down* routing to the example of Figure 3. Assume that $id(a) < id(b) < id(c) < id(d) < id(e) < id(f)$. That is, $id(a) = 1$, $id(b) = 2$, $id(c) = 3$, $id(d) = 4$, $id(e) = 5$, and $id(f) = 6$. The reachability string contains 6 bits, one for each a, b, c, d, e, and f (from left to right). Assume that the same spanning tree shown in Figure 3 is applied, and therefore, nodes a, b, c, d, and e are assigned levels 0, 1, 1, 2, 2, 2, respectively. The status of each channel (see Figure 3) is then decided based on the levels of two end nodes. Clearly, nodes e and f do not have outgoing down channels and they (simultaneously) start the process. Figure 3 only shows reachability strings of nodes. The reachability string of each down channel (u, v) is the same as the one for node v.

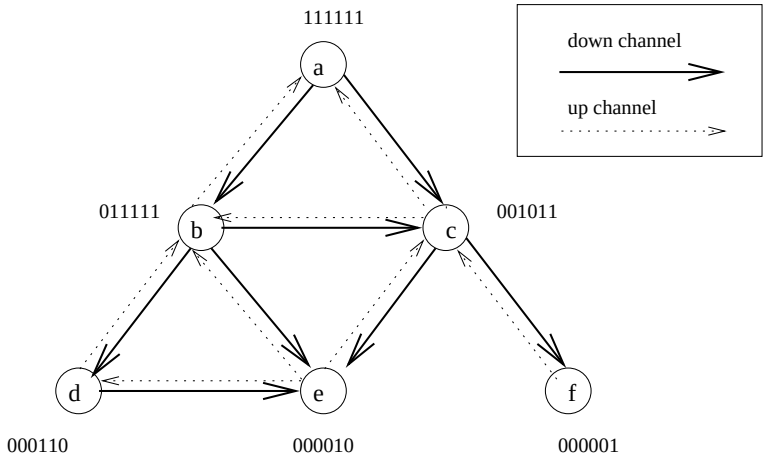

Figure 3: Up*/down* routing in the example of Figure 2.

The destination address is represented by a *destination string*, an n-bit string with all 0's except one 1 (whose index in the string matches the id of the destination). The routing process consists of two phases: up phase and down phase. During the up phase (using up channels), the routing process is random until reaching an intermediate node with a reachability string that contains the destination string. The formation of the reachability strings (associated with nodes) ensures that an intermediate node that meets the above condition exists (i.e., it is starvation-free). Once such an intermediate node is reached, the routing process is switched to the down phase. During the down phase (using down channels), the routing process is controlled by reachability strings associated with nodes and down channels. Basically, the special node is the *sink* of the up subnetwork and is the *source* of the down subnetwork. In addition, both up and down subnetworks are acyclic, routing is deadlock- and livelock-free.

In the example of Figure 3, assume node e is the source and node f is the destination. Three possible paths can be generated using the up*/down* routing: $e \rightarrow d \rightarrow b \rightarrow c \rightarrow f$; $e \rightarrow b \rightarrow c \rightarrow f$; $e \rightarrow c \rightarrow f$. Note that following the prefix routing algorithm, only one path is generated: $e \rightarrow c \rightarrow f$.

Comparison. Both prefix routing and up*/down* routing are based on string matching and employ two phases in the routing process: up phase and down phase. Prefix routing uses prefix string matching (i.e., a label is a prefix of the destination label) and up*/down* routing uses string containment (i.e., the destination string is contained in a reachability string).

We compare these two routing algorithms based on the cost of preprocessing and of the routing process. Preprocessing includes formation of a spanning tree, identification of channel

status (up/down in both routing algorithms and cross in prefix routing), and calculation of labels in prefix routing and reachability strings in up*/down* routing. Prefix routing needs only the construction of a spanning tree. Node and channel labels are determined during the formation of the spanning tree. Up*/down* routing also requires the construction of a spanning tree to determine up/down channels. Then a separate process is needed to determine reachability strings. Since each link will be visited once, the complexity of this process is $O(|E|)$. Clearly, prefix routing requires a simpler preprocessing than up*/down* routing.

In terms of storage requirement for strings and labels, each reachability string in up*/down* routing takes $n(= |V|)$ bits, while the length of each label in prefix routing varies but is bounded by $diam(G)\log\Delta$. Note that $diam(G)\log\Delta$ is in general smaller than n, especially in a dense graph where $diam(G)\log\Delta$ is close to $\log n$. Therefore, prefix routing requires less memory storage than up*/down* routing. Note that both routing algorithms are based on string matching, the shorter the string the quicker the matching process.

Both routing algorithms are non-optimal (i.e., the routing path is not necessarily shortest). In addition, they follow the up and then the down phases. However, in up*/down* routing, both the up and down phases follow a random process. For example, it randomly selects an outgoing up channel in the up phase. During the down phase, although the selection of an outgoing down channel is based on the associated reachability string, the selection is not unique (i.e., several outgoing down channels may meet the string containment requirement). The advantage of this flexibility is to fully use available channels. The down side is that the length of the routing path varies depending on the actual channel selection during the run time. In prefix routing, both the up and down phases are deterministic, and therefore, the length of the routing path is predictable to a certain accuracy by the source and destination labels. In fact, the length of a path from s to d is upper bounded by $|s^{'}| + |d^{'}|$ if the spanning tree is minimum, where $s = longest\_common\_prefix(s,d)||s^{'}$ and $d = longest\_common\_prefix(s,d)||d^{'}$. That is, the length of the path is bounded by the cardinality of both strings after removing the longest common prefix string. This is because that the $longest\_common\_prefix(s,d)$ is simply the label for the closest common ancestor, say x, of s and d. $|s^{'}|$ is the length of the tree path from s to x, and $|d^{'}|$ is the length of the tree path from x to d. For example, if $s = 111$ and $d = 121$, the longest common prefix string is 1, and therefore, $s^{'} = 11$ and $d^{'} = 21$. The length of the path is bounded by $2 + 2 = 4$. Note that if a cross channel exists, the corresponding shortcut will reduce the length of the path.

One potential problem with prefix routing is that channels are not evenly utilized, that is, up/down channels are heavily used while cross channels are rarely used: at most one cross channel is used for each routing. This will be the focus of our future work.

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. Also, we plan to extend the proposed work by improving channel utilization and reducing the length of a routing path. Possible solutions include routing with multiple cross channels, multiple spanning trees (including edge-disjoint spanning trees [14] and [15], escape channels [2], and constructing minimum spanning trees.

References

[1] 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.

[2] J. Duato. Deadlock-free adaptive routing algorithms for multicomputers: evaluation of a new algorithm. *Proc. of the 3rd IEEE Symp. on Parallel and Distributed Processing.* 1991,840-847.

[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] T. C. Lee and J. P. Hayes. A fault-tolerant communication scheme for hypercube computers. *IEEE Transactions on Computers.* 41, (10), Oct. 1992, 1242-1256.

[6] R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan. Tree-based multicasting in wormhole-routed irregular topologies. *Proc. of the First IPPS/SPDP.* April 1998, 244-249.

[7] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishing, Inc., 1996.

[8] H. Park and D. P. Agrawal. Generic methodologies for deadlock-free routing. *Proc. of the 10th International Parallel Processing Symposium.* April, 1996, 638-643.

[9] 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.

[10] 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. I, Aug. 1996, 46-60.

[11] M. Schroeder and et al. 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.

[12] F. Silla and J. Duato. Is it worth the flexibility pro- vided by irregular topologies in networks of worksta- tions? *Proc. of Workshop on Communications, Ar- chitecture, and Applications for Network-based Paral- lel Computing (CANPC'99).* Jan. 1999.

[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] H. Wang and D. M. Blough. Multicast in wormhole-switched networks using edge-disjoint spanning trees. *Journal of Parallel and Distributed Computing.* Accepted to appear.

[15] N. C. Wang, T. S. Chen, and C. P. Chu. Improving tree-based multicasting for wormhole switch-based networks. *Proc. of ISCA International Conference on Parallel and Distributed Computing Systems.* Aug. 2001, 13-18.

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