
The Journal of Supercomputing, 31, 63–78, 2005

Deadlock-Free Multicasting in Irregular Networks Using Prefix Routing*

JIE WU
Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA

LI SHENG
Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA 19104, USA

Abstract. A deadlock-free multicast scheme called prefix multicasting in irregular networks (i.e., networks with irregular topology) is studied. In prefix routing, a compact routing table is 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 labelling in an irregular network is based on a pre-defined spanning tree which may or may not be minimum. The routing process follows a two-phase process of going up and then down along the spanning tree, with a possible cross channel between two branches of the tree between two phases. It is shown that the proposed routing scheme is deadlock- and livelock-free. The approach is extended to multicasting in which the multicast packet is first forwarded up the tree to the *longest common prefix* (LCP) of destinations in the multicast. The packet is then treated as a multi-head worm that can split at branches of the spanning tree as the packet is sent down the tree.

Keywords: deadlock, multicasting, multiprocessor systems, prefix routing

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 [4], and ATM. In general, switch-based networks provide virtual point-to-point communication, and hence, offer better throughput and lower latency for many applications. Systems area network (SAN), one of the crucial components of network-based parallel processing [16], consists of switches connected with point-to-point links. The emerging InfiniBand Architecture (IBA) [1], is also designed around a switched-based interconnect technology with high-speed point-to-point links. Currently, there are around 200 partners in the InfiniBand Trade Association. Some proposals are also given to enhance performance of the InfiniBand Architecture [9].

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 a source node to one or more destination nodes in a given system. Multicasting is a routing process involving one source and multiple

*This work was supported in part by NSF grants CCR 0329741, ANI 0093936, EIA 0130806.

destinations. Unicasting is a special case of multicasting with only one destination. Multicast communication is essential for many useful operations such as barrier synchronization, DSM systems, and collective communication in MPI and PVM.

Most recent research in the field focused on deadlock-free multicasting with wormhole switching. In wormhole-routed systems, a packet is divided into a sequence of flits and these flits are forwarded to destinations in a pipeline fashion controlled by a header flit. The packet as a whole can be viewed as a multi-head worm that can split into multiple heads. Suppose a packet to be sent to multiple destinations follows a tree that covers all destinations (the corresponding approach is called tree-based routing), the multi-head worm can split at each branch of the tree. A multi-head worm can be implemented either synchronously or asynchronously [13]. In a synchronous multi-head worm, no branch of the multi-head worm can proceed until all output channels are available. In an asynchronous multi-head worm, each branch can forward independently without coordinating with branches as shown in Figure 1 (a flit with label 1 is the header). Each header is pseudo independent. Since each header advances asynchronously, space may exist between two adjacent flits along a branch. Such space is filled with dummy flits called bubbles (see Figure 1).

A *deadlock* occurs when several routing processes are in a circular waiting state and cannot advance toward their destinations because the channels required by them are not available. A *livelock* occurs when a routing process travels around its destination node and never reaches it. Deadlock- and livelock-free routing in regular topologies such as hypercubes [10] and meshes [14] has been extensively studied. Unlike regular topologies, irregular topologies pose some new challenges to design a deadlock- and livelock-free routing process:

- It is difficult, if not impossible, to derive an efficient routing scheme without using a complete routing table.

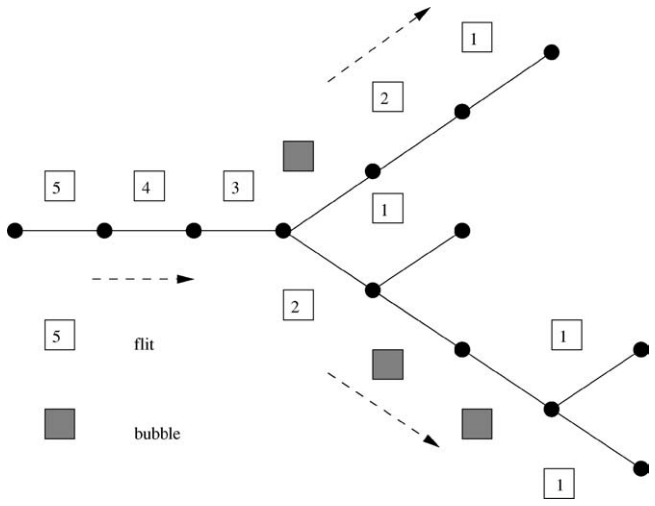

Figure 1. An asynchronous multi-head worm.

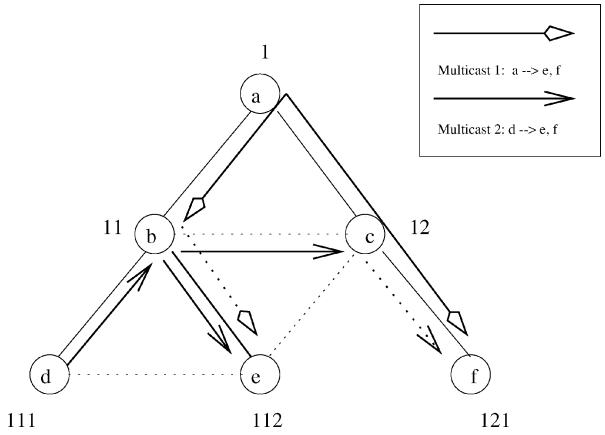

Figure 2. An example of deadlock involving two multicasts.

- Irregular routing paths (because of irregular topologies) pose an additional dimension of difficulty to ensure deadlock- and livelock-freedom.

Multicasting in wormhole-routed networks itself adds more difficulties in preventing deadlock which can occur even when the underlying base-line unicasting is deadlock-free. Specifically, deadlock could happen under either the synchronous or asynchronous model of multiple-head worms. Figure 2 shows a deadlock situation between two multicasts. Multicast 1 sends a packet from node a to nodes e and f, while Multicast 2 sends a packet from node d to nodes e and f. Multicast 1 is waiting for Multicast 2 to release channel (c, f), while Multicast 2 is waiting for Multicast 1 to release channel (b, e). In this case, these two multicasts wait for each other to release channels and they are in a deadlock state. Note that in this case the same routing policy for unicasting is deadlock-free. That is, the unicasting follows up channels of the spanning tree (tree links are represented as solid lines in Figure 2), one cross channel, and down channels.

Most deadlock-free routing/multicasting algorithms for irregular networks are based on either a spanning tree (the corresponding approach is called tree-based) [11, 20] or a Eulerian path (the corresponding approach is called path-based) [17]. By restricting the routing path along branches in the spanning tree (with limited and controlled *jumps* between tree branches), deadlock-free routing is derived. In a path-based approach, a Eulerian path (a path that visits each edge once and only once) 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 for generating short routing paths. However, to ensure deadlock-freedom in multicasting, the tree-based approach is more involved [11].

In this paper, a deadlock-free multicast scheme called *prefix multicasting* in irregular networks (i.e., networks with irregular topology) is studied. Prefix routing [23] is a special type of routing with a compact routing table associated with each node (processor). A compact routing table is a reduced-size routing table. 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 labelling in an irregular network is done based on a pre-defined spanning tree that may or may not be minimum. The routing process follows a two-phase process of first going up and then going down the spanning tree, with a possible cross channel between two branches of the tree between two phases. It is shown that the prefix routing scheme [23] is deadlock- and livelock-free under either the synchronous or asynchronous model. In this paper, prefix unicasting is extended to multicasting in which the multicast packet is first forwarded to the *longest common prefix* (LCP) of destinations in the multicast. The packet can then be treated as a multi-head worm that can split at branches of the spanning tree as the packet is sent down the spanning tree. Comparison between prefix multicasting and Libeskind-Hadas et al's approach for multicasting is given. Possible extensions are also discussed including reliable prefix routing and multicasting.

The paper is organized as follows. Section 2 summarizes related work. Section 3 reviews the prefix routing algorithm. Several new features are revealed together with a new proof of the livelock- and deadlock-free property. Discussion is included on how to carry out prefix routing when the spanning tree is non-minimum. Section 4 proposes the livelock- and deadlock-free prefix multicasting. Section 5 compares prefix multicasting with Libeskind-Hadas et al's approach for multicasting and discusses possible extensions, including reliable prefix routing and multicasting. An efficient distribution of label table is also discussed. Finally, Section 6 concludes this paper and discusses possible future work.

2. Related work

In tree-based unicasting, a spanning tree is first constructed and the packet is forwarded to its destination by going up/down the tree. In [19], an up*/down* unicast algorithm is proposed aiming at better utilization of all the available channels in the network. First, an arbitrary node is selected as a special node (root node), and then, the network is partitioned into two subnetworks: *up subnetwork* and *down subnetwork*. The up subnetwork consists of unidirectional channels directed towards a special node (root 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 (assume that node ids are distinct). 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. It is impossible to have a cycle that involves up and down channels, because a transition from a down channel to an up channel is forbidden.

To determine the status of each channel (up or down), a *minimum spanning tree* is constructed from the special node (root node) that connects each node through a shortest path in terms of hop count. The status of a channel in the spanning tree can be easily determined. A channel not in the tree is called a *cross channel* and its status, *up cross* or *down cross*, can be determined by the levels of the two end nodes (of the channel) in the spanning tree. When the levels of two end nodes are the same, node ids are used to break a tie.

The routing in Autonet [19] was built based on the up*/down* unicast algorithm. The packet is first routed in the up subnetwork (consists of up and up cross channels) and is then routed in the down subnetwork (consists of down and down cross channels). Basically, the up subnetwork is a directed acyclic graph (DAG) with a unique sink which is the root of the spanning tree. The down subnetwork is also a DAG; however, in order to direct a packet to its destination, a *reachability string* is attached to each node in the down subnetwork. More specifically, each header of a packet is associated with a bit-string of length n to represent distribution of destinations, where n is the number of nodes in the network. Every node in the down subnetwork has an n-bit reachability string associated with every one of its outgoing channels that lead to channels in the down direction. These reachability strings can be constructed during the formation of the spanning tree. This approach resembles the traditional table lookup approach; however, table lookup cancels out some elegant features of the approach.

Unfortunately, the up*/down* unicasting cannot be directly applied to multicasting because of excessive freedom in using different types of channels. There are two tree-based multicast schemes in literature. Deadlock-freedom is ensured by restricting the way a channel is used. In [8], unicasting is allowed to use cross channels (ones that are not in the spanning tree) as shortcuts to reach destinations through shorter paths. Multicasting is restricted to tree channels only. Whenever a multicast packet is blocked by a unicast packet at an intermediate node, it has to suspend its transmission and yield its path to the unicast packet. Another option is to exchange the role of multicasting and unicasting when they are competing for channels. Libeskind-Hadas et al. [11], propose a scheme that distinguishes between down tree channels and down cross channels. Unlike the up*/down* unicast algorithm, a multicast packet is routed along zero or more channels in the up subnetwork (consisting of up and up cross channels), followed by zero or more down cross channels, and finally zero or more down tree channels. A multicast packet uses this approach to reach the least common ancestor (LCA) of its destination set. It is restricted to down tree channels after that. Note that in a unicasting, the destination itself is the least common ancestor.

The proposed prefix multicasting is an elegant approach that works for both unicasting and multicasting without distinguishing them. This approach is based on the notion of *compact routing* that uses a routing table with reduced size [7]. Two commonly used compact routing schemes are *interval routing* [2, 6] and *prefix routing* [3]. 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 string and each node is also labelled with a 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.

3. Prefix routing

Suppose $G = (V, E)$ is an undirected graph representing an irregular network, where V is the vertex set and E is the edge set. $v \in V$ represents a vertex (also called a node) in the

network and uv represents an edge (also called 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 call each of them a node. Each link uv has two unidirectional channels: (u, v) and (v, u). Prefix routing is based on a labelling scheme that assigns a label to each node and channel. $L(v)$ and $L(u, v)$ are 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. The unicasting consists of two steps [23]:

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 (root node).
- Assign labels to nodes and channels of the spanning tree during its formation.
- Complete labels to all the remaining channels in the graph.

Routing: Construct a unicast algorithm.

- Suppose d is the destination and v is the current node, node u is selected as the forwarding node if $L(v, u)$ is a prefix of $L(d)$.

3.1. Building a spanning tree

The spanning tree can be built using one of the traditional breadth-first search (BFS) methods. Initially, all nodes are unmarked. The process starts from a selected node, r, called the 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 the signal to its adjacent nodes $adj(v)$. A marked node ignores any signal received. This process generates a minimum spanning tree if it is implemented by a centralized algorithm using a priority queue, i.e., each node is reached from the root through a shortest path in terms of hop count. In a decentralized implementation, a minimum spanning tree can be constructed if the 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; the way a spanning tree is constructed is no longer an issue. 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 initiates its own process (could be at the same time); however, only one winner (which becomes the root node) is selected [15]. More general ways of constructing a spanning tree can be found in [12]. Based on the definition of a spanning tree, we can define four types of channels: *up, down, cross,* and *shortcut*. Shortcut channels are further divided into *up_shortcut* and *down_shortcut* as shown in Figure 3.

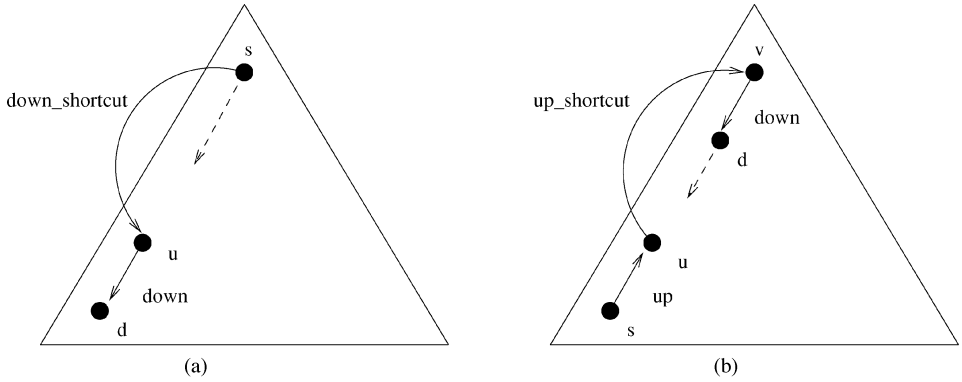

Figure 3. Two cases of shortcuts: (a) down_shortcut and (b) up_shortcut.

3.2. Assignment of labels to nodes and channels

The labelling scheme is extended from the one in [3]. Assignment of labels to nodes and channels of the spanning tree is done as follows: The label of the root is 1, that is, $L(r) = 1$. 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 parent 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 labelling scheme, each node v decides its label and labels for channels (v, u), where $u \in adj(v)$.

The labelling of channels that are outside the spanning tree is based on labels of two end

- **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, but it connects two nodes at different branches of the tree. (We do not distinguish the up/down status of a cross channel.)
- **shortcut**: a channel that is not in the spanning tree, but it connects two nodes in the same branch of the tree. (Such a channel occurs only in a non-minimum spanning tree.)
  - **up_shortcut**: a shortcut that directs towards the root.
  - **down_shortcut**: a shortcut that directs away from the root.

nodes: If there is no parent-child relation between v and u and $uv \in E$, then $L(v, u) = L(u)$ and $L(u, v) = L(v)$.

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

3.3. Unicast algorithm

The routing process is decentralized. At an intermediate node (also called a forwarding node), it decides the next forwarding node by selecting an appropriate outgoing channel.

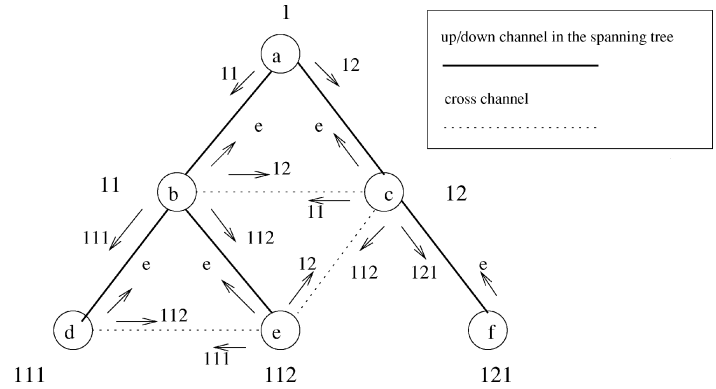

Figure 4. Labels for nodes and channels.

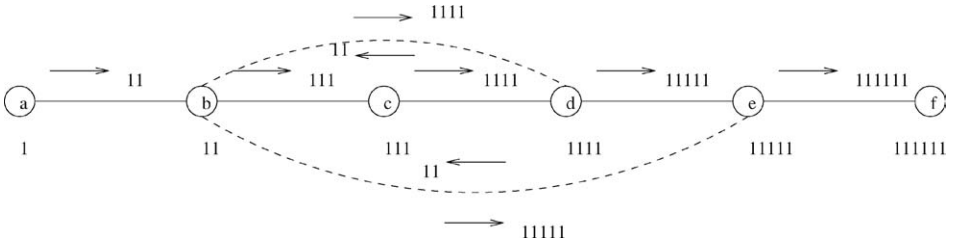

Figure 5. A sample linear network.

Basically, the prefix routing 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. When there are several forwarding nodes (in a non-minimum spanning tree), the one that has the longest prefix of the destination is selected. If there is no outgoing channel that has a label matching the label of the destination, an up channel (with label e) is selected. If the spanning tree is minimum, a routing process proceeds by visiting a sequence of up channels (if any), followed by at most one cross channel, and ends with a sequence of down channels (if any).

In Figure 4, an irregular network with six nodes is shown, together with a minimum spanning tree with its links represented as solid lines. By applying the proposed prefix routing algorithm, path $11 \rightarrow 12 \rightarrow 121$ for $(s, d) = (11, 121)$ and path $112 \rightarrow 12 \rightarrow 121$ for $(s, d) = (112, 121)$ can be derived. Note that we use s to represent both the source node and its label $L(s)$. Figure 5 shows another irregular network with a non-minimum spanning tree rooted at node a. Actually, this non-minimum spanning tree is a linear line

Unicast algorithm

- 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 there are several forwarding nodes, the one that has the *longest prefix* of the destination is selected.
- If such a neighbor does not exist, select a neighbor w such that $L(v, w) = e$.

Table 1. A label table associated with each node of Figure 3

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

and the corresponding routing is path-based. Using the prefix routing, we can derive path $1 \rightarrow 11 \rightarrow 11111 \rightarrow 111111$ for $(s, d) = (1, 111111)$. At node 11, nodes 111, 1111 and 11111 are eligible forwarding nodes. Node 11111 is selected because it has the longest prefix of the destination. For $(s, d) = (11111, 111)$, the corresponding path is $11111 \rightarrow 11 \rightarrow 111$.

It is assumed 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 of Figure 4. 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 pairs of node id and label, a label table is constructed. Finally, r broadcasts the label table down the spanning tree.

For a minimum spanning tree, we can easily see that any routing process proceeds by visiting a sequence of up channels (if any), followed by at most one cross channel, and ends with a sequence of down channels (if any). Since any sequence of up (down) channels is acyclic, the routing process is deadlock-free [23]. For a non-minimum spanning tree, the case is more involved as shown in the following three cases. Note that based on the channel labelling 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.

Case 1: Source s is an ancestor of destination d. Based on the channel labelling scheme, destination d can be reached through a unique sequence of down channels. We now show that it is the only 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 labelling scheme, both w and v are ancestors of destination d, which is a contradiction. If the spanning tree is non-minimum, down_shortcuts may be used. A down_shortcut (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. (Figure 3(a) shows such a case.) Note that if the spanning tree is minimum, i.e., 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 ($\cdots \rightarrow v \rightarrow u \cdots \rightarrow d$) than the current one ($\cdots \rightarrow v \cdots \rightarrow u \cdots \rightarrow$). Suppose "+", "$*$", and "|" represent "one or more", "zero or more", and "or", respectively. The above cases can be summarized as follows:

1. minimum spanning tree: $down^+$
2. non-minimum spanning tree: $(down\_shortcut \mid down)^+$

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 unique 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_shortcut 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_shortcut channels. Note that if the spanning tree is a minimal one, that is, each node is reached from the root of the spanning tree through a shortest path, the above case will never occur, because otherwise node s can be reached from the root through a shorter path ($\cdots \rightarrow v \rightarrow u \cdots \rightarrow s$) than the current one ($\cdots \rightarrow v \rightarrow \cdots \rightarrow d \rightarrow \cdots \rightarrow u \cdots \rightarrow s$). 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 reaches 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. The above cases can be summarized as follows:

1. minimum spanning tree: up^+
2. non-minimum spanning tree: up^+ or $up^* \rightarrow up\_shortcut \rightarrow (down\_shortcut \mid down)^*$

Case 3: Source s and destination d do not have the ancestor/descendant 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. 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 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_shortcut, and ends with a sequence of down or down_shortcut channels. The above cases can be summarized as follows:

1. minimum spanning tree: $up^* \rightarrow cross \rightarrow down^*$ or $up^+ \rightarrow down^+$
2. non-minimum spanning tree: $up^* \rightarrow cross \rightarrow (down\_shortcut \mid down)^*$, $up^+ \rightarrow (down\_shortcut \mid down)^+$, or $up^* \rightarrow up\_shortcut \rightarrow (down\_shortcut \mid down)^+$.

Theorem 1 *The proposed prefix routing is deadlock- and livelock-free.*

Proof: Based on the definition of the proposed routing algorithm, the routing process starts with a sequence of up channels (if any). Therefore, two stages are used: stage 1 uses up channels only and stage 2 uses other channels. Stage 1 completes in a finite number of steps, because the subgraph induced from up-channels is a tree. Stage 2 uses channels in an increasing order in terms of lengths of channel labels and the message always gets closer to the destination at each step. Because the up channels at stage 1 belong to the up subnetwork which is a DAG, there is no cyclic wait among these channels. Stage 2 is also cycle-free as shown earlier. Therefore, the proposed algorithm is deadlock-free. At stage 1, because the number of up channels are finite and they belong to a DAG with a unique sink

(root node), in the worst case the packet is sent to the root with its label being a prefix of the label of any destination. Stage 2 uses prefix matching and it will eventually terminate because the id (string label) of each node is finite. Therefore, the proposed algorithm is livelock-free. □

4. Prefix multicasting

The prefix routing cannot be directly extended to prefix multicasting. Suppose we use a simple extension to tree-based multicasting that uses a multi-head worm, the worm splits at a node whenever destinations in the multicast have to follow different branches, based on prefix matching of labels defined in prefix routing, through different output channels. That is, if the label of one of the outgoing channels is a common prefix of a proper subset of destinations, then this subset is sent along that outgoing channel. In Figure 2, suppose the spanning tree is shown as solid lines, a multicast from node d to nodes e and f is first sent to node b via channel (d, b) with an empty string label. Once at node b, the worm splits into two heads, one towards node c (and then to node f) and the other one towards node e. As shown earlier, this approach will cause deadlock. A deadlock may occur when there is another multicast from node a to nodes e and f as shown in Figure 2.

The proposed algorithm, called prefix multicasting, consists of two phases: *up phase* (with a single-head worm) and *down phase* (with a multi-head worm). The algorithm restricts the location of each split to avoid deadlock. Specifically, no split is allowed until the worm reaches a node that is the *longest common prefix* (LCP) of destinations in the multicast.

Multicast algorithm.
Up phase:

- Apply prefix routing and use LCP to direct the single-head worm until reaching a node with label LCP.

Down phase:

- The single-head worm splits at LCP to a multi-head worm as needed.
- The head of each multi-head worm may split further at a branch of the spanning tree. If there are several forwarding nodes (when the spanning tree is non-minimum), the one that has the *longest prefix* of the destination is selected.

The LCP of destinations is calculated first at the source node and attached to the packet during the transmission. Once passing the LCP, the multi-head can split without further restriction. Each destination is associated with only one head after the split. Also, the address of a destination is removed from the header once the destination is reached. Note that when the spanning tree is minimum, each destination has one forwarding node at each step of down phase. However, when the spanning tree is non-minimum, a destination may have several forwarding nodes. In this case, the one that has the *longest prefix* of the destination is selected. It can be easily proved that when the above rule is followed, two heads of the same message will not involve in a *self-lock* situation. Two heads of the same message are

said to be self-locked if they compete for channel(s) at a point; the block of one head will eventually cause the blocking of the other.

To ensure freedom from deadlock, a request for a set of output channels at each node is assumed to be an *atomic operation* like the one used in [11]. That is, a worm entering a node waits until all the requests for FIFO output channel request queues have arrived at the heads of queues and these channels become free.

Theorem 2 *The proposed prefix multicasting is deadlock- and livelock-free.*

Proof (Sketch): Consider two multicast messages m_1 and m_2 with LCP(m_1) and LCP(m_2) as their longest common prefixes, respectively. It is clear that deadlock between m_1 and m_2 will not occur at the up phase since each message is a single-head worm at the upper phase. Therefore, we only need to focus on the down phase.

The following two cases are considered: (1) LCP(m_1) and LCP(m_2) do not have the ancestor and descendant relation. In this case, m_1 and m_2 do not compete for any channel during the down phase, and hence, no deadlock will occur. (2) LCP(m_1) and LCP(m_2) have the ancestor and descendant relation. Without loss of generality, we assume that LCP(m_1) is the ancestor of LCP(m_2). Note that this case also includes when LCP(m_1) and LCP(m_2) point to the same node. This tie situation is broken by the timestamp associated with each message, where timestamp is the time the message is generated. Clearly, a branch of m_1 is competing with m_2 *only at node* LCP(m_2). Since the output channels are assigned as an atomic operation, either m_2 or a branch of m_1 gets all the needed channels of LCP(m_2) or no channels at all. If m_2 get all channels of LCP(m_2), we say m_1 waits for m_2 at point LCP(m_2); otherwise, m_2 waits for m_1 at point LCP(m_2). In either case no deadlock will occur.

Assume there is a deadlock involving k multicast messages, $m_1, m_2, \ldots, m_k$. Without loss of generality, m_i waits for m_{i+1} at point (node) p_i for $i = 1, 2, \ldots, k-1$ and m_k waits for m_1 at p_k. Based on the organization of nodes in a tree structure, p_i is higher than p_{i+1} for $i = 1, 2, \ldots, k-1$, that is, p_i is closer to the root of the tree than p_{i+1} is. In addition, p_k is higher than p_1. This is a contradiction. □

In the example of Figure 2, a packet from node d to nodes e and f cannot split before reaching their LCP(112, 121) = 1 which is the label for node a. Once the packet reaches node a, it splits into two heads, one follows channel (a, b) with label 11 to reach destination e (with label 112) via node b and another follows channel (a, c) with label 12 to reach destination f (with label 121) via node c. A packet from node a to nodes e and f can follow the multicast tree as shown in Figure 2, because node a is the LCP of nodes e and f and no restriction on splitting is needed. Clearly, no deadlock will occur between these two multicasts.

5. Comparisons and extensions

5.1. Comparisons

Both prefix multicasting and Libeskind-Hadas et al's approach for multicasting use route information represented as a string to direct a packet to destinations and both employ two

phases in the routing process: up phase and down phase. Prefix multicasting uses prefix string matching (i.e., a label is a prefix of the destination label) and Libeskind-Hadas et al's approach 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 and down in both routing algorithms and cross and shortcut in prefix multicasting), calculation of labels in prefix multicasting and reachability strings in Libeskind-Hadas et al's approach for multicasting. Prefix multicasting needs only the construction of a spanning tree. Libeskind-Hadas et al's approach for multicasting requires the construction of a spanning tree to determine up/down channels. At the same time, reachability strings need to be constructed. Specifically, the following procedure is followed to construct reachability strings: (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 matches the node id) in the new string. The reachability string is then forwarded to all the outgoing up channels of the node.

On the other hand, labelling of nodes and channels in prefix multicasting is determined locally. This locality offers a convenient way of reconfiguration in a dynamic setting where links are subject to failure. The extension of prefix routing and multicasting in an unreliable system will be discussed in the next section. Clearly, prefix multicasting requires a simpler preprocessing than Libeskind-Hadas et al's approach for multicasting.

One disadvantage of prefix multicasting is its limited flexibility in the up phase. That is, fewer channels are utilized in this phase compared with the ones used in the Libeskind-Hadas et al's approach.

5.2. Extensions

Two directions of extensions are considered [23]: (1) Improve channel utilization and (2) reduce the length of a routing path. Four possible extensions are discussed in [23] for prefix routing:

- routing with multiple cross channels
- multiple spanning trees
- escape channels
- minimum depth spanning trees

All these extensions can be applied to prefix multicasting. Multiple cross channels can be used in the up phase, like the approach used in [11]. However, more labels will be associated

with channels. Multiple spanning trees can be used to increase channel utilization. The method proposed by Roskind and Tarjan [18] can be applied to find multiple edge-disjoint spanning trees for an irregular network. In a twisted version of adaptive routing [5], a hybrid routing can be introduced using two routing processes: One is fully adaptive that uses virtual channels labelled as *nonwaiting*, and the other is restrictive but deadlock-free routing that uses virtual channels labelled as *waiting* (also called *escape channels*). Initially, the fully adaptive routing is used until it is blocked; then the hybrid routing is switched to the restrictive routing. Clearly, channels (up, down, cross, shortcut) used in prefix multicasting can be used to form escape channels. In the proposed routing algorithm, both the length of each label and the routing path depend on the depth of the minimum spanning tree. We can use one of the existing algorithms to construct a minimum depth spanning tree. In some cases these extensions can be considered at the same time. For example, Wang and Blough [21] propose a way to find two spanning trees in a 2-D torus with small diameters. More recently, Wang et al. [22] propose to use two virtual channels to support two spanning trees in any irregular networks: one tree is left-to-right BFS (breadth-first search) tree and the other is right-to-left BFS tree.

We focus here on an extension for reliable prefix routing/multicasting. It is assumed that only links are subject to failures and a faulty link is detected by one of its end nodes. Also each node u checks the status of its output channels at a certain time interval. When a faulty output channel is detected at node u, one of following two actions is carried out.

- If the faulty output channel is not in the spanning tree, the label of this channel is removed and nothing else needs to be done.
- If the faulty output channel is in the spanning tree, node u initiates the process of constructing a new spanning tree. Any uncommitted packets will be discarded and re-sent later following the new spanning tree.

The locality property of node/channel labelling in prefix routing/multicasting ensures a simple reconfiguration strategy, especially when faulty links are not in the spanning tree. When a faulty link is in the spanning tree, an alternative approach is to delay the construction process for a new spanning tree. The call for a new spanning tree is initiated only when a packet that needs to use the faulty channel arrives at one of its end nodes.

In the example of Figure 6(a), if channel (d, e) fails, it is removed and nothing else needs to be done. However, if channel (b, e) in the spanning tree fails, node b (which detects the failure) initiates a process to generate a new spanning tree rooted at node b. The resultant spanning tree together with a new label for each node is shown in Figure 6(b).

Another possible extension is to consider dynamic reconfiguration where the network reconfiguration is performed without stopping the transmission of user packets. However, the assurance of the freedom from deadlock and livelock is significantly harder than a system that uses static reconfiguration.

5.3. Distribution of label table

The knowledge of label table at each node is essential to carry out the prefix routing/multicasting process. This global information limits the applicability of prefix routing

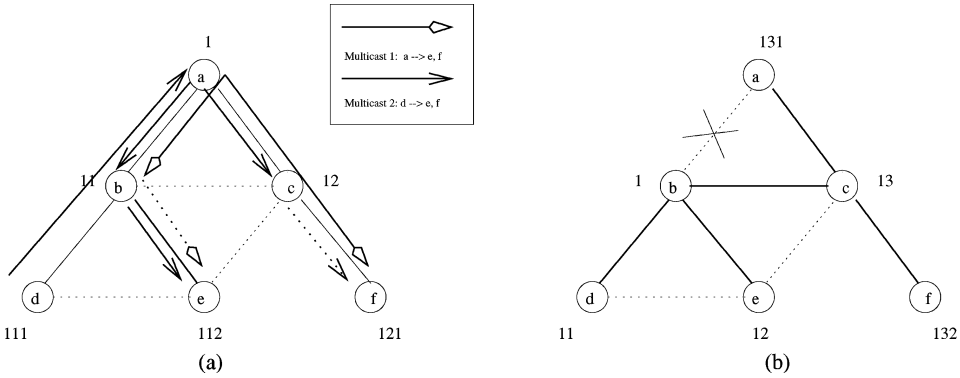

Figure 6. (a) Two multicast trees. (b) A new spanning tree initiated at node b.

(multicasting), especially in a dynamic network with faulty nodes/links or even mobile nodes. Fortunately, the inherit hierarchical structure of prefix routing supports a distributed implementation of label table. Instead of keeping a global label table at each node, each node v only needs to keep a *partial label table* that includes only descendant nodes (i.e., labels of these nodes contain the label of v as a prefix). For example, node b in Figure 3 keeps a label table containing ones for nodes b, d, and e. When an intermediate node v receives a routing packet, it first checks to see if the destination in unicasting (LCP in multicasting) is in its label table. If the destination is in its local table, the proposed prefix unicasting (multicasting) is followed; otherwise, the packet is sent to a neighbor w such that $L(v, w) = e$.

6. Conclusions

In this paper, we have extended a prefix-based routing scheme to prefix multicasting in irregular networks and shown that it is deadlock- and livelock-free. Unlike traditional path-based and tree-based routing, prefix multicasting is based on a simple labelling scheme and labels to nodes and channels are assigned during the formation of a spanning tree. Both prefix multicasting and Libeskind-Hadas et al's approach for multicasting are based on string matching during the routing process. Prefix routing can be considered as a complement to Libeskind-Hadas et al's approach. Our future work will focus on comparing the proposed scheme with existing ones, such as Libeskind-Hadas et al's approach, through simulation.

References

1. I. T. Association. InfiniBand architurecture. Vol. 1, release 1.0.a, 2001, also available at http://www.infinibandta.com.
2. E. M. Bakker and J. van Leeuwen. Linear interval routing. *Algorithms Review*, 45–61, 1991.
3. E. M. Bakker, J. van Leeuwen, and R. B. Tan. Prefix routing schemes in dynamic networks. *Computer Networks and ISDN Systems*, 26:403–421, 1993.

4. N. J. Boden et al. Myrinet: A gigabit-per-second local area network. *IEEE Micro*., 15(1):29–35, 1995.
5. 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, pp. 840–847.
6. 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, pp. 279–290.
7. 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, pp. 88–97.
8. M. Gerla, R. Palnati, and S. Walton. Encoding and decoding of address information in multicast message. *Computer Communication Review*, 26:184–194, 1996.
9. E. J. Kim, K. H. Yum, C. R. Das, M. Yousif, and J. Duato. Performance enhancement techniques for InfiniBand architecture. *Proc. of HPCA'03*. 2003, pp. 256–264.
10. T. C. Lee and J. P. Hayes. A fault-tolerant communication scheme for hypercube computers. *IEEE Transactions on Computers*, 41(10):1242–1256, 1992.
11. R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan. Tree-based multicasting in wormhole-routed irregular topologies. *Proc. of the First IPPS/SPDP*, April 1998, pp. 244–249.
12. N. A. Lynch. *Distributed Algorithms*, Morgan Kaufmann Publishing, Inc., 1996.
13. L. M. Mi. Should scalable parallel computers support efficient hardware multicast? *Proc. of the 1995 ICPP Workshop on Challenges for Parallel Processing*, 1995, pp. 2–5.
14. H. Park and D. P. Agrawal. Generic methodologies for deadlock-free routing. *Proc. of the 10th International Parallel Processing Symposium*, April, 1996, pp. 638–643.
15. R. Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. *Proc. of the 9th Data Communication Symp.*, Sept. 1985, pp. 44–53.
16. F. Petrini, W. Feng, and A. Hoisie. The Quadrics network (QsNet): high-performance clustering technology. *Proc. of Hot Interconnects*. Aug. 2001, pp. 125–130.
17. 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, pp. 46–60.
18. J. Roskind and R. Tarjan. A note on finding minimum-cost edge-disjoint spanning trees. *Math. of Oper. Res.*, 10:701–708, 1985.
19. M. Schroeder 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):1318–1335, 1991.
20. 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.
21. 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).
22. 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, pp. 13–18.
23. J. Wu and L. Sheng. Deadlock-free routing in irregular networks using prefix routing. *Proc. of the ISCA 12th Int'l Conf. on Paralell and Distributed Computing Systems*, 1999, pp. 424–430.