

A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks

Bhargav Bellur Richard G. Ogier
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract—We present, prove correctness for, and evaluate a protocol for the reliable broadcast of topology and link-state information in a multihop communication network with a dynamic topology, such as a wireless network with mobile nodes. The protocol is called Topology Broadcast based on Reverse Path Forwarding (TBRPF), and uses the concept of reverse-path forwarding (RPF) to broadcast link-state updates in the reverse direction along the spanning tree formed by the minimum-hop paths from all nodes to the source of the update. TBRPF uses the topology information received along the broadcast trees to compute the minimum-hop paths that form the trees themselves, and is the first topology broadcast protocol based on RPF with this property. The use of minimum-hop trees instead of shortest-path trees (based on link costs) results in less frequent changes to the broadcast trees and therefore less communication cost to maintain the trees. Simulations show that TBRPF achieves up to a 98% reduction in communication cost compared to flooding in a 20-node network.

Keywords—Topology broadcast, link-state routing, reverse-path forwarding, mobile network, packet-radio network.

I. INTRODUCTION

We consider the problem of broadcasting topology information (including link costs and up/down status) to all nodes of a communication network. This information, together with a path selection algorithm, can be used by each node to compute preferred paths to all destinations, i.e., to perform routing based on link states. Most link-state routing protocols, including OSPF, are based on flooding [9], [10], [12]. In these protocols, each link-state update is sent on every link of the network. Although flooding is useful in networks with high bandwidth links, it can consume a significant percentage of link bandwidth in networks containing links of relatively low bandwidth, including wireless networks.

The communication cost of broadcasting topology information can be reduced if the updates are sent along spanning trees, but there is additional communication cost for maintaining these trees. The main concern here (which is addressed in this paper) is whether the total communication cost is significantly reduced despite this additional cost. The protocol we present is based on the *extended reverse-path forwarding* ERPF algorithm [3], in which messages generated by a given source are broadcast in the reverse direction along the directed spanning tree formed by the shortest paths from all nodes to the source. ERPF assumes an underlying routing algorithm is used by each node i to select the next node $p_i(v)$ on the

shortest path to each destination (or broadcast source) v . The node $p_i(v)$ then becomes the *parent* of i on the broadcast tree rooted at source v . Each node informs its parent of this selection, so that each parent becomes aware of its *children* for each source. A node i receiving a broadcast message originating from source v from its parent $p_i(v)$ forwards the message to its children for source v (if it has children). As stated by the authors, ERPF is not reliable when the shortest paths can change due to a dynamic topology. We note that ERPF was designed for general broadcast, not specifically for topology broadcast. In fact, since ERPF is not reliable, the underlying routing algorithm cannot depend on ERPF for topology broadcast.

In this paper, we present a reliable topology broadcast protocol called Topology Broadcast based on Reverse-Path Forwarding (TBRPF). TBRPF combines the concept of ERPF with the use of sequence numbers to achieve reliability, and the computation of minimum-hop paths based on the topology information received along the broadcast tree rooted at the source of the information. Since minimum-hop paths are computed, each source node broadcasts link-state updates for its outgoing links along a minimum-hop tree rooted at the source. (Thus, a separate broadcast tree is created for each source.) The use of minimum-hop trees instead of shortest-path trees (based on link costs) results in less frequent changes to the broadcast trees and therefore less communication cost to maintain the trees.

TBRPF has the following chicken-egg paradox: it computes the paths that form the broadcast trees using information that is received along the trees themselves. Thus, the path computation and topology broadcast functions of TBRPF are highly interdependent, and the correctness of TBRPF is not obvious. We prove that TBRPF is correct in the sense that every node knows the correct topology in finite time if no topology changes occur after some time.

TBRPF is a simple, practical protocol that generates much less update/control traffic than flooding in networks with a dynamic topology. It is therefore especially useful in networks that have frequent topology changes and limited bandwidth, including packet radio networks with low-bandwidth links.

We present simulation results that show TBRPF achieves a dramatic reduction in update traffic as compared to flooding. This improvement holds both for networks with point-to-point links and for wireless networks with broadcast links. Reductions of up to 98% were obtained for point-to-point links, and

97% for broadcast links. One reason for the improvement with broadcast links is that, in TBRPF, unlike flooding, the leaves of the broadcast tree need not forward updates. In simulations, we have found that most nodes in a typical multihop radio network (having a diameter between 2 and 6) are leaves of a given min-hop spanning tree.

TBRPF can be extended to hierarchical link-state routing, in which the network is divided into areas or clusters. However, in this paper we focus on flat (non-hierarchical) networks. Because TBRPF reduces update traffic dramatically compared to flooding, it reduces the need for hierarchical routing.

Section II compares TBRPF to related protocols, Section III describes TBRPF, Section IV proves correctness for TBRPF, Section V derives the communication, time, computational, and storage complexities of the protocol, Section VI gives simulation results, and Section VII presents conclusions.

II. COMPARISON TO OTHER PROTOCOLS

Humblet and Soloway [6] also developed a topology broadcast algorithm that builds the spanning trees while the topology information is being broadcast. This algorithm does not use reverse-path forwarding; instead, each node computes its children (instead of its parent) for each source, based on the topology information stored at the node. This algorithm was developed for a fixed topology, but can be implemented in a dynamic topology by restarting the algorithm whenever a topology change is detected; however, this incurs additional communication cost and requires a significant time to recover from each topology change.

In contrast, TBRPF does not need to be restarted when a topology change occurs. Instead, when a node learns of a topology change that results in a new min-hop path to some source, the node immediately selects a new parent for that source, thus recovering quickly from topology changes with minimal communication cost. In addition, the algorithm of [6] for computing children is more complex than the algorithm (breadth-first search or Dijkstra's shortest-path algorithm) used by TBRPF to compute parents.

A reliable version of ERPF was developed in [11] for the special case of a fixed topology. This algorithm, called the Reliable Routing-Broadcast (RRB) algorithm, combines ERPF with the routing protocol of Merlin and Segall [8]. The important property of the Merlin-Segall algorithm that is used to achieve reliability is that it maintains, at all times, a directed spanning tree rooted at the destination. RRB will not operate in networks with link failures, since it requires messages to be sent to and received from the old parent.

The Distance-Vector Multicast-Routing Protocol (DVMRP) [13] is a multicast protocol based on ERPF, and assumes the underlying routing algorithm is a distance-vector algorithm. It is not reliable in networks with a dynamic topology. We note that any unreliable multicast or broadcast protocol can be made reliable by using end-to-end (transport layer) retransmissions. However, in a highly dynamic network, such an approach would result in large delays and increased communica-

tion cost due to the retransmitted messages.

A reliable broadcast protocol suitable for networks with a dynamic topology was developed in [1]. The basic broadcast protocol (BBP) presented in that paper is also based on reverse-path forwarding. BBP can be used with an underlying routing algorithm that does not depend on the broadcast capability, but this raises the question of what routing algorithm to use. A distance-vector algorithm can be used, as in DVMRP, but this incurs additional communication cost for passing the distance information. In [1], a method is presented whereby the broadcast *by itself* constructs the routing structure rather than receiving the routing structure as an external input. This is done by selecting the parent to be the neighbor that "knows" more than other neighbors. However, this method requires additional control messages. In addition, the resulting routing structure can change very quickly, requiring a damping mechanism that is also presented in the cited paper.

By using the topology information received along the broadcast trees to compute the min-hop paths that form the trees, TBRPF automatically obtains a slowly-changing broadcast tree without requiring a damping mechanism, resulting in a simpler protocol with no additional control messages. We note that the method of [1] is for the broadcast of general messages, not topology updates, which is why the method does not utilize topology information for computing routes. Our paper focuses on topology broadcast, and so it makes sense to utilize the topology information received along the broadcast trees to construct the broadcast trees themselves. Of course, this topology information can also be used to compute paths subject to any QoS objectives or constraints.

Routing algorithms exist that provide each node with only *partial* topology information, including the link-vector algorithm (LVA) [5] and optimized link-state routing (OLSR) [7]. These algorithms provide each node with sufficient information to compute a path to any other node. Providing each node with full topology information (as in TBRPF) has the following advantages: it allows computation of alternate or disjoint paths, thus allowing faster recovery from failures; and it allows the computation of paths subject to any combination of QoS objectives and constraints. Further study is needed to compare TBRPF to partial link-state algorithms with respect to efficiency and other performance measures.

III. DESCRIPTION OF TBRPF

This section describes the operation of the TBRPF protocol. We begin with a description of the model representing the topology of the network.

A. Topology Representation and Network Model

The topology of a communication network is represented by a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges or links. The nodes of the graph consist of switches (or routers) and networks. Each switch and network is assigned a unique identifier (e.g., an IP address). A bidirectional point-to-point link between two switches u and v

is represented by two edges (u, v) and (v, u) . Each link has a positive cost that can vary in time, and the cost of (u, v) may be different from that of (v, u) . Switch u is responsible for reporting the cost and up/down status of each link (u, v) to neighbors.

We assume that at any time a link (u, v) is up (operational) if and only if the reverse link (v, u) is up. If link (u, v) fails or recovers, we assume that each endpoint will detect the failure or recovery in finite time (e.g., using hello packets). We also assume that a link layer protocol is used to ensure that messages sent on a link arrive in finite time after their transmission, with no errors and in the same order as sent.

A broadcast network is a network such that any pair of switches attached to the network can communicate directly (e.g., an ethernet or a wireless local area network). A broadcast network can be represented either by assigning a point-to-point link between each pair of nodes attached to the network, or by assigning a pair of links (u, A) and (A, u) for each switch u attached to network A . In this case, the cost of link (A, u) is zero, and while switch u reports the cost of link (u, A) , the reverse link (A, u) is implied and not reported.

A multihop packet radio network is not considered a broadcast network, since not all pairs of nodes can communicate directly. In this case, the links are modeled as point-to-point for the topology graph, but TBRPF can take advantage of the ability to send to multiple neighbors simultaneously. In our simulations, we consider both networks with *unicast links* (point-to-point or receiver-directed CDMA), in which a node must send a separate copy of each message to each neighbor that it wishes to receive the message; and networks that allow (for each message) the option of sending the message to all neighbors on a *broadcast link*.

B. Operation of TBRPF

We now describe the operation of TBRPF, which is specified in detail by the pseudocode in Section III-C. A topology update reporting the state of the link (u, v) is a tuple (u, v, c, sn) , where c and sn are the cost and the sequence number associated with the link. We say that node u is the *head node* of link (u, v) . As noted before, it is the only node that can report changes in parameters of link (u, v) . Therefore, any link-state update (u, v, c, sn) originates from node u .

The protocol stores the following information at each node i of the network:

1. A *topology table*, denoted TT_i , consisting of all link-states stored at the node. The entry for link (u, v) in this table is denoted $TT_i(u, v)$ and consists of the most recent update (u, v, c, sn) received for this link. The components c and sn of the entry for link (u, v) will be denoted $TT_i(u, v).c$ and $TT_i(u, v).sn$.
2. The list of neighbor nodes, denoted N_i .
3. The following is maintained for each node $src \neq i$:
 - A *parent*, denoted $p_i(src)$, which is the neighbor of i that is the next node on the minimum-hop path from node i to node src , as obtained from TT_i .
 - A list of *children*, denoted $children_i(src)$, and

- The sequence number of the most recent link-state update originating from node src and received by node i , denoted $sn_i(src)$.

The parents $p_i(src)$, for all $i \neq src$, form a minimum-hop spanning tree directed toward src (assuming the protocol has converged). The basic idea of TBRPF is to broadcast topology updates in the reverse direction along this spanning tree, while at the same time modifying this tree based on the topology information received along the tree.

To focus on the main features of TBRPF, we assume in this section that a *time stamp* is used for the sequence numbers, and that sequence numbers are unbounded (so that wraparound does not occur). Each link-state update originating from a given node is assigned the current time (measured in eighths of seconds for example) at the node as its sequence number. Thus, the sequence number field is used for both validating and expiring link-state updates. A practical implementation of TBRPF would use bounded sequence numbers combined with periodic updates to avoid wraparound problems.

A node selects a neighbor as the parent for source src by sending a NEW PARENT(src, sn) message containing the identity of node src and the sequence number $sn = sn_i(src)$. A node cancels an existing parent by sending a CANCEL PARENT(src) message containing the identity of the source. Consequently, the set of children at a node with respect to node src is the set of neighbors from which it has received a new parent message pertaining to node src .

The broadcast of topology information is achieved in the following manner. Any link-state update originating from node src is accepted by node i if

- it is received from the neighbor node $p_i(src)$, and
- it has a larger sequence number than the corresponding link-state entry in the topology table at node i .

If accepted, the link-state update is entered into the topology table of node i , and then forwarded to every node in $children_i(src)$ (see the procedures Update_Topology_Table and Process_Update in Section III-C).

Whenever its topology table changes, a node recomputes its parent with respect to every source node (see the procedure Update_Parents). This happens when the node receives a topology update, detects the existence of a new neighbor or the loss of connectivity to an existing neighbor, or expires a link-state update in its topology table. Then, if the node detects that its parent for node src has changed, it sends the following messages. It sends the message CANCEL PARENT(src) to the current (old) parent if it exists. It also sends the message NEW PARENT(src, sn) to the newly computed parent if it exists, where $sn = sn_i(src)$. This is indicative of the “position” up to which node i has received updates from the old parent, and after which the new parent should commence. Upon receiving the NEW PARENT(src, sn) message, the newly computed parent responds with a topology message consisting of all the link states originating from node src in its topology table which have sequence number greater than sn (see the procedure Process_New_Parent).

When a node i detects the existence of a new neighbor

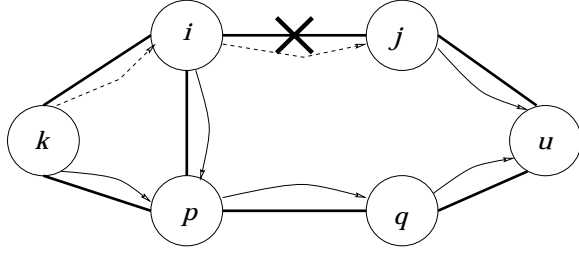


Fig. 1. The figure shows a network topology, where the thick lines represent the bidirectional links between nodes. The direction of the parent nodes for source u is shown. When link (i, j) fails, nodes i and k select node p as the new parent for source node u .

nbr , it executes $\text{Link_Up}(i, nbr)$ to process this newly established link. The link cost and sequence number fields for this link in the topology table at node i are updated. Then, the corresponding link-state message is sent to all neighbors in $\text{children}_i(i)$. As noted above, node i also recomputes its parent node $p_i(src)$, for every node src , in response to this topological change. In a similar manner, when node i detects the loss of connectivity to an existing neighbor nbr , it executes $\text{Link_Down}(i, nbr)$. $\text{Link_Change}(i, nbr)$ is likewise executed at node i in response to a change in the cost to an existing neighbor node nbr . However, this procedure does not recompute parents.

We assume that, initially, a node has no links to neighbors, and that its topology table is empty. Also initially, at each node i , $p_i(src) = \text{NULL}$ (i.e., not defined), $\text{children}_i(src) = \emptyset$, and $sn_i(src) = 0$ for every node src . Every node then executes Link_Up to process each link established with a neighbor.

We now present a simple example to illustrate the operation of TBRPF under topological changes. In Figure 1, we focus on source node u and show the direction of the parent nodes for this source. For example, $p_q(u) = u$, and $p_k(u) = i$. Suppose that $sn_i(u) = 5$ and $sn_k(u) = 5$. When link (i, j) fails, node i computes node p as the new parent, and sends a $\text{NEW_PARENT}(u, 5)$ message to node p . Node i also reports the failure of link (i, j) to node k , since $k \in \text{children}_i(i)$. Similarly, when node k learns of the failed link (i, j) , it sends a $\text{NEW_PARENT}(u, 5)$ message to node p and a $\text{CANCEL_PARENT}(u)$ message to node i . In response, node p sends all link states in its topology table originating from node u and having a sequence number greater than 5 to both these nodes. All subsequent topology messages originating at node u and received at node p will be forwarded to nodes in $\text{children}_p(u)$ (which include nodes i and k).

C. Pseudocode for TBRPF

```

Process_Update( $i, nbr, in\_message$ )
{
  Update_Topology_Table( $i, nbr, in\_message, update\_list$ );
  Update_Parents( $i$ );
  for each (node  $k \in N_i$ )  $out\_message(k) \leftarrow \emptyset$ ;
  for each (node  $src \in TT_i$  s.t.  $src \neq i$ ) {
    Let  $update\_list(src) \leftarrow \{(k, l, c, sn) \in update\_list \text{ s.t. } k = src\}$ ;
    for each (node  $k \in \text{children}_i(src)$ )

```

```

     $out\_message(k) \leftarrow out\_message(k) \cup update\_list(src)$ ;
  }
  for each (node  $k \in N_i$  such that  $out\_message(k) \neq \emptyset$ )
    Send the message  $out\_message(k)$  to node  $k$ ;
}

```

```

Update_Topology_Table( $i, nbr, in\_message, update\_list$ )
{
   $update\_list \leftarrow \emptyset$ ;
  for each ( $(u, v, c, sn) \in in\_message$ ) {
    if ( $p_i(u) \neq nbr$ ) ;
    else { (Process only updates received from the parent  $p_i(u)$ .)
      if ( $(u, v) \in TT_i$ ) {
        if ( $sn > TT_i(u, v).sn$ ) {
           $update\_list \leftarrow update\_list \cup \{(u, v, c, sn)\}$ ;
           $TT_i(u, v).sn \leftarrow sn$ ;
           $TT_i(u, v).c \leftarrow c$ ;
          if ( $sn > sn_i(u)$ )  $sn_i(u) \leftarrow sn$ ;
        }
      }
      else { (Link  $(u, v)$  not present in topology table.)
        Add  $(u, v, c, sn)$  to  $TT_i$ ;
         $update\_list \leftarrow update\_list \cup \{(u, v, c, sn)\}$ ;
        if ( $sn > sn_i(u)$ )  $sn_i(u) \leftarrow sn$ ;
      }
    }
  }
}

```

```

Link_Change( $i, j$ )
{
  if ( $(|TT_i(i, j).c - cost(i, j)| / TT_i(i, j).c > \epsilon)$  {
    (Update link cost only if percent change is greater than  $\epsilon$ )
     $TT_i(i, j).c \leftarrow cost(i, j)$ ;
     $TT_i(i, j).sn \leftarrow ts$ ;
     $out\_message \leftarrow (i, j, TT_i(i, j).c, TT_i(i, j).sn)$ ;
    for each (node  $k \in N_i$  s.t.  $k \in \text{children}_i(i)$ )
      Send  $out\_message$  to node  $k$ ;
  }
}

```

```

Link_Down( $i, j$ )
{
   $N_i \leftarrow N_i - \{j\}$ ;
   $TT_i(i, j).c \leftarrow \infty$ ;
   $TT_i(i, j).sn \leftarrow ts$ ;
  Update_Parents( $i$ );
  for each (node  $src \in TT_i$ )  $\text{children}_i(src) \leftarrow \text{children}_i(src) - \{j\}$ ;
   $out\_message \leftarrow (i, j, \infty, TT_i(i, j).sn)$ ;
  for each (node  $k \in N_i$  s.t.  $k \in \text{children}_i(i)$ )
    Send  $out\_message$  to node  $k$ ;
}

```

```

Link_Up( $i, j$ )
{
   $N_i \leftarrow N_i \cup \{j\}$ ;
   $TT_i(i, j).c \leftarrow cost(i, j)$ ;
   $TT_i(i, j).sn \leftarrow ts$ ;
  Update_Parents( $i$ );
   $out\_message \leftarrow (i, j, TT_i(i, j).c, TT_i(i, j).sn)$ ;
}

```

```

for each (node  $k \in N_i$  s.t.  $k \in \text{children}_i(i)$ )
  Send  $\text{out\_message}$  to node  $k$ ;
}

Compute_New_Parents( $i$ )
{
  for each (node  $\text{src} \neq i$ )  $\text{new\_p}_i(\text{src}) \leftarrow \text{NULL}$ ;
  Dijkstra( $i$ ); (Computes the min-hop path tree rooted at node  $i$ .)
  for each (node  $\text{src} \neq i$ )
     $\text{new\_p}_i(\text{src}) \leftarrow$  neighbor of node  $i$  on the min-hop path from  $i$  to  $\text{src}$ ;
}

```

```

Process_New_Parent( $i, \text{nbr}, \text{src\_list}, \text{sn\_list}$ )
{
   $\text{update\_list} \leftarrow \emptyset$ ;
  for each (node  $\text{src} \in \text{src\_list}$ ) {
     $\text{children}_i(\text{src}) \leftarrow \text{children}_i(\text{src}) \cup \{\text{nbr}\}$ ;
     $\text{new\_updates} \leftarrow \{(k, l, c, \text{sn}) \in TT_i \text{ such that } k = \text{src} \text{ and } \text{sn} > \text{sn\_list}.\text{src}\}$ ;
    ( $\text{sn\_list}.\text{src}$  refers to the sequence number corresponding to node  $\text{src}$ )
     $\text{update\_list} \leftarrow \text{new\_updates} \cup \text{update\_list}$ ;
  }
  Send the message  $\text{update\_list}$  to  $\text{nbr}$ .
}

```

```

Update_Parents( $i$ )
{
  Compute_New_Parents( $i$ );
  for each (node  $k \in N_i$ )
     $\text{cancel\_src\_list}(k) \leftarrow \emptyset$ ;  $\text{src\_list}(k) \leftarrow \emptyset$ ;  $\text{sn\_list}(k) \leftarrow \emptyset$ ;
  for each (node  $\text{src} \in TT_i$  s.t.  $\text{src} \neq i$ ) {
    if ( $\text{new\_p}_i(\text{src}) \neq p_i(\text{src})$ ) {
      if ( $p_i(\text{src}) \neq \text{NULL}$ ) {
        Let  $k \leftarrow p_i(\text{src})$  be the neighbor of  $i$ .
         $\text{cancel\_src\_list}(k) \leftarrow \text{cancel\_src\_list}(k) \cup \{\text{src}\}$ ;
      }
      if ( $\text{new\_p}_i(\text{src}) \neq \text{NULL}$ ) {
        Let  $k \leftarrow \text{new\_p}_i(\text{src})$  be the neighbor of  $i$ .
         $\text{src\_list}(k) \leftarrow \text{src\_list}(k) \cup \{\text{src}\}$ ;
         $\text{sn\_list}(k) \leftarrow \text{sn\_list}(k) \cup \{\text{sn}_i(\text{src})\}$ ;
      }
       $p_i(\text{src}) \leftarrow \text{new\_p}_i(\text{src})$ ;
    }
  }
  for each (node  $k \in N_i$ ) {
    if ( $\text{src\_list}(k) \neq \emptyset$ )
      Send the message NEW PARENT( $\text{src\_list}(k), \text{sn\_list}(k)$ ) to node  $k$ ;
    if ( $\text{cancel\_src\_list}(k) \neq \emptyset$ )
      Send the message CANCEL PARENT( $\text{cancel\_src\_list}(k)$ ) to node  $k$ ;
  }
}

```

```

Process_Cancel_Parent( $i, \text{nbr}, \text{src\_list}$ )
{
  for each ( $\text{src} \in \text{src\_list}$ )  $\text{children}_i(\text{src}) \leftarrow \text{children}_i(\text{src}) - \{\text{nbr}\}$ ;
}

```

```

Send_Periodic_Updates( $i$ )
{

```

```

   $\text{out\_message} \leftarrow \emptyset$ ;
  for each ( $v \in N_i$  such that  $TT_i(i, v).c \neq \infty$ ) {
     $TT_i(i, v).\text{sn} \leftarrow \text{ts}$ ;
     $\text{out\_message} \leftarrow (i, v, TT_i(i, v).c, TT_i(i, v).\text{sn}) \cup \text{out\_message}$ ;
  }
  for each (node  $k \in N_i$  s.t.  $k \in \text{children}_i(i)$ )
    Send  $\text{out\_message}$  to node  $k$ ;
}

```

IV. CORRECTNESS OF TBRPF

At any given time, the correct topology (as seen by an omniscient observer) is denoted by T^* . We borrow notation from [2]. We say that link (u, v) is connected with node i at a given time if there is a path from i to u consisting of links that are up (according to T^*). We say that node i knows the correct topology at a given time if TT_i agrees with T^* on the states of all links connected with i . Initially, at time 0, we assume that each node knows only the states of its own outgoing links, and that it generates an update for these links within finite time. We also assume that $\text{sn}_i(v) = 0$ for each node i and each source $v \neq i$. We assume there is a time t_0 after which no link changes state and each node is aware of the states of its outgoing links. We will show the following:

Theorem 1. TBRPF is correct in the sense that, under the preceding assumptions, there is a time $t_f \geq t_0$ such that each node knows the correct topology for all $t \geq t_f$.

The proof of Theorem 1 will make use of the following lemma, which states that each node i has the most recently generated update for each link (u, v) that has a sequence number less than or equal to $\text{sn}_i(v)$. We use the notation $\text{sn}_i(v)[t]$ and $TT_i[t]$ to denote the values of $\text{sn}_i(v)$ and TT_i at time t .

Lemma 1. At any time t and for any nodes i and u , $i \neq u$, if $\text{lsu} = (u, v, c, \text{sn})$ is the most recent update with $\text{sn} \leq \text{sn}_i(u)[t]$ that node u generated for link (u, v) , then $\text{lsu} \in TT_i[t]$.

Proof. The proof is by contradiction. Let $t^* > 0$ be the earliest time for which the lemma does not hold. (The lemma clearly holds for $t = 0$ since no updates have yet been received by any node and therefore $\text{sn}_i(u) = 0$ for all i and u .) Then there exists an update $\text{lsu} = (u, v, c, \text{sn})$ that is the most recent update with $\text{sn} \leq \text{sn}_i(u)[t^*]$ for link (u, v) , and is not in $TT_i[t^*]$. By definition, $\text{lsu}^* = (u, v^*, c^*, \text{sn}_i(u)[t^*])$ is in $TT_i[t^*]$, and so we must have $\text{sn} < \text{sn}_i(u)[t^*]$. Since $i \neq u$, let j be the parent node from which node i received the update lsu^* , and let $t' < t^*$ be the most recent time prior to t^* that node i sent a **NEW PARENT**($u, \text{sn}_i(u)[t']$) message to node j . We consider two cases: $\text{sn} \leq \text{sn}_i(u)[t']$ and $\text{sn} > \text{sn}_i(u)[t']$. First suppose $\text{sn} \leq \text{sn}_i(u)[t']$. Then lsu is the most recent update with $\text{sn} \leq \text{sn}_i(u)[t']$ that node u generated for link (u, v) . Therefore, since t^* is the earliest time for which the lemma does not hold, lsu must be in TT_i at time t' . Now since lsu is the most recent update for link (u, v) with $\text{sn} \leq \text{sn}_i(u)[t^*]$, lsu is still in TT_i at time t^* , contradicting our assumption that lsu is not in $TT_i[t^*]$.

We can therefore assume that $\text{sn} > \text{sn}_i(u)[t']$, where

$sn_i(u)[t']$ is the sequence number in the new parent message sent by node i to node j at time t' . Let $t^+ < t^*$ be the time at which node j sent the update lsu^* to node i . Then during the interval $[t', t^+]$, node j sent to node i all updates generated by source u with sequence numbers greater than $sn_i(u)[t']$ that were either contained in TT_j when the new parent message was processed by j or were later added to TT_j . Since updates sent on a link arrive at the receiver in the order sent, it follows that, at time t^* , node i had received from j all updates in $TT_j[t^+]$ generated by source u with sequence numbers greater than $sn_i(u)[t']$, the last such update being lsu^* . Since $sn_i(u)[t^*]$ is the largest sequence number of any such update received by node i , it follows that $sn_j(u)[t^+] = sn_i(u)[t^*]$. Now since the lemma holds for time $t^+ < t^*$, lsu must be in $TT_j[t^+]$, and therefore must have been received by node i by time t^* , contradicting our assumption that lsu is not in $TT_i[t^*]$. This proves the lemma. ■

Proof of Theorem 1. We will show by induction that, for each integer $m \geq 0$, there is a time $t_m \geq t_0$ after which, for each node i , TT_i agrees with T^* for all links originating from nodes that are m or fewer hops from i . This inductive hypothesis is clearly true for $m = 0$, since at time t_0 each node i knows the correct link state of its outgoing links and stores them in TT_i .

Now suppose there is a time $t_k \geq t_0$ such that, for all $t \geq t_k$ and for each node i , TT_i agrees with T^* for all links originating from nodes that are k or fewer hops from i . To complete the proof, we need to show that there is a time $t_{k+1} \geq t_k$ such that, for all $t \geq t_{k+1}$ and for each node i , TT_i agrees with T^* for all links originating from nodes that are $k + 1$ or fewer hops from i .

Now consider nodes i and v such that node v is $k + 1$ hops from node i . By the inductive hypothesis, node i has sufficient information by time t_k to compute the correct minimum-hop paths to all nodes that are up to $k + 1$ hops away, and in particular to node v . Node i therefore has sufficient information to compute the final parent node for source v . Letting j denote this parent node, node v is k hops from node j . Therefore, by the inductive hypothesis, for all $t \geq t_k$, TT_j agrees with T^* for all links originating from node v . In particular, $sn_j(v)[t] = sn_v(v)[t]$ for all $t \geq t_k$, where $sn_v(v)[t]$ is the sequence number of the final update generated by node v .

Let sn be the sequence number of the last new parent message that node i sent to node j for source v . (We note that this new parent message may have been sent before time t_k or shortly after time t_k .) In response to this new parent message, node j sends to node i all updates in TT_j that were generated by source v and have sequence numbers greater than sn . Subsequently, any updates generated by node v and received by node j that resulted in a change to TT_j are forwarded to node i . Therefore, since we already showed that $sn_j(v)[t] = sn_v(v)[t]$ for all $t \geq t_k$, there exists a time $t^* \geq t_k$ such that $sn_i(v)[t] = sn_v(v)[t]$ for all $t \geq t^*$, where $sn_v(v)[t]$ is the sequence number of the last update generated by node

v . Therefore, by Lemma 1, for all $t \geq t^*$, TT_i contains the most recent update that node v generated for each link originating from node v . That is, TT_i agrees with T^* for all links originating from node v . Since the above argument holds for all pairs i and v that are $k + 1$ hops apart, and letting t_{k+1} be the maximum t^* over all such pairs, the inductive hypothesis holds for $m = k + 1$, completing the proof of the theorem. ■

V. COMPLEXITIES OF TBRPF

In this section, we derive the worst-case communication complexity, time complexity, computational complexity, and storage complexity for TBRPF.

A. Communication Complexity

We define a message unit to be the number of bits in a node ID or a sequence number, whichever is larger. We will derive the worst-case number of message units generated in four cases: a link-cost change, a link going down, a link coming up that does not result in the recovery of a network partition, and a link coming up that results in the recovery of a network partition.

A link-cost change (without a change of up/down status) for link (u, v) results in an update sent on each link of the min-hop reverse path tree rooted at node u , resulting in $O(|V|)$ message units. No new parent or cancel parent messages are generated. This compares to $O(|E|)$ for flooding.

Next suppose the link (u, v) and the reverse link (v, u) fail. For each node i , only one of (u, v) and (v, u) can be on the min-hop tree MHT_i computed by node i , implying that node i may need to select a new parent for source u or source v but not for both. Node i will receive the update for each link (u, v) and (v, u) either from the old parent for u or v (which may result in the selection of a new parent for the other source), or from the new parent. Therefore, each node will receive two updates, resulting in a total of $2|V|$ updates or $O(|V|)$ units sent over all links. In addition, since either (u, v) or (v, u) but not both may belong to MHT_i , each node may send a new parent message and a cancel parent message that include up to $|V|$ sources, or a total of $O(|V|^2)$ units for new parent and cancel parent messages. Therefore, the worst-case communication complexity for a link failure is $O(|V|^2)$, which compares to $O(|E|)$ for flooding. However, on average the parent will change for only a small fraction of node pairs (i, u) , and so we expect (and observe in simulations) that the average communication complexity is much smaller than this.

If the links (u, v) and (v, u) come up, and this does not result in the joining of two components of a partitioned network, the worst-case complexity is similarly $O(|V|^2)$. Again, we expect the average complexity to be much smaller than this.

Finally, suppose that the links (u, v) and (v, u) come up, and that this connects two components A and B of the network that were previously disconnected. We first comment that if the links (u, v) and (v, u) were down for only a short time, so that the nodes in A still know the correct states of links originating from nodes in B (and vice versa), then the only updates that

need to be broadcast are the ones for links (u, v) and (v, u) . In the worst-case, the nodes in A do not know the current state of any link originating from a node in B (and vice versa). In this case, the state of every link on each side of the partition must be broadcast to all nodes on the other side along a tree, resulting in $O(|E||V|)$ message units. This compares to $O(|E|^2)$ for flooding, since each link on one side of the partition must be sent over each link on the other side.

B. Time Complexity

We define time complexity to be the maximum time required for the protocol to converge (so that each node knows the correct topology and knows its parents and children for all sources), where we define a unit of time to be the maximum difference between the time a message A arrives at node i and the time a resulting message B arrives at a neighbor of node i .

Before examining the four cases considered for communication complexity, we consider the general case addressed by Theorem 1. That is, we assume there is a time t_0 after which no further link-state changes occur. From the proof of Theorem 1, it is clear that the difference between t_k and t_{k+1} is at most the time required for all nodes to send a number of new parent messages and to receive the update messages that were sent by the new parents in response. Therefore, since k can be at most D , where D is the diameter of the network, the algorithm converges in at most $2D$ time units, or $O(D)$. The time complexity of flooding is D time units.

A link-cost change (without a change of up/down status) for link (u, v) results in an update sent on each link of the minimum hop reverse path tree rooted at node u , requiring at most D time units.

For the case of a link failure, it can be seen from the discussion of communication complexity that the maximum time to converge is at most $D + 2$, where the additional 2 units of time may be required for some nodes to send a new parent message and receive an update in response. For a link recovery that does not result in a partition recovery, the maximum time to converge is similarly $D + 2$.

Finally, suppose that the links (u, v) and (v, u) come up, and that this results in the joining of two network components A and B that were previously disconnected. In the worst case, the nodes in A have no knowledge of the states of links in B . In this case, node u (in A) would first send a new parent message to v for source v and in response would receive an update from v for links originating from v ; node u would then send a new parent message to v for all sources in B that are 2 hops away and in response would receive an update from v for links originating from these nodes; etc. This process of sending a new parent message and receiving a response will be repeated at most D times, resulting in a maximum time to converge of $2D$ time units, the same as for the general case.

C. Computational and Storage Complexities

For each update received, a node must compute a minimum-hop spanning tree for the topology stored at the node, which

requires $O(|E|)$ time using breadth-first search (BFS). Parents can be computed during the execution of BFS with no additional time requirement. None of the procedures of TBRPF require more than $O(|E|)$ time, which is therefore the computational complexity per received update. TBRPF is a topology broadcast algorithm and does not compute paths based on link costs. An additional path selection algorithm would be needed to compute paths based on the link costs stored in TT_i . For example, Dijkstra's shortest-path algorithm, which runs in $O(|E|\log|V|)$ time, could be used to compute shortest paths.

The main storage requirements are for the topology table, parent assignment, and children assignment. The topology table at each node contains $|E|$ entries, one for each link. The parent assignment (one per source) requires $O(|V|)$ storage, and the children assignment requires up to $O(|V||N_i|) \leq O(|V|^2)$ storage, where $|N_i|$ is the number of neighbors of node i . To reduce the storage requirement, the children assignment can be represented by a vector of $|V|$ bits for each neighbor (as in [3]) or by a vector of $|N_i|$ bits for each source node.

VI. PERFORMANCE EVALUATION

Simulation experiments were conducted to compare TBRPF to two versions of flooding with respect to communication cost (amount of update and control traffic) and convergence time. Both versions of flooding have the property that a new update (with a higher sequence number) received from a neighbor is forwarded to all other neighbors. The two versions differ as follows when a link comes up. In version 1 (Flooding1), which is similar to the version of [12], each node of the link sends to the other node its entire topology table, consisting of the updates (u, v, c, sn) . In version 2 (Flooding2), each node of the link first sends to the other node a message containing the pairs $(u, sn(u))$ for all u , where $sn(u)$ is the sequence number of the most recent update generated by u that has been received; each node then responds by sending the other node only updates (u, v, c, sn) such that sn is larger than the sequence number $sn(u)$ received from the other node (i.e., updates that are more recent than those known by the other node).

A. Communication Cost

The following network model was used to measure communication cost. Twenty nodes are initially placed randomly in a unit square. Each node moves in a random direction with maximum distance DELTA every 0.1 s, where DELTA is either 0.001 (slow movement) or 0.004 (fast movement). A link can exist between two nodes only if they are within distance RADIUS of each other, where RADIUS is either 0.3 or 0.5. The average number of links for RADIUS equal to 0.3 and 0.5 were 126 and 240, respectively. Network partitioning occurred frequently with RADIUS equal to 0.3. The simulation time of each experiment was 3600 s. Periodic updates were not used.

The size of each link-state update (u, v, c, sn) is assumed to be the same (40 bits) for all protocols: 8 bits for each of u, v , and c , and 16 bits for sn . (These numbers were cho-

sen to be small for wireless networks with very limited bandwidth.) Correspondingly, the size of a NEW PARENT(u, sn) and CANCEL PARENT(u) message is assumed to be 24 bits and 8 bits, respectively.

The cost $c(u, v)$ of link (u, v) is assumed to be proportional to the distance between u and v , and to be equal to infinity if this distance is greater than RADIUS. A link-cost threshold of 20% was used, so that a link-cost was updated only if it changed by at least 20% from its last stored value.

To model the random link delays of a typical asynchronous network, a message sent by a node i at time t is assumed to arrive at the neighbor of i at time $t + \tau$, where τ is a random number between 0 and 0.1 s. The average link delay is therefore 0.05 s. The time required to process a message is assumed to be negligible.

We considered two transmission models. In the first model, we assume that the network has only unicast links (point-to-point links or receiver-directed CDMA), so that a node must send a separate copy of each message to each neighbor that it wishes to receive the message. For this model, the average amount of update/control traffic per link measured in bits/sec is given in Table I.

In the second model, we assume that a node has the option of sending each message to all neighbors simultaneously on a broadcast link. This option improves efficiency if the same update is to be sent to several neighbors. For this model, two different transmission rules were used, each with a corresponding measure. The first rule uses the broadcast option if the update has at least two intended receivers, and otherwise uses the unicast (receiver-directed) option. For this rule, Table II gives the average amount of update/control traffic per node for broadcast transmissions, unicast transmissions, and the sum of these two, measured in bits/sec.

The sum of broadcast and unicast transmissions, although a useful measure, does not represent the fact that broadcast transmissions are more costly than unicast transmissions, in the sense that several neighbors must spend time receiving the transmission (during which time they could be communicating with other nodes). Therefore, the second measure for the second model is the average amount of update/control traffic per node that is *sent or received*, which is also given in Table II. For a given message, let k be the number of intended receivers (e.g., the number of children in TBRPF), n the total number of neighbors, and $|M|$ the size of the message in bits. Then unicasting the message separately to the k intended receivers would result in $2k|M|$ bits sent or received (by any node), and broadcasting the message simultaneously to all neighbors would result in $(n + 1)|M|$ bits sent or received. Therefore, to minimize this measure, the second rule uses the broadcast option if $n + 1 \leq 2k$, and otherwise uses the unicast option.

For the unicast model, we see that TBRPF achieves a 91% to 98% reduction in communication cost compared to Flooding1, and a 90% to 97% reduction compared to Flooding2. This dramatic improvement can be explained as follows. First, flooding sends each update on roughly every link of the network, while TBRPF sends each update only on a spanning tree. For RA-

TABLE I
AMOUNT OF UPDATE/CONTROL TRAFFIC (BITS/S/LINK) GENERATED IN THE UNICAST MODEL

RADIUS	DELTA	Flooding1	Flooding2	TBRPF
0.3	.001	688.7	611.5	57.0
0.3	.004	4969.9	4393.1	437.3
0.5	.001	1470.8	1367.9	35.2
0.5	.004	6301.2	5896.3	218.6

TABLE II
AMOUNT OF UPDATE/CONTROL TRAFFIC (BITS/S/NODE) GENERATED IN THE BROADCAST MODEL

RADIUS DELTA	Measure	Flooding1	Flooding2	TBRPF
0.3 0.001	Broadcast	692.8	692.8	51.9
	Unicast	579.4	96.9	165.3
	Ucast+Bcast	1272.2	789.7	217.2
	Sent+Rcvd	6521.0	5556.0	637.7
0.3 0.004	Broadcast	4998.2	4998.2	453.6
	Unicast	4324.3	661.4	1126.0
	Ucast+Bcast	9322.6	5659.7	1579.6
	Sent+Rcvd	47388.5	40062.8	4994.1
0.5 0.001	Broadcast	1552.8	1552.8	40.8
	Unicast	1367.5	128.1	177.9
	Ucast+Bcast	2920.3	1680.9	218.7
	Sent+Rcvd	23093.0	20614.3	707.9
0.5 0.004	Broadcast	6709.9	6709.9	303.4
	Unicast	5308.5	466.7	852.4
	Ucast+Bcast	12018.4	7176.7	1155.8
	Sent+Rcvd	97764.5	88081.0	4268.2

DIUS = 0.5, the average number of links in the 20-node network is 240, which compares to 19 links in the spanning tree. This accounts for 92% of the reduction. In addition, when a link comes up, TBRPF is more efficient than both versions of flooding, since in TBRPF each node i adjacent to the link selects the other node as parent for only a *subset* of the sources and obtains from the new parent only link-state updates that are more recent than those stored in node i 's topology table.

For the broadcast model, both versions of flooding generate the same amount of broadcast traffic, but Flooding2 generates much less unicast traffic than Flooding1. (This is expected, since the topology information exchanged when a link comes up is unicast traffic.) As a result, Flooding2 performs about 40% better than Flooding1 with respect to the first measure (the sum of unicast and broadcast traffic). However, since the second measure (which includes both sent and received traffic) weighs broadcast traffic more than unicast traffic, Flooding2 performs only 10% to 15% better than Flooding1 with respect to this measure.

TABLE III
AVERAGE CONVERGENCE TIMES FOR TBRPF AND FLOODING

RADIUS	Event	Flooding	TBRPF
0.3	Link Fail	0.276	0.108
	Link Recover	0.332	0.126
	Node Fail	0.317	0.291
	Node Recover	0.399	0.409
0.5	Link Fail	0.167	0.116
	Link Recover	0.220	0.122
	Node Fail	0.206	0.282
	Node Recover	0.275	0.276

The main advantage of TBRPF over flooding in the broadcast model is that in TBRPF, a node that is a leaf of the broadcast tree rooted at the source of a received update (i.e., that has no children) does not forward that update; and a node that has only one (or a few) children can unicast the update (to minimize the second measure). The average number of leaves of the min-hop trees for RADIUS equal to 0.3 and 0.5 were 13.6 and 16.5, respectively. We would therefore expect the advantage of TBRPF to be greater for the second measure (since it gives more weight to broadcast traffic), and the results confirm this. For the first measure, TBRPF performs 85% to 93% better than Flooding1 and 72% to 87% better than Flooding2. For the second measure, TBRPF performs 89% to 97% better than Flooding1 and 88% to 97% better than Flooding2.

B. Convergence Time

Simulations were conducted to measure the average time to converge following four events: link failure, link recovery, node failure, and node recovery. (Because the protocol may never converge in a network with constantly moving nodes, we could not use the same scenario as for measuring communication cost.) In these experiments, nodes are initially placed randomly in a unit square, but do not move. For one set of experiments, each link was made to fail and then recover, one at a time. For a second set of experiments, each node was made to fail and then recover. After each topology change, the time for the protocol to converge was measured as the time between the first message sent and the last message received. Each experiment was repeated for 20 random topologies to obtain results that do not depend on a single topology.

The results are given in Table III. Version 1 of flooding was used. Since the average link delay is 0.05 s, the convergence time can be expressed in number of hops by dividing by this number. TBRPF had significantly better recovery time than flooding for link failure and link recovery and about the same as flooding for node recovery. The only case where flooding performed significantly better than TBRPF is for node failure with RADIUS = 0.5, where TBRPF required 37% more time on average than flooding.

VII. CONCLUSIONS

We presented a new protocol (TBRPF) for the efficient, reliable broadcast of topology and link-state information in a multihop communication network with a dynamic topology. We also proved its correctness, derived its worst-case complexities, and presented simulation results that compare TBRPF to versions of flooding with respect to average communication cost and convergence times.

In simulations, TBRPF achieved a dramatic reduction in average communication cost compared to both versions of flooding, especially in the unicast (point-to-point or receiver-directed) model and in the broadcast model when both sent and received traffic were counted. In the broadcast model, one advantage of TBRPF is that the leaves of a broadcast tree need not forward updates; and simulations showed that most nodes in a typical multihop radio network are leaves of a given min-hop spanning tree.

One potential drawback of TBRPF is that its worst-case convergence time is $2D$, twice that of flooding. However, in simulations, there was only one event for which the average convergence time of TBRPF was significantly greater than that of flooding.

We discussed the advantages of algorithms (including TBRPF) that provide each node with complete topology information, compared to algorithms (including LVA and OLSR) that provide each node with only part of the network topology. Further study is needed to compare TBRPF to partial-topology algorithms with respect to communication cost and other performance measures.

REFERENCES

- [1] B. Awerbuch and S. Even, "Reliable Broadcast Protocols in Unreliable Networks," *Networks*, Vol. 16, 1986, pp. 381-396.
- [2] D. Bertsekas and R. Gallager, *Data Networks*, 2nd edition, Englewood-Cliffs, NJ:Prentice-Hall:1992.
- [3] Y. Dalal, and R. Metcalfe, "Reverse Path Forwarding of Broadcast Packets," *Communications of the ACM*, Vol. 21, No. 12, Dec. 1978, pp. 1040-1048.
- [4] S. Deering, "Multicast Routing in Internetworks and Extended LANs," in *ACM SIGCOMM*, Aug 1988, pp. 55-64.
- [5] J.J. Garcia-Luna-Aceves and J. Behrens, "Distributed Scalable Routing Based on Vectors of Link States," *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 8, 1995.
- [6] P. Humblet and S. Soloway, "Topology Broadcast Algorithms," *Computer Networks and ISDN Systems*, North-Holland, 16 (1988/89), pp. 179-186.
- [7] P. Jacquet, P. Muhlethaler, and A. Qayyum, "Optimized Link State Routing Protocol," work in progress, INRIA Rocquencourt, France, November 1998.
- [8] P. Merlin and A. Segall, "A failsafe distributed routing protocol," *IEEE Trans. Comm.* Vol. COM-27, pp. 1280-1287, Sept. 1987.
- [9] J. Moy, "OSPF Version 2," RFC 1583, Network Working Group, March 1994.
- [10] R. Perlman, "Fault-Tolerant Broadcast of Routing Information," in *Computer Networks*, North-Holland, Vol. 7, 1983, pp. 395-405.
- [11] A. Segall and B. Awerbuch, "A Reliable Broadcast Protocol," *IEEE Trans. of Comm.*, Vol. COM-31, No. 7, July 1983, pp. 896-901.
- [12] U. Vishkin, "A distributed orientation algorithm," *IEEE Trans. Info. Theory*, 1983.
- [13] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," RFC 1075, BBN STC, Nov. 1988.