INTERNET-DRAFT

Bhargav Bellur
Richard G. Ogier
Fred L. Templin
SRI International
11 July 2000

Topology Broadcast based on Reverse-Path Forwarding (TBRPF)

<draft-ogier-manet-tbrpf-00.txt>

Status of This Memo

Abstract

Topology Broadcast based on Reverse-Path Forwarding (TBRPF) is a link-state routing protocol designed for use in a mobile ad-hoc network (MANET) or an internet. TBRPF provides each node with the state of each link in the network, but does so much more efficiently than flooding (e.g., OSPF). TBRPF uses the concept of reverse-path forwarding to efficiently and reliably broadcast each link-state update along the minimum-hop-path tree rooted at the source of the update. The broadcast trees (one tree per source) are updated dynamically using the topology information that is received along the trees themselves, thus requiring very little additional overhead for maintaining the trees. This document describes TBRPF and discusses details for the implementation of TBRPF in IPv4-based MANETs.

Contents

1. Introduction

This document describes the Topology Broadcast based on Reverse-Path Forwarding (TBRPF) protocol [1], developed in the DARPA Small Unit Operations (SUO) program. TBRPF addresses the problem of efficient routing in a communication network, where by efficient we mean that the amount of update and control traffic required to maintain shortest (or nearly shortest) paths to all destinations is minimized. This problem is important if the network incurs frequent topology and link-cost changes, or must use links of limited bandwidth, or if the network is very large (for scalable routing). In particular, this problem is important for mobile ad-hoc networks (MANETs).

Routing protocols can be classified as proactive protocols, in which each node maintains paths at all times to all possible destinations, and reactive (or on-demand) protocols, in which paths are found (using route discovery) only when they are required. TBRPF is a proactive, link-state protocol, as are flooding (e.g., OSPF) and STAR (Source Tree Adaptive Routing) [4]. Examples of reactive protocols are DSR [6] and AODV [8].

Reactive protocols tend to be more efficient when route requests are infrequent, while proactive protocols tend to be more efficient when route requests are frequent (since route discovery typically involves flooding). In addition, the delay required by reactive protocols for route discovery may be too large for some applications. Therefore, the best solution may be a hybrid routing solution that combines proactive and reactive techniques, or a protocol that changes dynamically between proactive and reactive methods, based on network measurements. TBRPF can be used as the proactive part of such a hybrid solution.

Routing protocols can also be classified according to whether they find optimal (shortest) routes or suboptimal routes. By not requiring routes to be optimal, it is possible to reduce the amount of control traffic (including routing updates) necessary to maintain the routes. However, optimal routes are desirable because they minimize delay and the amount of resources (e.g., bandwidth and power) consumed. TBRPF computes optimal routes based on the advertised link states; however, the advertised link states themselves may be approximate in order to reduce the frequency at which each link is updated.

TBRPF is a full-topology link-state protocol: each node is provided with the state of each link in the network (or within a cluster if hierarchical routing is used). Flooding is another full-topology link-state protocol, but is very inefficient due to the following redundancies: (1) updates are sent over multiple paths to each node; and (2) every node forwards every update to all neighbors, even if

none or only one neighbor needs to receive it. TBRPF removes these redundancies, thus providing the same routing information as flooding with much greater efficiency. In fact, in simulation experiments [7], TBRPF generated up to 85% less update/control traffic than an efficient version of flooding.

Full-topology link-state protocols have the following advantages over partial-topology protocols: (1) alternate paths and disjoint paths are immediately available, allowing faster recovery from failures and topology changes; and (2) paths can be computed subject to any combination of quality-of-service (QoS) constraints and objectives. Examples of partial-topology link-state protocols include STAR and OLSR. These protocols provide each node with sufficient topology information to compute at least one path to each destination.

TBRPF uses the concept of reverse-path forwarding to broadcast each link-state update in the reverse direction along the spanning tree formed by the minimum-hop paths from all nodes to the source of the update. That is, each link-state update is broadcast along the minimum-hop-path tree rooted at the source of the update. The broadcast trees (one tree per source) are updated dynamically using the topology information that is received along the trees themselves, thus requiring very little additional overhead for maintaining the trees. (The fact that this is possible in a dynamic network is a new result.) Minimum-hop-path trees are used because they change less frequently than shortest-path trees based on a metric such as delay. Based on the information received along the broadcast trees, each node computes its parent and children for the broadcast tree rooted at each source u. Each node forwards updates originating from source u to its children on the tree rooted at source u. TBRPF achieves reliability despite topology changes, using sequence numbers. The correctness of TBRPF has been proven [1].

For point-to-point links (or receiver-directed transmissions), each link-state update is sent on only |V|-1 tree links in TBRPF versus approximately |E| links for flooding, where |V| and |E| are the number of nodes and links, respectively. For networks that allow broadcast transmissions, a node having no children for source u is a leaf and need not forward updates generated by u. In typical networks, most nodes are leaves. In addition, a node having only only one child for source u can send the updates generated by u to only that child (receiver directed), instead of broadcasting the updates to all neighbors.

TBRPF can be easily 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, they

can be used instead of hierarchical routing or in combination with hierarchical routing, to reduce update traffic more than is possible by using hierarchical routing alone.

2. Assumptions and Terminology

TBRPF requires very few assumptions regarding the network model. TBRPF can operate in any internet or subnet in which IP hosts are connected through broadcast or point-to-point links to routers. The current implementation of TBRPF operates on top of IP, similarly to an internet routing protocol such as OSPF. TBRPF can operate in ad-hoc networks that use different medium access control (MAC) protocols, which need not permit promiscuous listening of transmissions.

To describe TBRPF, the topology of a network is modeled as a directed graph G = (V, E), where V is the set of nodes and E is the set of edges or links. Each node has a unique identifier and represents a router that runs TBRPF. In a wireless network, a node can have connectivity with multiple nodes over a single physical radio link. We consider two nodes u and v to be adjacent (i.e., neighbors) if each node can reliably receive messages from the other. Thus, we map a physical broadcast link connecting multiple nodes into multiple point-to-point bidirectional links. Such a bidirectional link between two nodes u and v is represented by a pair of links (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). Each router u is responsible for updating and reporting the cost and up/down status of each outgoing link (u,v) to neighbors. TBRPF can also support the broadcast of multiple link costs (or link metrics) for purposes of quality-of-service routing.

TBRPF uses a neighbor discovery protocol based on hello packets to establish links to new neighbors and to detect link failures. TBRPF supports both unicast transmissions (e.g., point-to-point or receiver directed), in which a packet reaches only a single neighbor; and broadcast transmissions, in which a single packet is transmitted simultaneously to all neighbors. In particular, TBRPF allows an update to be sent either on a common broadcast channel or on one or more unicast channels, depending on the number of neighbors that need to receive the update.

3. Conceptual Data Structures and Messages

A link-state 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 update. A cost of infinity represents a failed link. We say that node u is the head node of link (u,v). It is the only node that can report changes to parameters of link (u,v). Therefore, any link-state update (u,v,c,sn) originates from node u.

Each node i maintains a counter SN_i, which is incremented by at least one each time the cost of one or more outgoing links (i,v) changes value. For example, SN_i can be a time stamp that represents the number of seconds (or other units of time) elapsed from some fixed time. When node i generates a link-state update (i,v,c,sn), the sequence number sn is set to the current value of SN_i. We note that, unlike most link-state protocols, TBRPF does not require link-state updates to contain an age field.

TBRPF 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 node i. 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. TBRPF can optionally support the dissemination of multiple link metrics, in which case the single cost c would be replaced by a vector of multiple metrics.

2. The list of neighbor nodes, denoted N_i.

3. The following is maintained for each node u not equal to i:

   a. The parent, denoted p_i(u), which is the neighbor of i that is the next node on the minimum-hop path from node i to node u, as obtained from TT_i.

   b. A list of children, denoted children_i(u).

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

   d. The routing table entry for node u, consisting of the next node on the preferred path to u.

We note that the routing table entry for node u can be equal to the parent p_i(u) if minimum-hop routing is used for data packets. However, in general, the routing table entry for node u is not p_i(u), since the selection of routes for data traffic can be based on any objective.

TBRPF uses the following message types. Message formats for the current implementation of TBRPF are specified in Section 6. However, as TBRPF is work in progress, message formats are subject to change.

Link-State Update (LSU)
: A message containing one or more link-state updates (u,v,c,sn).

NEW PARENT
: A message informing a neighbor that it has been selected as a parent with respect to one or more sources.

CANCEL PARENT
: A message informing a neighbor that it is no longer a parent with respect to one or more sources.

HELLO
: A message sent periodically by each node i for neighbor discovery.

NEIGHBOR
: A message sent in response to a HELLO message.

NEIGHBOR ACK
: A message sent in response to a NEIGHBOR message.

ACK
: A link-level acknowledgment to a unicast transmission.

NACK
: A link-level negative acknowledgment reporting that one or more update messages sent on the broadcast channel were not received.

RETRANSMISSION OF BROADCAST
: A retransmission, on a unicast channel, of link-state updates belonging to an update message for which a NACK was received.

HEARTBEAT
: A message sent periodically on the broadcast channel when there are no updates to be sent on this channel, used to achieve reliable link-level broadcast of update messages based on NACKs.

END OF BROADCAST
: A message sent to a neighbor over a unicast channel, to report that updates originating from one or more sources are now being sent on the unicast channel instead of the broadcast channel.

4. TBRPF Operation

This section describes the network-level operation of TBRPF, with the aid of the pseudocode given at the end of the section. The next section describes the link-level functions of TBRPF, i.e., neighbor discovery and reliable transmission of control packets. Examples illustrating the operation of TBRPF and the the proof of correctness for TBRPF can be found in [1].

For a given node u, the parents p_i(u) for all i not equal to u define a minimum-hop spanning tree rooted at u (assuming the protocol has converged). The basic idea of TBRPF is to broadcast link-state updates generated by u along this spanning tree, while at the same time modifying this tree based on the link-state information received along the tree.

To simplify the presentation, we assume that the sequence number space is large enough so that wraparound does not occur. However, if a smaller space is used, wraparound can be handled by employing infrequent periodic updates with a period that is less than half the minimum wraparound period, and by using a cyclic comparison of sequence numbers: i.e., sn is considered less than sn' if either sn < sn' and their difference is less than half the largest possible sequence number, or sn' < sn and their difference is greater than half the largest possible sequence number.

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, children_i(src), at node i with respect to node src is the set of neighbors from which it has received a NEW PARENT message with source src. In general, a node will simultaneously select a neighbor as the parent for multiple sources, so that it sends a NEW PARENT(src_list, sn_list) message to the new parent, where src_list is the list of sources and sn_list is the corresponding list of sequence numbers. Similarly, a CANCEL PARENT message will generally contain a list of sources.

The broadcast of link-state information is achieved in the following manner. Any link-state update originating from node src is accepted by node i if (1) it is received from the parent node p_i(src), and (2) 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).

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, establishes a link to a new neighbor, or detects the failure or change in cost of a link to an existing neighbor. A node computes its parents by (1) computing minimum-hop paths to all other nodes using Dijkstra's algorithm, and then (2) selecting the parent for each source src to be the next node on the min-hop path to src (see the procedure Compute_New_Parents).

Then, if the parent p_i(src) has changed, node i 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) is the sequence number of the most recent link-state update originating from node src received by node i. This number indicates 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 CANCEL PARENT(src) message, the old parent k removes node i from the list children_k(src). Upon receiving the NEW PARENT(src, sn) message, the new parent j = p_i(src) adds node i to the list children_j(src) and sends to node i a link-state update 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). Thus, only updates not yet known to node i are sent to node i.

When a node i detects the existence of a new neighbor nbr, it executes 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 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 Link_Down(i, nbr). 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.

The method for assigning costs to links is beyond the scope of this specification. As an example, the cost of a link could simply be one (for min-hop routing), or the link delay plus a constant bias.

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) = NULL (i.e., not defined), children_i(src) is the empty

set, and sn_i(src) = 0 for every node src. Every node then executes Link_Up to process each link established with a neighbor, which will result in a NEW PARENT message being sent to each neighbor. Since each neighbor j of node i is (trivially) the next node on the min-hop path from i to j, p_i(j) = j whenever j is a neighbor of i. Therefore, the NEW PARENT message sent to a new neighbor j contains j (and possibly other sources) in its source list.

It is helpful to understand how TBRPF works when initially all nodes of an ad-hoc network have no topology information. Each node i will first select each of its neighbor j as a new parent p_i(j) for source j. Then each neighbor j will inform node i of its outgoing links, which will allow node i to compute min-hop paths, and thus new parents p_i(u), for all sources u that are two hops away. Then each parent p_i(u) for each such u will inform node i of node u's outgoing links, which will allow node i to compute new parents for all sources that are three hops away. This process continues until node i has computed parents for all sources u.

TBRPF does not require an age field in link-state updates. However, failed links (represented by an infinite cost) and links that are unreachable (i.e., links (u,v) such that p_i(u) = NULL) are deleted from the topology table after MAX_AGE seconds (e.g., 1 hour) in order to conserve memory. The reason these updates are not deleted immediately is as follows. Failed links (u,v) must be maintained for some time in the topology table to ensure that a node i that changes its parent p_i(u) near the time of failure (or had no parent p_i(u) during the failure) is informed of the failure by the new parent. Unreachable links, i.e., links (u,v) such that i and u are on different sides of a network partition, are maintained for a period of time to avoid having to rebroadcast the old link state for (u,v) throughout i's side of the partition, if the network partition recovers soon (which is likely to happen if the network partition is caused by a marginal link that oscillates between the up and down states). This property is important to ensure the efficiency of TBRPF when network partitions are common.

A node that is turned off (or goes to sleep) operates as if the links to all neighbors have gone down. Thus, the node remembers the link-state information it had when it was turned off. Since all such links are either down or unreachable, these link states are deleted from the topology table if the node awakens after being in sleep mode for more than MAX_AGE seconds.

Periodic updates are not strictly required for the correctness of TBRPF. However, we allow infrequent periodic updates to correct errors that may occur in table entries or update messages. (See Send_Periodic_Updates.) As discussed above, periodic updates are

also useful if the sequence number space is not large enough to avoid wraparound.

TBRPF provides each node with complete link-state information. Each node can then apply a path selection algorithm to compute preferred paths to all possible destinations, and to update these paths when link states are updated. The default path selection algorithm for TBRPF is to apply Dijkstra's algorithm to compute shortest paths (with respect to c) to all destinations. However, TBRPF can employ any path selection algorithm. Once preferred paths are computed, the routing table entry for node u is set to the next node on the preferred path to u. If min-hop routing is desired, then the routing table entry for u can be set to the parent p_i(u). In the current implementation of TBRPF, data packets are forwarded based on the routing table as in IP routing. However, source routing can also be used.

4.1. Pseudocode

The following pseudocode describes the network-level procedures performed at node i by TBRPF. The notation LSU(update_list) represents a link-state-update message that includes the updates (u,v,c,sn) in update_list.

```
Process_Update(i, nbr, in_message){
(Called when an update message in_message is received from nbr.)
  Update_Topology_Table(i, nbr, in_message, update_list).
  Update_Parents(i).
    For each node src in TT_i {
       Let update_list(src) consist of all tuples (k,l,c,sn)
       in update_list such that k = src.
       If update_list(src) is nonempty
         Send message LSU(update_list(src)) to children_i(src).}}

Update_Topology_Table(i, nbr, in_message, update_list){
  Set update_list to empty list.
  For each ((u,v,c,sn) in in_message) {
    If (p_i(u) == nbr) {
      If ((u,v) is in TT_i and sn > TT_i(u,v).sn) {
        Add (u,v,c,sn) to update_list.
        Set TT_i(u,v).sn = sn.
        Set TT_i(u,v).c = c.
        If (sn > sn_i(u)) Set sn_i(u) = sn.}
      If ((u,v) is not in TT_i) {
        Add (u,v,c,sn) to TT_i.
        Add (u,v,c,sn) to update_list.
        If (sn > sn_i(u)) Set sn_i(u) = sn.}}}}
```

```
Link_Change(i,j){
(Called when the cost of link (i,j) changes.)
  If (|TT_i(i,j).c - cost(i,j)|/TT_i(i,j).c > epsilon) {
    Set TT_i(i,j).c = cost(i,j).
    Set TT_i(i,j).sn = current time stamp SN_i.
    Set update_list = {(i, j, TT_i(i, j).c, TT_i(i, j).sn)}.
    Send message LSU(update_list) to children_i(i).}}

Link_Down(i,j){
(Called when link (i,j) goes down.)
  Remove j from N_i.
  Set TT_i(i,j).c = infinity.
  Set TT_i(i,j).sn = current time stamp SN_i.
  Update_Parents(i).
  For each (node src in TT_i) remove j from children_i(src).
  Set update_list = {(i,j, infinity, TT_i(i,j).sn)}.
  Send message LSU(update_list) to children_i(i).}

Link_Up(i,j){
(Called when link (i,j) comes up.)
  Add j to N_i.
  Set TT_i(i,j).c = cost(i,j).
  Set TT_i(i,j).sn = current time stamp SN_i.
  Update_Parents(i).
  Set update_list = {(i, j, TT_i(i,j).c, TT_i(i,j).sn)}.
  Send message LSU(update_list) to children_i(i).}

Update_Parents(i){
  Compute_New_Parents(i).
  For each (node k in N_i)
    Set cancel_src_list(k), src_list(k), and sn_list(k) to empty.
  For each (node src in TT_i such that src != i){
    If (new_p_i(src) != p_i(src)){
      If (p_i(src) != NULL){
        Set k = p_i(src).
        Add src to cancel_src_list(k).}
      Set p_i(src) = new_p_i(src).
      If (new_p_i(src) != NULL){
        Set k = new_p_i(src).
        Add src to src_list(k).
        Add sn_i(src) to sn_list(k).}}}
  For each (node k in N_i){
    If (src_list(k) is nonempty)
      Send message NEW PARENT(src_list(k), sn_list(k)) to k.
    If (cancel_src_list(k) is nonempty)
      Send message CANCEL PARENT(cancel_src_list(k)) to k.}}
```

```
Compute_New_Parents(i){
  For each (node src in TT_i such that src != i)
    Set new_p_i(src) = NULL.
  Compute min-hop paths using Dijkstra.
  For each (node src in TT_i such that src != i)
    Set new_p_i(src) equal to the neighbor of node i along the
    minimum hop path from i to src.}

Process_New_Parent(i, nbr, src_list, sn_list){
(Called when node i receives a NEW PARENT(src_list, sn_list)
message from nbr.)
  Set update_list to empty list.
  For each (node src in src_list) {
    Let sn_list.src denote the sequence number
    corresponding to src in sn_list.
    Add nbr to children_i(src).
    Set new_updates = {(k,l,c,sn) in TT_i such that k = src
    and sn > sn_list.src}.
    Add new_updates to update_list.}
  Send message LSU(update_list) to nbr.}

Process_Cancel_Parent(i,nbr,src_list )
(Called when node i receives a CANCEL PARENT(src_list)
message from nbr.)
  For each (node src in src_list) remove nbr from children_i(src).

Send_Periodic_Updates(i){
  Set update_list to empty.
  For each (j in N_i such that TT_i(i,j). c != infinity){
    Set TT_i(i,j).sn = current time stamp SN_i.
    Add (i, j, TT_i(i,j).c, TT_i(i,j).sn) to update_list. }
  Send message LSU(update_list) to children_i(i).}
```

5. Link-Level Functions

This section describes the link-level functions of TBRPF, i.e., neighbor discovery and the reliable link-level transmission of control messages.

5.1. Neighbor Discovery

The neighbor discovery protocol detects the following events:

1. A link to a new neighbor is established,
2. The link to an existing neighbor goes down.

Neighbor discovery uses the following three types of control messages: HELLO, NEIGHBOR, and NEIGHBOR ACK. Every HELLO_INTVL seconds, each node i transmits, on the broadcast channel, a HELLO message containing the identity of node i. Upon receiving a HELLO message from a new neighbor i, a node j responds with a NEIGHBOR message containing the identity of node j. Finally, upon receiving the NEIGHBOR message, node i sends to node j a NEIGHBOR ACK containing the identity of node i. NEIGHBOR and NEIGHBOR ACK messages also contain the current link-level sequence number for the broadcast channel (discussed below). A link from node i to node j is established if node i receives a NEIGHBOR packet or NEIGHBOR ACK from node j.

The link to an existing neighbor is declared to be down if no traffic (including HELLO messages and ACKs) has been received from the neighbor in the last LINKDOWN_INTVL seconds.

5.2. Reliable Link-Level Transmission of Control Messages

In most link-state routing protocols, e.g., OSPF, each node forwards the same link-state information to all neighbors. In contrast, in TBRPF each node sends each link-state update only to neighbors that are children on the minimum-hop path tree rooted at the source of the update. TBRPF can therefore utilize bandwidth more efficiently by using unicast transmissions if only one child (or a few children) exists for the update source, and broadcast transmissions when several children exist for the update. Therefore, TBRPF uses a rule to decide whether to use unicast or broadcast transmissions, depending on the number of children and the total number of neighbors.

Updates with only one intended receiver (e.g., only one child), should use unicast transmissions. Updates with several intended receivers should use broadcast transmissions, to avoid transmitting the message several times. Therefore, it is reasonable to use the following simple transmission rule, where k is the number of intended receivers: Use unicast if k = 1 and use broadcast if k > 1. This simple rule has the following possible drawback, where n is the number of neighbors. If k = 2 and n = 20, then 18 neighbors are wasting time listening to a message not intended for them when they could be sending or receiving another message. To avoid this drawback, another (optional) rule is to use broadcast if k > (n+1)/2 and unicast otherwise. In general, any rule of the form k > g(n) can be used. We note that, for update messages, the number of children k may be different for different update sources. Therefore, it is possible to use unicast for some sources and broadcast for other sources, and the transmission mode for a given source u, denoted mode_i(u), can change dynamically between unicast and broadcast as the number of children changes.

While Link-State Update messages can be transmitted in either unicast or broadcast mode, HELLO messages and HEARTBEAT messages (discussed below) are always transmitted on the broadcast channel, and the following messages are always transmitted on the unicast channel (to a single neighbor): NEIGHBOR, NEIGHBOR ACK, ACK, NACK, NEW PARENT, CANCEL PARENT, RETRANSMISSION OF BROADCAST, END OF BROADCAST, and Link-State Update messages sent in response to a NEW PARENT message.

The procedure for sending a Link-State Update message (that is not a response to a NEW PARENT message) on the broadcast or unicast channel is as follows:

```
If (mode_i(src) == BROADCAST)
  Append the message update_msg to the message queue
  associated with the broadcast channel.
If (mode_i(src) == UNICAST)
  For (each node k in children_i(src))
    Append the message update_msg to the message queue
    associated with the unicast channel to node k.
```

The reliable unicast transmission of control packets MUST be provided, either by an underlying link-layer protocol or by TBRPF itself. Any existing method for reliable link-layer unicast transmission can be used. Such methods typically use sequence numbers and ACKs, in which a packet is retransmitted if an ACK for it is not received within a given amount of time. The method used for reliable unicast of control packets is not part of the TBRPF specification.

The following two subsections describe the mechanism for the reliable transmission of update messages in broadcast mode, and the procedure for dynamically selecting the transmission mode.

5.2.1. Reliable Transmission of Update Messages in Broadcast Mode

In this subsection, we describe the procedure for the reliable transmission of Link-State Update messages in broadcast mode. A broadcast update message includes one or more link-state updates, denoted lsu(src), originating from sources src for which the transmission mode is BROADCAST. Each broadcast control packet is identified by a sequence number that is incremented each time a new broadcast control packet is transmitted.

Before describing the procedure, we discuss a few of its properties. NACKs are used instead of ACKs for reliable transmission, so that the amount of ACK/NACK traffic is minimized if most transmissions are successful. Suppose node i receives a NACK from a neighbor k for a broadcast update message. Then all updates lsu(src) in the original

message, for each node src such that k belongs to children_i(src), are retransmitted (reliably) on the UNICAST channel to neighbor k, in a RETRANSMISSION OF BROADCAST message. This message includes the original broadcast sequence number to allow node k to process the updates in the correct order.

The procedure for the reliable transmission of broadcast update packets uses the following message types (in addition to Link-State Update messages): HEARTBEAT(sn), NACK(sn, bit_map), and RETRANSMISSION OF BROADCAST(sn, update_msg). A NACK(sn, bit_map) message contains the sequence number (sn) of the last received broadcast control packet, and a 16-bit vector (bit_map) specifying which of the 16 broadcast control packets from sn-15 to sn have been successfully received.

The description of the procedure at node i requires the following notation:

Pkt(sn)
: Control packet with sequence number sn transmitted on the broadcast channel by node i.

MsgQ
: Message queue for new control messages to be sent on the broadcast channel from node i.

brdcst_sn_i
: sequence number of the last packet transmitted on the broadcast channel by node i.

Heartbeat_Timer
: Timer used in the transmission of the Heartbeat message.

Following the transmission of the broadcast control packet Pkt(brdcst_sn_i) on the broadcast channel, node i increments brdcst_sn_i and reinitializes Heartbeat_Timer.

When Heartbeat_Timer expires at node i, the node appends the control message HEARTBEAT(brdcst_sn_i) to the message queue associated with the broadcast channel, and reinitializes Heartbeat_Timer.

When node i receives NACK(sn, bit_map) from node nbr, node i performs the following:

```
For each (sn' not received as indicated by bit_map){
  Let update_msg = {(src*, v*, sn*, c*) in Pkt(sn') such that
    nbr is in children_i(src*)}.
  Append the message RETRANSMISSION OF BROADCAST(sn', update_msg)
```

```
        to the message queue associated with the unicast channel to
        node nbr. (Message must be sent even if update_msg is empty.)}
```

Upon receipt at node nbr of control packet Pkt(sn) transmitted on the broadcast channel by node i, node nbr performs the following:

```
   If the control packet Pkt(sn) is received in error{
     Append the control message NACK(sn, bit_map) to the message
       queue associated with the unicast channel to node i.}
   If the control packet Pkt(sn) is received out of order (i.e.,
     at least one previous sequence number is skipped){
     Withhold the processing of the control packet Pkt(sn).
     Append the control message NACK(sn, bit_map') to the message
     queue associated with the unicast channel to node i.}
   Else (control packet Pkt(sn) is received correctly and in order){
     For each Link-State Update message update_msg in Pkt(sn), call
     Process_Update(i, nbr, update_msg).}
```

When a link is established from node i to a new neighbor k, node i obtains the current value of brdcst_sn_k from the NEIGHBOR message or NEIGHBOR ACK that was received from node k.

5.2.2. Changing the Transmission Mode Between Unicast and Broadcast

This subsection describes the mechanism used by each node i to dynamically select the transmission mode for link-state updates originating from each source node src. As discussed above, this decision uses a rule of the form k > g(n), where k is the number of children (for src) and n is the number of neighbors of node i. However, additional care must be taken to ensure that updates are received in the correct order, or that the receiver has enough information to reorder the updates. This is accomplished by sending an END OF BROADCAST(last_seq_no, src) message on the unicast channel to each child when the mode changes to UNICAST, and by waiting for all update packets sent on unicast channels to be acked on before changing to BROADCAST mode.

To facilitate this process, each node i maintains a binary variable unacked_i(nbr, src) for each neighbor nbr and source src, indicating whether there are any unacked control packets sent to nbr containing link-state updates originating at node src. The following procedure is executed periodically at each node i.

```
For each (node src){
  If (mode_i(src) = BROADCAST and |children_i(src)| <= g(n)){
    For each (node k in children_i(src)){
      Append the message END OF BROADCAST(brdcst_sn_i, src) to the
```

```
        message queue associated with the unicast channel to node k.}
      Set mode_i(src) = UNICAST.}

    If (mode_i(src) = UNICAST and |children_i(src)| > g(n)){
      Set switch_flag = YES.
      For each (node k in children_i(src)){
        If (unacked_i(k,src) = YES) Set switch_flag = NO.}
      If (switch_flag = YES) Set mode_i(src) = BROADCAST.}}
```

6. TBRPF Implementation for IPv4-based MANETs

The TBRPF routing protocols provide automatic full topology discovery with dynamic adaptation to link state changes in highly mobile environments. TBRPF is particularly well suited to MANETs consisting of mobile nodes with wireless data link interfaces operating in peer-to-peer fashion over either a single multiple access communications channel or a collection of receiver directed point-to-point channels with a multiple access broadcast channel. (An example of the former is the IEEE 802.11 Distributed Coordination Function (DCF) [13], in which the mobile nodes collectively arbitrate channel access via a Carrier Sense, Multiple Access with Collision Avoidance (CSMA/CA) strategy for all unicast, multicast and broadcast message transmissions.) Additionally, TBRPF is particularly well suited for carrying routing information that supports the standard DARPA Internet protocols (IPv4) [10]. In the following sections, we discuss assumptions for the data link layer, Internetworking assumptions, and TBRPF neighbor discovery and routing protocol message transmission in an Internet addressing environment.

6.1. Data Link Layer Assumptions

We assume a data link layer broadcast channel (*) with multiple access capabilities, such that nodes wishing to transmit IPv4 broadcast and/or multicast messages must arbitrate with other nodes for channel access before doing so. We further assume that messages transmitted over the data link layer broadcast channel will be received by only a proper subset of the collection of nodes on the multiple access data link, due to wireless interface range limitations, terrain features, and other hidden terminal conditions incurred in a mobile environment. We consider a pair of nodes (A and B) to have a bi-directional link (or simply "link") if and only if both node A can receive messages sent from node B and node B can receive messages sent from node A at a given instant in time.

(*) Some wireless data link layer protocols may provide point-to-point receiver directed (unicast) channels which operate out-of-

band with respect to the multiple access broadcast channel while others (such as IEEE 802.11) utilize a single multiple access channel for all unicast and broadcast/multicast message transmissions.

While TBRPF can be readily applied to a fixed network (in which no link state changes due to node mobility occur) such a static network configuration merely represents a simplified case of the general mobile ad-hoc network model. We therefore assume a highly dynamic mobile environment at the data link layer, in which link state changes occur frequently and rapidly. We further assume a data link layer addressing scheme that supports broadcast, multicast and unicast addressing with best-effort (not guaranteed) message delivery services between nodes with instantaneous bi-directional links. Finally, we assume that each node in the MANET has a unique data link layer unicast address assignment. While uniqueness for data link layer address assignments is difficult to guarantee, the assumption of uniqueness is consistent with current practices for the deployment of IPv4 on specific link layers, such as Ethernet [5]. Methods for duplicate data link layer address detection and deconfliction are beyond the scope of this document.

6.2. Internetworking Assumptions

While TBRPF can be readily extended to support hierarchical addressing compatible with the generalized Internet addressing architecture, we assume a "flat" addressing scheme within a singe MANET. In the flat addressing model, we assume that each node is assigned an IPv4 address [10] which presumably has some topological relevance to its "home" MANET, but we do not assume that all nodes within the MANET share a common IPv4 network prefix. While it may be reasonable to assume that all nodes in a MANET might initially be assigned IPv4 addresses with a common network prefix, dynamic topology changes may result in nodes leaving their home MANETs and subsequently joining foreign MANETs. Such mobility factors may result in an heterogeneous conglomerate of IPv4 network prefixes within a single MANET unless some form of dynamic address assignment (or other address renumbering method) is used. As with data link layer addressing, we assume that each node in the MANET is assigned a unique IPv4 address, but we require each node to provide a means for duplicate IPv4 address detection. Again, methods for duplicate address deconfliction are beyond the scope of this document.

While we make no assumptions regarding IPv4 network address prefix assignments within a MANET, we assume that each node in the MANET will participate in the TBRPF routing protocol scheme. Therefore, since each node in the MANET participates in the TBRPF dynamic full

topology discovery process, the requirement for on-demand discovery through the Address Resolution Protocol (ARP) [9] is obviated. We further assume that each node in the MANET supports the multi-hop relaying paradigm; in other words, each node in the MANET must be prepared to provide a third-party message relay service as dictated by the instantaneous MANET topology. Finally, we note that the multi-hop relay paradigm occurs at a lower network sub-layer than standard Internet routing. The multi-hop relaying paradigm provides intra-MANET message forwarding services, whereas standard Internet routing provides message forwarding between distinct IPv4 networks.

6.3. TBRPF Neighbor Discovery in IPv4 MANETs

6.3.1. Address Resolution Extensions for TBRPF Neighbor Discovery

TBRPF is an automatic, full topology discovery protocol that provides dynamic reaction to link state changes. TBRPF employs a neighbor discovery protocol that dynamically establishes bi-directional links and detects bi-directional link failures through the periodic transmission of HELLO messages. Thus, since the TBRPF neighbor discovery process is both automatic and continuous, we piggyback a data link-to-IPv4 address resolution capability onto the neighbor discovery protocol messages in MANET applications. Since address resolution is built-in to the TBRPF neighbor discovery protocol, the use of ARP [9] is deprecated for TBRPF-based MANETs. Additionally, since link state changes occur dynamically, stale ARP cache issues are avoided by handling address resolution from within the TBRPF neighbor discovery protocol itself.

6.3.2. TBRPF Neighbor Discovery Message Transmission

As discussed in the previous section, TBRPF neighbor discovery is responsible for both link state maintenance and data link-to-IPv4 address resolution in MANETs. Therefore, TBRPF neighbor discovery operates as a data link level protocol on MANETs. In the future, a new IANA [12] Ethernet protocol ID assignment for TBRPF neighbor discovery will be procured. Currently, the Ethernet protocol ID assignment for ARP (0x806) is used, since the use of ARP is deprecated for TBRPF-based MANETs.

As described in Section 5.1, TBRPF neighbor discovery message types include HELLO, NEIGHBOR, and NEIGHBOR_ACK. TBRPF HELLO messages MUST be sent to the data link level broadcast address, while NEIGHBOR and NEIGHBOR_ACK messages MUST be sent to the data link level unicast address of a neighboring node on the MANET. Implementations SHOULD

detect the event of a data link-to-IPv4 address mapping change for existing links. This may occur in one of the following instances:

1. Two or more nodes in the MANET are using the same IPv4 address.

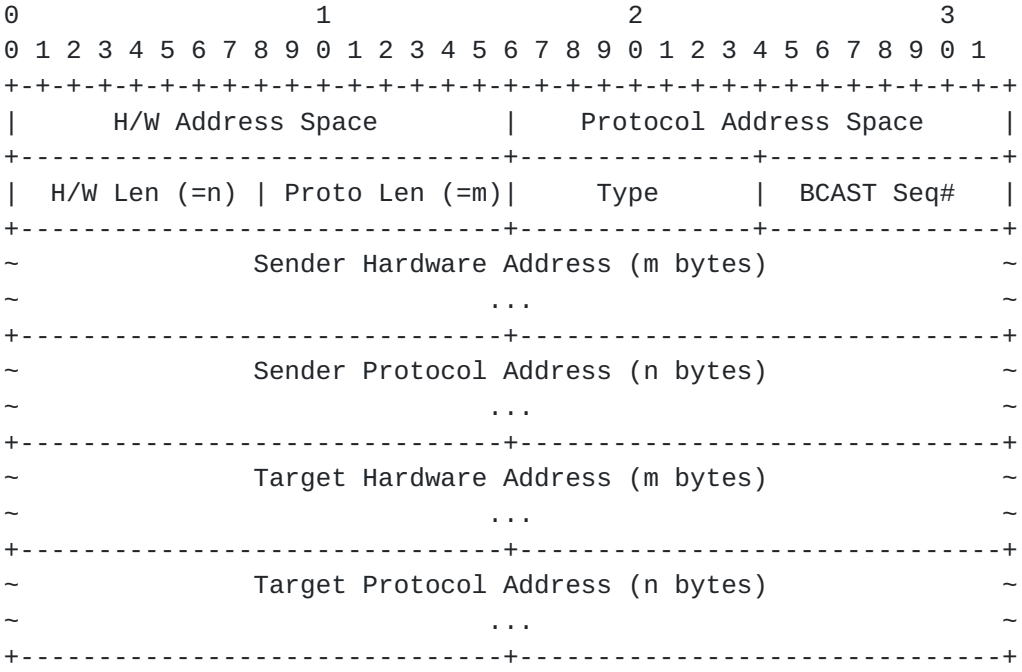
2. An existing node in the MANET has changed its data link layer address.

3. A new node is now using the IPv4 address of a former node which MAY have left the MANET.

In case 1, the implementation SHOULD print some form of "duplicate IPv4 address detected" message to the console. In cases 2 and 3, the cached link state SHOULD be updated to reflect the new data link-to-IPv4 address mapping.

6.3.3. TBRPF Neighbor Discovery Packet Format

We now present the packet format for the TBRPF HELLO, NEIGHBOR, and NEIGHBOR_ACK neighbor discovery protocol messages on MANETs. Note that the packet format is similar to the format used by ARP [9]. The data link header for each message is not shown, since it is specific to the underlying data link layer. TBRPF places no requirements on the underlying data link layer other than those described under "data link layer assumptions" above.



Type (8 bits):

HELLO	10
NEIGHBOR	11
NEIGHBOR_ACK	12

BCAST Seq# (8 bits):
A sequence number from 0...15 (effectively, only 4 bits), used in NEIGHBOR and NEIGHBOR_ACK messages as described in Section 5.1.

All Other Fields: Identical to their usage in ARP [9].

The TBRPF neighbor discovery protocol requires that each node in the MANET MUST send periodic HELLO messages to the data link broadcast address at HELLO_INTVL timeout intervals. (The HELLO_INTVL value MUST be common to all nodes within a MANET, but different MANETs MAY use different HELLO_INTVL values.) A node receiving a HELLO message from a new neighbor MUST send a NEIGHBOR message to the new neighbor's data link unicast address. Finally, a node receiving a NEIGHBOR message from a new neighbor MUST send a NEIGHBOR_ACK message to the new neighbor's data link unicast address.

As with standard ARP, the four address fields (sender hardware address; sender protocol address; target hardware address; target protocol address) facilitate the address resolution process. The fields contain the following values, based on the TBRPF neighbor discovery message opcode:

HELLO

Sender Hardware Address:	data link address of sender
Sender Protocol Address:	IPv4 address of sender
Target Hardware Address:	data link broadcast address
Target Protocol Address:	unused

NEIGHBOR

Sender Hardware Address:	data link address of sender
Sender Protocol Address:	IPv4 address of sender
Target Hardware Address:	sender H/W Address from HELLO
Target Protocol Address:	sender IPv4 Address from HELLO

NEIGHBOR_ACK

Sender Hardware Address:	data link address of sender
Sender Protocol Address:	IPv4 address of sender
Target Hardware Address:	sender H/W address from NEIGHBOR
Target Protocol Address:	sender IPv4 address from NEIGHBOR

6.4. TBRPF Protocol Messages in IPv4 MANETs

TBRPF protocol messages are exchanged between current neighbor nodes which have established bi-directional links and performed data link to IPv4 address resolution via TBRPF neighbor discovery as described in the previous section. IPv4 addresses are therefore available for use as node IDs in TBRPF protocol messages. TBRPF protocol messages are sent via the User Datagram Protocol (UDP) [11]. This approach requires an official UDP service port number registration (see: IANA Considerations). The use of UDP/IPv4 provides several advantages over a data link level approach, including:

- IP segmentation/reassembly facilities
- UDP checksum facilities
- Simple application level access for routing daemons
- IPv4 multicast addressing for link state messages

TBRPF protocol messages are sent to either the IPv4 unicast address of a current neighbor or the "All_TBRPF_Neighbors" IPv4 multicast address (see: IANA Considerations). In most cases, a message SHOULD be sent to the IPv4 unicast address of a current neighbor if ALL components of the message pertain ONLY to that neighbor. Similarly, a message SHOULD be sent to the All_TBRPF_Neighbors IPv4 multicast address if the message contains components which pertain to more than one neighbor neighbors. Receivers MUST be prepared to receive TBRPF protocol messages sent either to their own IPV4 unicast address or the All_TBRPF_Neighbors multicast address. Actual addressing strategies are highly dependent on the underlying data link layer. (**)

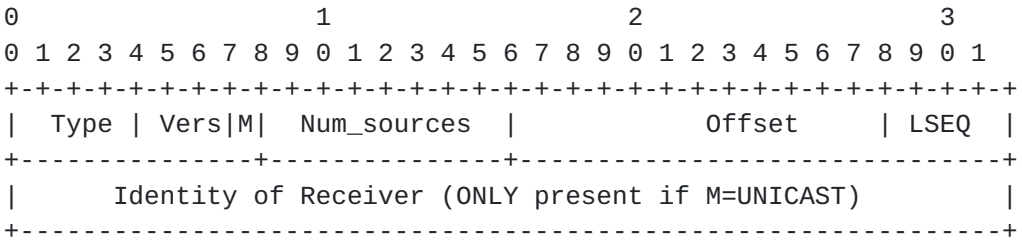
(**) For data links such as IEEE 802.11, a single, multiple access channel is available for all unicast and broadcast/multicast messages. In such cases, since channel occupancy for unicast and multicast messages is identical, it would seem frugal to send a single message to the All_TBRPF_Neighbors multicast address rather than multiple unicast messages even if the message contains components which pertain to only a subset of the current neighbors. In other cases, in which point-to-point receiver directed channels are available, sending multiple unicast messages may reduce contention on the multiple access broadcast channel. Specific link layer strategies are beyond the scope of the current document.

6.4.1. Atomic TBRPF Message Format

Individual (atomic) TBRPF messages consist of a message header followed by a message body. Atomic messages may be transmitted either individually or as components of a compound TBRPF message consisting of multiple atomic messages within a single UDP/IPv4 datagram. See

Section 6.4.2 for the compound message format.

TBRPF message headers are either 32-bits or 64-bits in length depending on whether the atomic message is BROADCAST or UNICAST. The format for the TBRPF message header (and a detailed description of the individual fields) is given below:



where the header fields are defined as follows:

Type (4 bits):
The atomic message type. The following types are currently defined:

ACK	1
NACK	2
NEW_PARENT	3
CANCEL_PARENT	4
HEARTBEAT	5
END_OF_BROADCAST	6
LINK_STATE_UPDATE_A	7
LINK_STATE_UPDATE_B	8
RETRANSMISSION_OF_BROADCAST	9

Version (3 bits):
The TBRPF protocol version field provides a transition mechanism for future versions of the TBRPF protocol. Also provides a sanity check for host software to identify bogus messages purporting to be TBRPF control messages. We currently define a single value for TBRPF version 1:

TBRPFVERSION_1	1

Mode (1 bit):
The transmission mode for this atomic TBRPF message; either UNICAST or BROADCAST. UNICAST refers to an atomic message which must be processed by only a single neighbor. BROADCAST refers to an atomic message which must be processed by ALL neighbor nodes. (NB: For IPv4 MANETs, UNICAST implies a specific IPv4 unicast address while BROADCAST implies the All_TBRPF_Neighbors

IPv4 multicast address.) The following mode bits are defined:

UNICAST	0
BROADCAST	1

Messages of type ACK, NACK, NEW_PARENT, and CANCEL_PARENT MUST be sent as UNICAST. Messages of type LINK_STATE_UPDATE_A and LINK_STATE_UPDATE_B MAY be sent as either UNICAST or BROADCAST. Messages of type RETRANSMISSION_OF_BROADCAST and END_OF_BROADCAST MUST be sent as UNICAST.

Num_Sources (==m) (8 bits):
Number of sources 'm' included in the atomic message. Takes on a value from 1...255 for messages of type: NEW_PARENT, CANCEL_PARENT, LINK_STATE_UPDATE_A, and LINK_STATE_UPDATE_B. All other message types SHOULD set Num_Sources = 0.

Offset (12 bits):
Offset (in bytes) from the 0'th byte of the current atomic message header to the 0'th byte of the NEXT atomic message header in the "compound message" (see the following section for the compound message specification.) Offset == 0 indicates no further atomic messages follow. The 12-bit offset field imposes a 4 kilobyte length restriction on individual atomic messages.

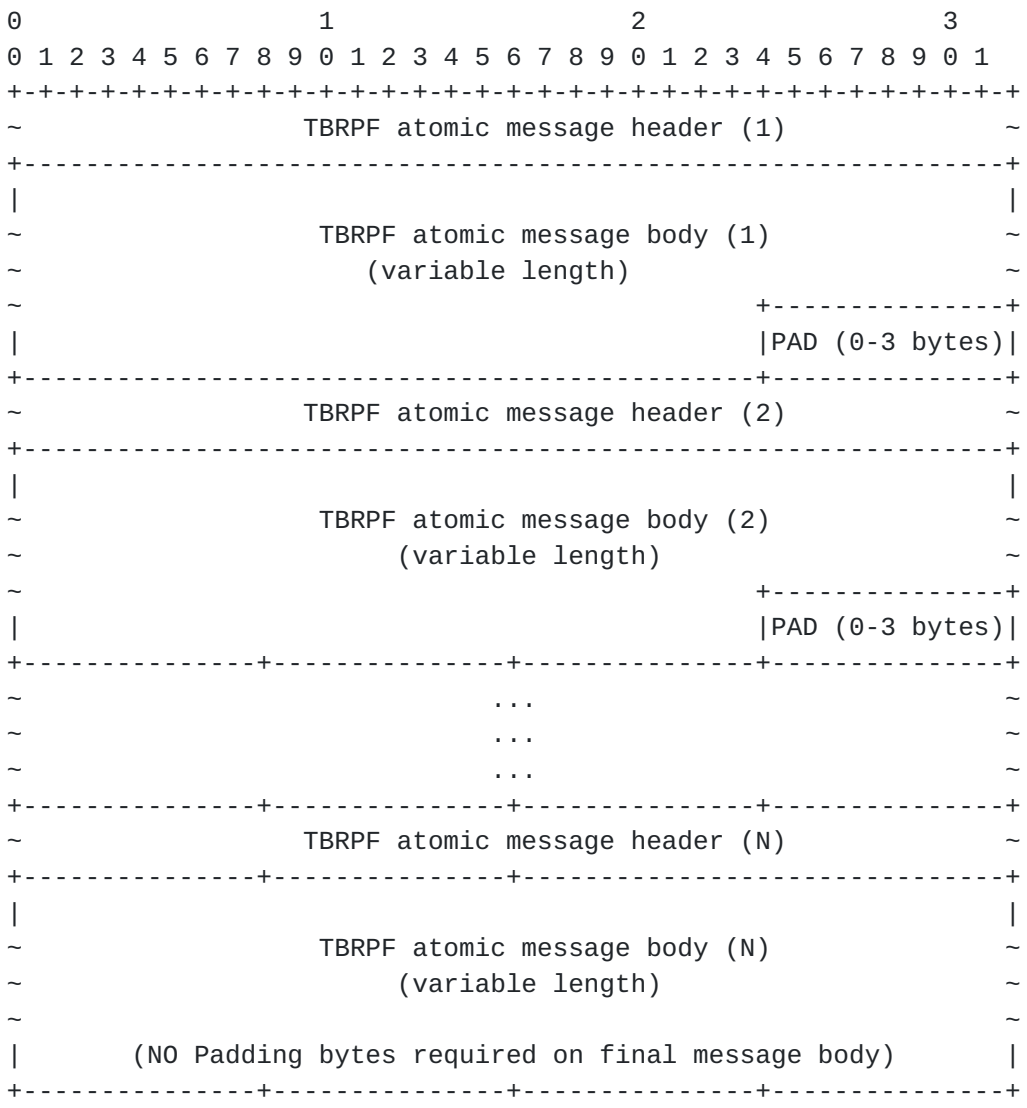
LSEQ (4 bits):
The link sequence number for this message.

Identity of Receiver (32 bits):
The IPv4 address of the receiving node which MUST process this atomic message. All other nodes MUST NOT process this atomic message. This field exists ONLY if the M bit is set to UNICAST.

6.4.2. Compound TBRPF Message Format

Multiple atomic TBRPF messages may be concatenated to form a compound message within a single UDP/IPv4 packet. Atomic message headers MUST be aligned on 32-bit boundaries, therefore an atomic message body with a non-integral number of 32-bit words will include 1, 2 or 3 padding bytes preceding a subsequent message header. NO additional header is required for a compound message; the header from the 1st atomic message serves as the initial header for the compound message.

The format for a compound TBRPF message is shown below (where N is the number of concatenated atomic messages):



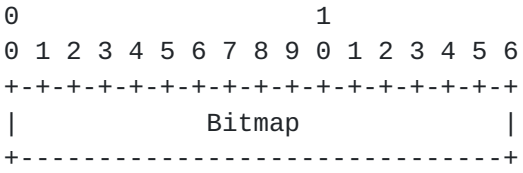
6.4.3. TBRPF Atomic Message Body Format

The format of the atomic message body depends on the 'Type' field in the corresponding message header. The following sections define the formats for atomic message bodies for TBRPF Version 1.

6.4.3.1. ACK

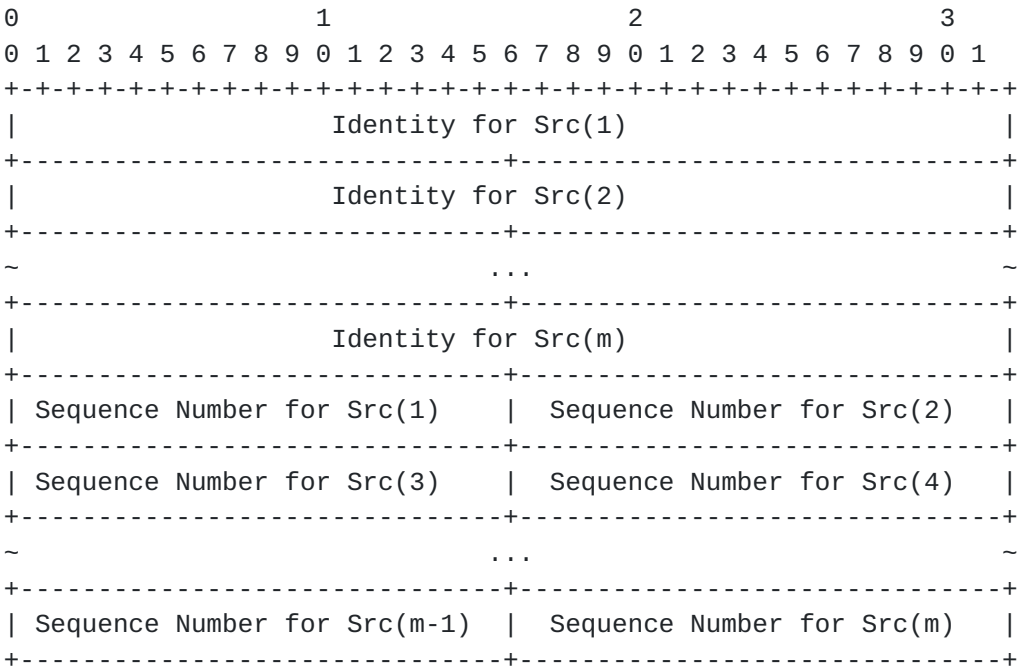
The ACK message carries a NULL message body. The 4-bit acknowledgment sequence number (from 0...15) is carried in the TBRPF message header LSEQ field.

6.4.3.2. NACK



Bitmap (16 bits):
A 16-bit vector; bits indicate messages lost/received for the 16 messages prior to the 4-bit sequence number supplied in the TBRPF message header LSEQ field. As described in Section 5.2.1, the LSEQ field is set to the sequence number of the last broadcast control message received from the neighbor to which the NACK is being sent.

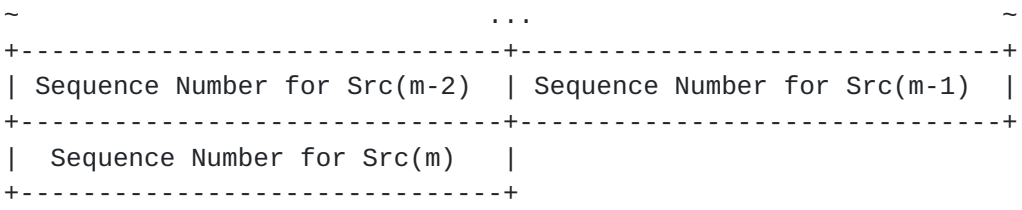
6.4.3.3. NEW_PARENT:



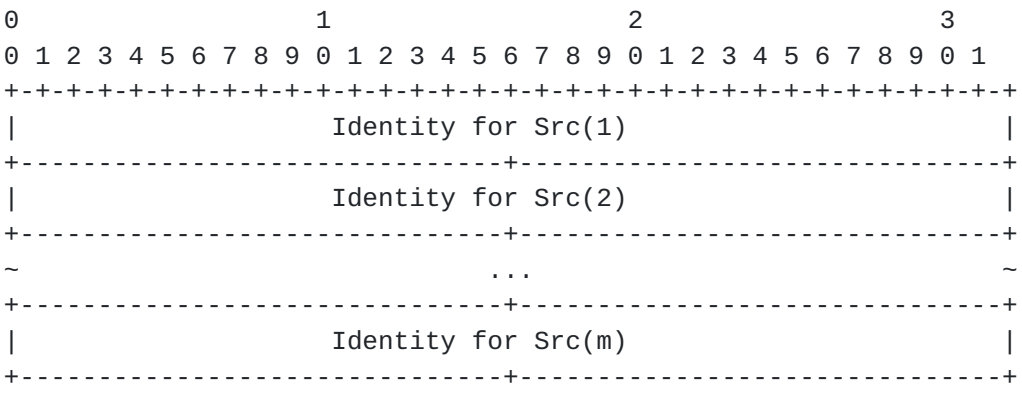
Identity for Src(i) (32 bits):
The IPv4 address of the ith Source (1 <= i <= m)

Sequence Number for Src(i) (16 bits):
A sequence number from 0...65535

NB: The above diagram shows the message format for 'm' being an even number. For 'm' being an odd number, the message ends as follows:

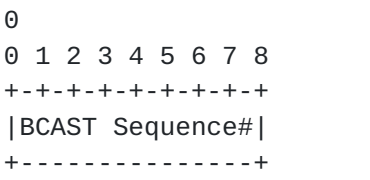


6.4.3.4. CANCEL_PARENT:



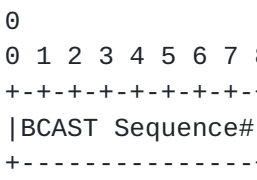
Identity for Src(i) (32 bits):
The IPv4 address of the ith Source (1 <= i <= m)

6.4.3.5. HEARTBEAT



Sequence # for BCAST Channel (8 bits):
A sequence number from 0...15 (effectively, only 4 bits)

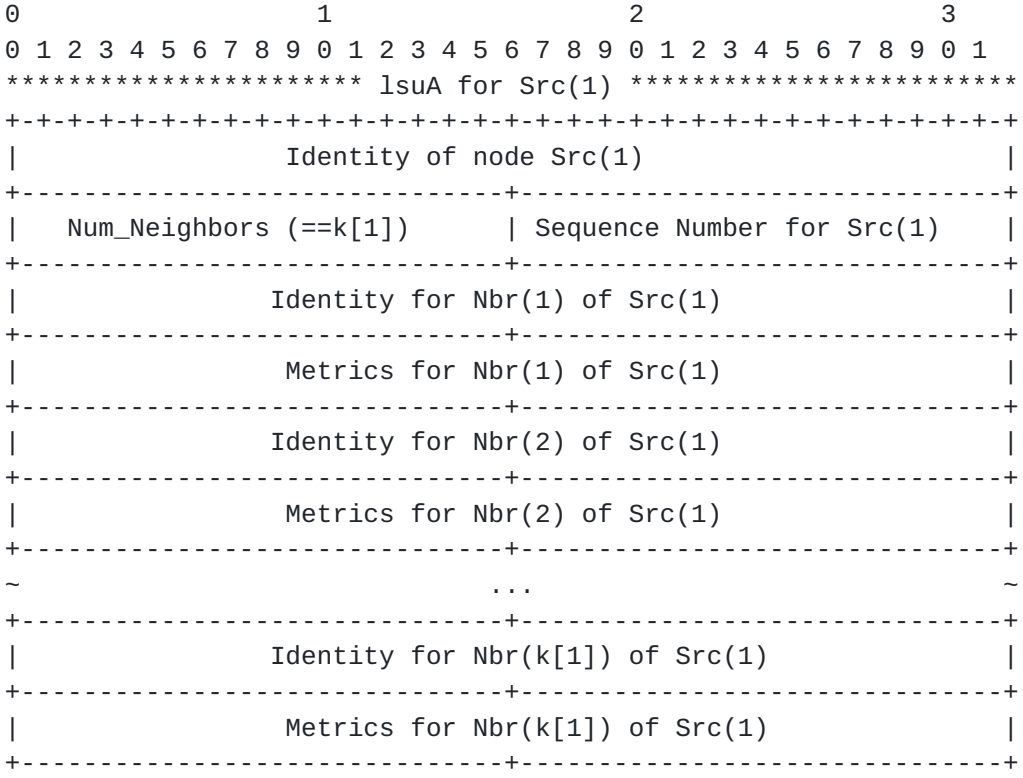
6.4.3.6. END_OF_BROADCAST

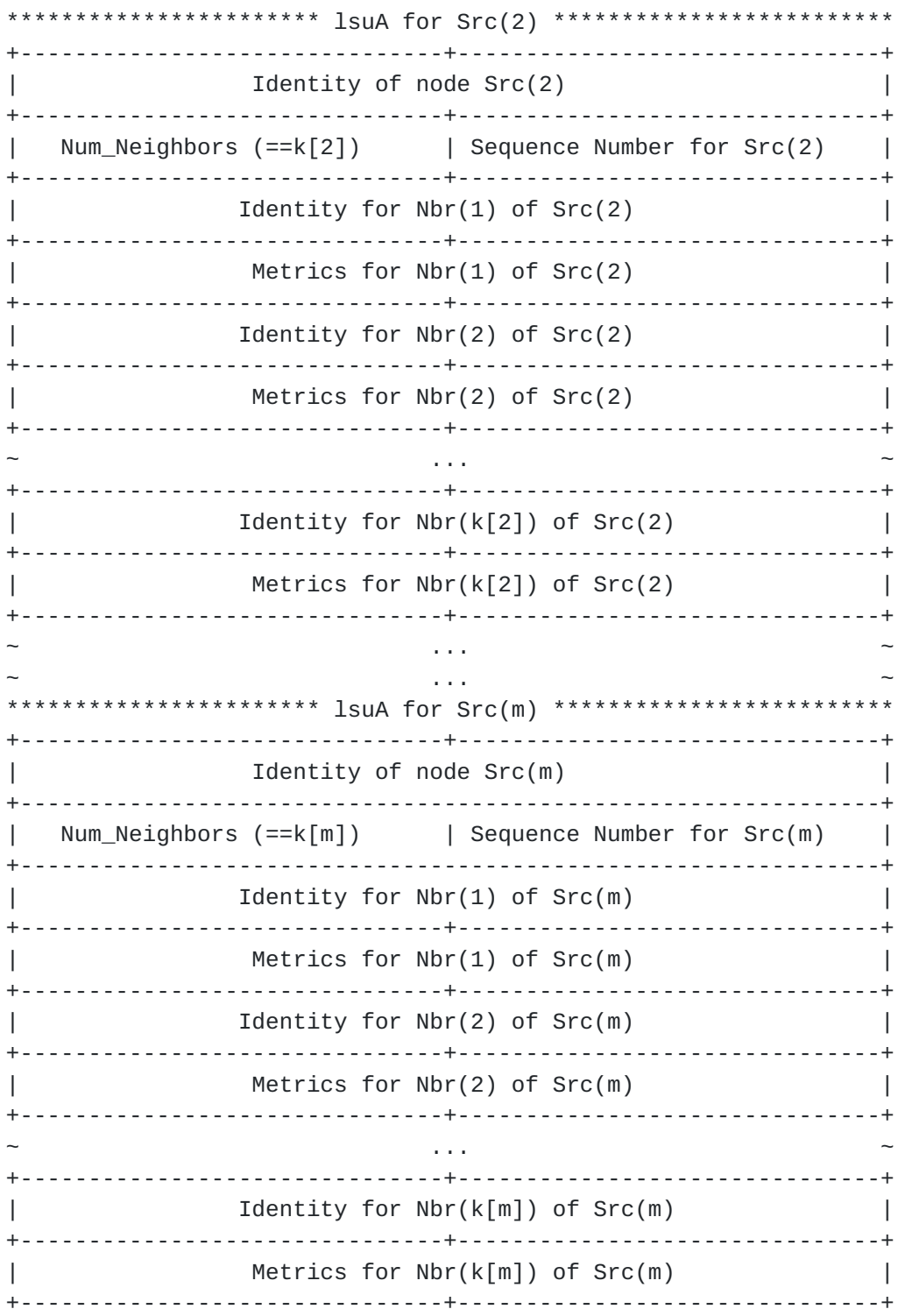


Sequence # for BCAST Channel (8 bits):
A sequence number from 0...15 (effectively, only 4 bits)

6.4.3.7. LINK_STATE_UPDATE_A

TBRPF provides two formats for link-state update messages. Messages of type LINK_STATE_UPDATE_A include a single sequence number per source, and is therefore used only if the updates for all links coming out of the same source have the same sequence number. (For example, periodic updates have this property.) This is done to reduce the message size. Messages of type LINK_STATE_UPDATE_B include a separate sequence number for each link.





Each lsuA for Src(i), for (1 <= i <= m), contains:

Identity for Src(i) (32 bits):
The IPv4 address of Src(i)

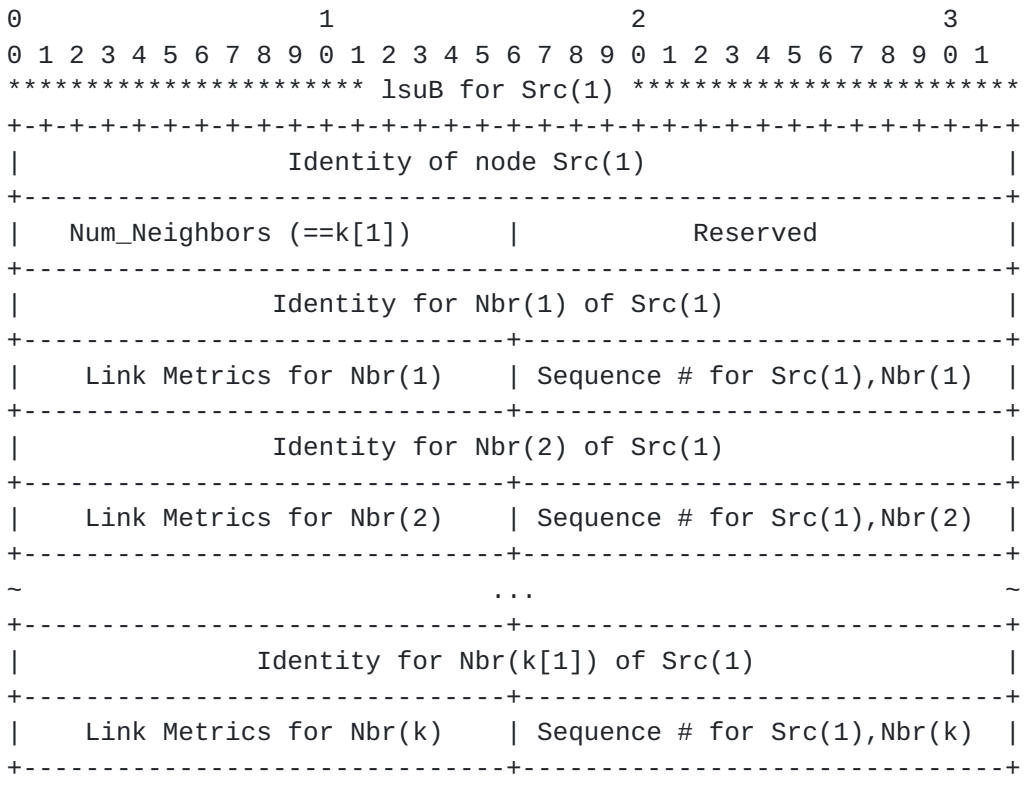
Num_Neighbors (==k[i]) (16 bits):
The number of neighbors 'k[i]' of Src(i)

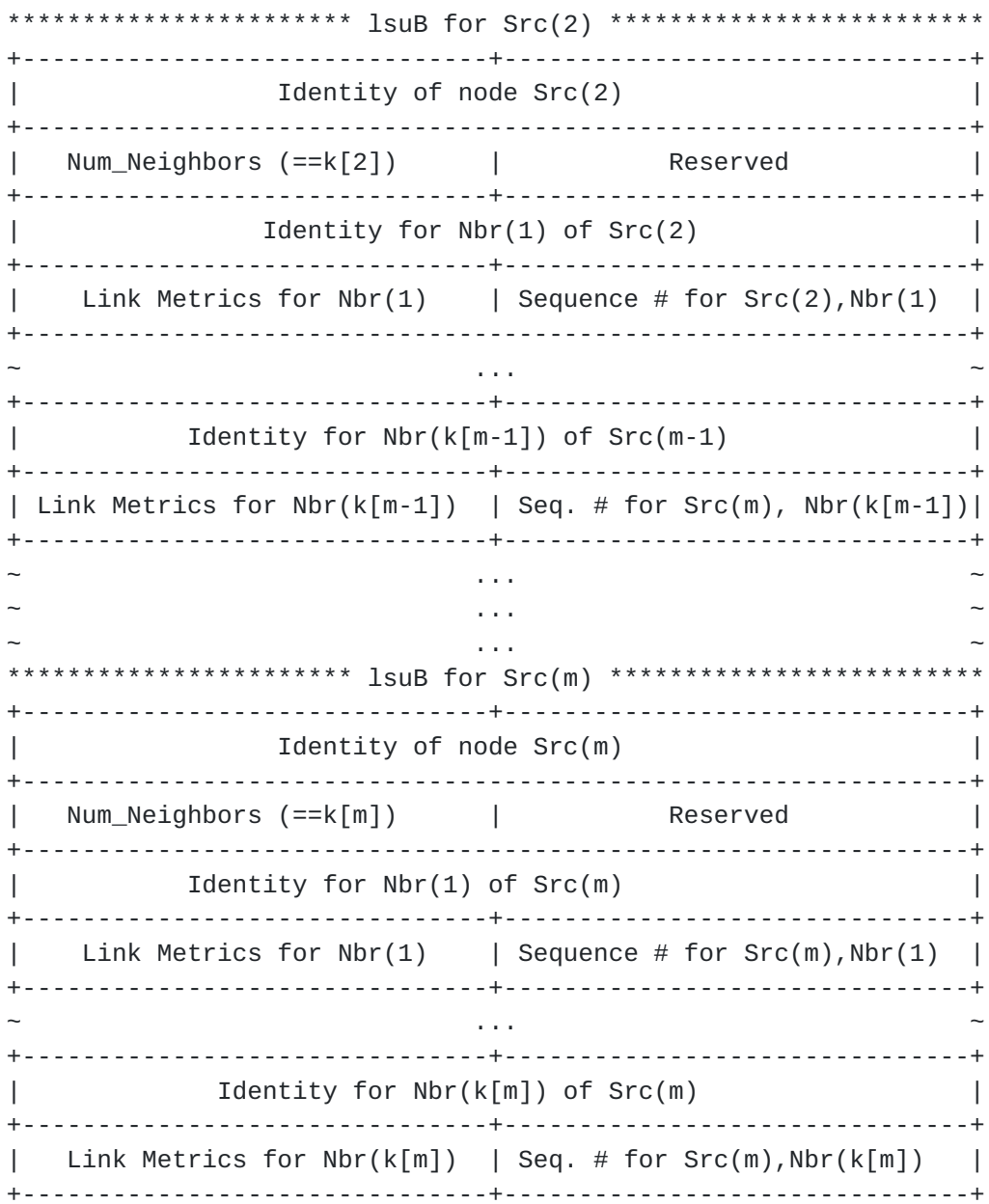
Sequence Number for Src(i) (16 bits):
A sequence number from 0...65535

Identity for Nbr(j), for (1 <= j <= k[i]) (32 bits):
The IPv4 address of Nbr(j) of Src(i)

Link Metrics for Nbr(j), for (1 <= j <= k[i]) (32 bits):
The link metrics associated with Nbr(j) of Src(i)

6.4.3.8. LINK_STATE_UPDATE_B





Each lsuB for Src(i), for (1 <= i <= m), contains:

Identity for Src(i) (32 bits):
The IPv4 address of Src(i)

Num_Neighbors (==k[i]) (16 bits):
The number of neighbors 'k[i]' of Src(i)

Identity for Nbr(j), for (1 <= j <= k[i]) (32 bits):
The IPv4 address of Nbr(j) of Src(i)

Link Metrics for Nbr(j), for (1 <= j <= k[i]) (16 bits):
The link metrics associated with Nbr(j) of Src(i) (really, only 8 bits are used)

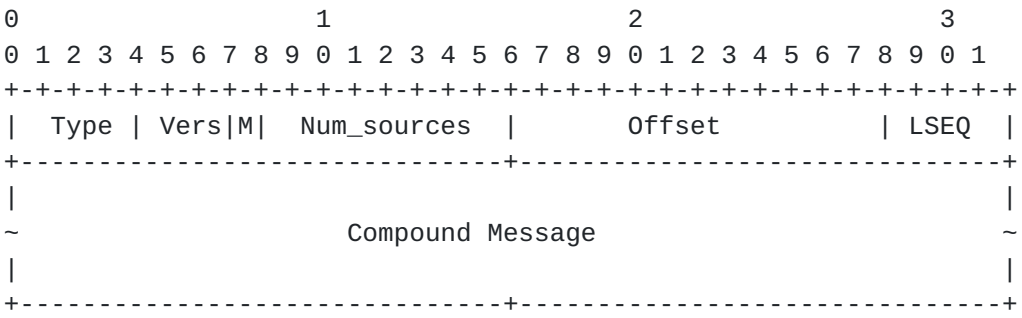
Sequence # for Src(i), Nbr(j) for (1 <= j <= k[i]) (16 bits):
Sequence number associated with link Src(i), Nbr(j)

6.4.3.9. RETRANSMISSION_OF_BROADCAST

The RETRANSMISSION_OF_BROADCAST message provides the retransmission of a compound update message in response to a NACK message. As described in Section 5.2.1, broadcast update messages are retransmitted only on unicast channels. The compound message has the same form as that shown in Section 6.4.2, and may contain one or more atomic messages of type LINK_STATE_UPDATE_A or LINK_STATE_UPDATE_B concatenated together.

The compound message is preceded by an atomic message header, where LSEQ is the sequence number corresponding to the unicast channel on which the message is sent. The LSEQ of each atomic message in the compound message is the broadcast sequence number that was included in the original (broadcast) transmission of the message. Multiple RETRANSMISSION_OF_BROADCAST messages can be bundled into a compound message as described in Section 6.4.2.

The RETRANSMISSION_OF_BROADCAST message has the following format:



where the header fields have the following values:

Type (4 bits):
MUST be set to RETRANSMISSION_OF_BROADCAST (=9).

Version (3 bits):
Set to TBRPFVERSION_1 (=1) for the initial version.

Mode (1 bit):
MUST be set to UNICAST (=0).

Num_Sources (8 bits):
SHOULD be set to 0.

Offset (16 bits):
Offset (in bytes) from the 0'th byte of the current compound message header to the 0'th byte of the NEXT compound message header in the RETRANSMISSION_OF_BROADCAST message. Allows concatenation of compound messages up to 64 kilobytes in length.

7. IANA Considerations

An application of TBRPF for IPv4-based MANETs will require the following assigned number procurements from IANA:

1. an official IPv4 multicast address assignment for All_TBRPF_Neighbors,

2. an official UDP service port number for TBRPF protocol messages,

3. an official ETHERTYPE assignment for TBRPF Neighbor Discovery.

As an alternative to 3., TBRPF could continue to use the existing ETHERTYPE assignment for ARP and new ARP opcodes could be reserved for the three TBRPF neighbor discovery message types presented in Section XXX. It must be noted, however, that TBRPF neighbor discovery and ARP are mutually exclusive. TBRPF neighbor discovery is NOT an extension of ARP.

8. Security Considerations

Authentication issues exist for allowing "foreign" nodes to join a MANET via TBRPF neighbor discovery. Additionally, privacy issues exist for any networking protocols run over unencrypted wireless data links such as IEEE 802.11. Finally, denial-of-service attacks are possible if rogue nodes join a TBRPF MANET and offer to provide a multi-hop relay service, but then fail to perform the service when it is required. We believe that IPSec may be useful in addressing some/all of these issues.

9. Implementation Status

The current version of the TBRPF protocol as it applies to IPv4 MANETs (as described in this document) has been implemented in the FreeBSD V3.2 operating system with the Merit Multi-Threaded Routing Toolkit daemon (mrtd). The current implementation has been in use for laboratory and fielded experiments since June 1999.

10. References

[1] Bhargav Bellur and Richard G. Ogier. A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks. Proc. IEEE INFOCOM '99, New York, March 1999.

[2] Scott Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[3] M. Scott Corson and Joe Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Consideration. RFC 2501, 1999.

[4] J.J Garcia-Luna-Aceves and M. Spohn. Efficient Routing in Packet-Radio Networks Using Link-State Information. Proc. IEEE WCNC '99, September 1999.

[5] C. Hornig. Standard for the Transmission of IP Datagrams over Ethernet Networks. STD0041, April 1984.

[6] David B. Johnson and David A. Maltz. Protocols for Adaptive Wireless and Mobile Networking. IEEE Personal Communications, Vol. 3, No. 1, pp 34-42, February 1996.

[7] Richard G. Ogier. Efficient Routing Protocols for Packet-Radio Networks Based on Tree Sharing. Proc. Sixth IEEE Intl. Workshop on Mobile Multimedia Communications (MOMUC'99), November 1999.

[8] Charles E. Perkins, Elizabeth M Royer, and Samir R. Das. Ad Hoc On-Demand Distance Vector (AODV) Routing. draft-ietf-manet-aodv-05.txt, March 2000 (work in progress).

[9] David C. Plummer. An Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit Ethernet addresses for transmission on Ethernet hardware. RFC 826, November 1982.

[10] J. Postel. Internet Protocol. RFC 791, September 1981.

[11] J. Postel. User Datagram Protocol. RFC 768, August 1980.

[12] J. Reynolds and J. Postel. Assigned Numbers. RFC 1700, October 1994.

[13] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ISO/IEC Std. 8802-11, ANSI/IEEE Std 802.11, 1999.

Authors' Addresses

Bhargav Bellur
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
USA

Phone: +1 650 859-6335
Fax: +1 650 859-4812
Email: bhargav@erg.sri.com

Richard Ogier
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
USA

Phone: +1 650 859-4216
Fax: +1 650 859-4812
Email: ogier@erg.sri.com

Fred L. Templin
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
USA

Phone: +1 650 859-3144
Fax: +1 650 859-4812
Email: templin@erg.sri.com