An Adaptive Protocol for Reliable Multicast in Mobile Multi-hop Radio Networks

Sandeep K. S. Gupta and Pradip K. Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523
{gupta, srimani}@cs.colostate.edu

Abstract

In this paper we propose a new protocol for reliable multicast in a multi-hop mobile radio network. The protocol is reliable, i.e., it guarantees message delivery to all multicast nodes even when the topology of the network changes during multicasting. The proposed protocol uses a core based shared tree. The multicast tree may get fragmented due to node movements. A notion of forwarding region is introduced which is used to glue together fragments of a multicast trees. The gluing process involves flooding of the forwarding region of the nodes which witness topology change due to node movements. Delivery of multicast messages to mobile nodes is expedited through i) pushing of the message by witness nodes in their forwarding regions and ii) pulling of messages by a mobile node during (re)joining process. Hence, the protocol conserves network bandwidth by using a combination of the push-pull approach and limiting network flooding to only minimal parts of the network that is affected by topology change. The protocol adapts to both topology change and the distribution of the multicast group members to minimize the use of system resources.

Keywords: *Mobile Ad Hoc Networks, Mobile Multi-hop Networks, Multicast Routing, Group Communication, Reliable Communication, Fault-tolerant Communication, Adaptive Communication Protocol.*

1 Introduction

Wireless mobile networks are characterized by severe constraints on resources, such as bandwidth and battery power, and by rapid fluctuations in availability of these resources; this makes it difficult for the system software to provide guaranteed quality-of-service at levels required by many distributed and collaborative applications. Further, mobility adds a new dimension to the distributed computing paradigm which has implications on specification, design, verification, and implementation of both system and application software. One of the challenging issue is to determine the interface and the guarantees system software can efficiently provide to developers of both location-independent and location-dependent (context-aware) applications on mobile networks.

In this paper, we take up the challenge of developing a reliable multicast protocol for mobile ad hoc networks (MANET) [CK87, GT95, PR97]. Reliable multicast protocol are used in various distributed applications. Further, mobile ad hoc networks are being increasingly used for military operations, law enforcement, rescue missions, virtual class rooms, and local area networks. Mobile ad hoc networks present a different network model than have been used for communication protocol design and implementation. Network reconfiguration and link failures are common occurrence in these networks. Further, bandwidth is a comparatively precious resource and needs to be conserved. Moreover, in mobile ad hoc networks, the topology of the network is dynamic and it can change at different rates at different times in different regions of the network.

Recently, several protocols for delivering multicast messages in mobile networks have been developed [AB96, CB94]. These works are based on the assumption that the node mobility does not alter the network topology, i.e., mobile hosts can directly communicate to the fixed mobile switching stations, which are connected by fixed wired network. Chiang and Gerla have presented a "best effort" multicast algorithm in [CG97]. This algorithm is based on underlying multi-cluster multi-hop packet network [GT95] and uses core based trees for multicast framework [BFC93]. This algorithm does not guarantee reliability in the sense that messages may be lost if the topology changes during multicasting of a message. This method is suitable when the rate of topology change is low. Pagani and Rossi have presented a reliable broadcast algorithm in [PR97] which is also based on underlying multi-cluster multi-hop packet network [GT95, EWB87]. The algorithm builds a dynamic forwarding tree involving all the *cluster-heads* and *gateways* in the network. Reliable transmission is guaranteed by using a message stabilization mechanism. High mobility is handled by using timeouts and acknowledgment flooding. This algorithm can be adapted in a straightforward way to perform multicast by restricting the delivery of message to only members of the multicast group within each cluster. However, this approach would be wasteful in network bandwidth since the forwarding tree involves all cluster-heads and gateways of the clusters; some or many of those clusters may not belong to the multicast group or may not contain any members of the multicast group. Hence, this algorithm is suitable for only dense multicast and is not scalable. When the multicast group contains relatively less number of hosts compared to the total number of hosts in the system, we need to have more efficient protocols to conserve network bandwidth.

In this paper, we present a reliable multicast algorithm which adjusts to the rate of topology change in the network. A core based shared multicast tree is constructed as nodes in the multicast group send JOIN message which grafts branches from multicast tress to the nodes. Whenever a sender wants to multicast a message to members of group gid it sends a MULTICAST message to the core node of group gid which initiates the dissemination of the message down the multicast tree. The acknowledgments from the multicast group nodes flows in the reverse direction towards the core node. Acknowledgment aggregation is used to reduce the bandwidth wastage. A message stabilizes when the core node receives acknowledgments from all the multicast group nodes. This knowledge about message stabilization is piggy backed in subsequent multicast messages. A node delivers and deletes its local copy of the message upon gaining the knowledge about message stabilization.

A multicast tree may get fragmented due to node movements. A notion of *forwarding region* is introduced which is used to glue together fragments of a multicast trees. The gluing process involves flooding of the forwarding region of the nodes which witness topology change due to node movements. Delivery of multicast messages to mobile nodes is expedited through i) pushing of the message by witness nodes in their forwarding regions and ii) pulling of messages by a mobile node during (re)joining process. Our protocol is reliable, i.e., it guarantees delivery of the multicast message in presence of node mobility (by using message stabilization) and at the same time preserves network bandwidth by limiting message flooding only to the parts of the network affected by host movements. Furthermore, our protocol is more efficient in conserving network bandwidth especially when the multicast groups are small in size and the multicast group members are localized to a certain region of the network; the implicitly formed forwarding regions can detect the topology changes during multicast and limits flooding only to the part of the network that is affected by the change. As opposed to Pagani and Rossi's protocol [PR97], our protocol not only adapts to topology changes but also to the distribution of the multicast group members to minimize the use of system resources.

2 System Model

The mobile multi-hop network consists of n identical mobile hosts (nodes) with unique ids $p_1, \ldots, p_n$. These mobile hosts communicate among each other via a packet radio network. When a node transmits (broadcasts) a message, the nodes in the *coverage area* of the sender can simultaneously receive the message. A node p_i is called a *neighbor* of node p_j in the network if p_j is in the *coverage area* of node p_i (this relationship is symmetric and time varying since the nodes can and do move). At any given time a node p_i can correctly receive a message from one of its neighbors, say p_j, iff p_j is the only neighbor of p_i transmitting at that time. We make the following assumptions about the system.

- A data link layer protocol at each node p_i maintains the identities of its neighbors in some list $neighbors(p_i)$; this data link protocol also resolves any contention for the shared medium by supporting logical links between neighbors and ensures that a message sent over a correct (or functioning) logical link is correctly received by the node at the other end of that link. The link layer protocol informs the upper layer of any creation/deletion of logical links.

- A routing protocol is used to send unicast messages between two non neighboring nodes in the network. These unicast messages are reliably delivered, i.e., a message sent by a node p_i to node p_j will be eventually delivered to p_j.

- The topology of the ad-hoc (dynamic or time varying) network is modeled by a (undirected) graph $G = (V, E)$, where V is the set of nodes and E is the set of logical links between neighboring nodes[1]. Since the nodes are mobile, the network topology graph changes with time. We assume that no node leaves the system and no new node joins the system; we also assume that transient link failures are handled by the link layer protocol by using timeouts, retransmission, and per hop acknowledgment. Thus, the network graph has always the same node set but different edge sets.

[1] We assume for simplicity that all links are bidirectional.

We also make the following assumptions about rate of change of topology. The change in topology is arbitrary but satisfies the **Liveness** property: if there are pending messages for a processor p, then the network eventually remains constant long enough so that p receives at least one of these messages and acknowledges the receipt. If liveness is not guaranteed, it is possible that message m destined for processor p can never be delivered to p or is delivered but p cannot acknowledge the receipt; this may prevent the termination of any protocol. Further, we assume that all types of failures in the system are temporary. Thus, the model captures the mobility and communication failures in mobile radio networks.

3 Multicast Protocol

3.1 Basic Idea and Definitions

Our protocol for reliable multicast uses the concept of the core-based tree protocol developed in [CG97]. Each multicast group has a unique multicast group identifier gid and has a unique $core$ node. We assume that each node u knows the identity of the core node of each multicast group u belongs to[2]. Reliable multicast consists of two components: (1) delivery of message m to processors in the multicast group, and (2) delivery of acknowledgment (receipt confirmation) of m to the source and the destinations.

3.1.1 Delivery of Multicast Messages

A multicast message is delivered using a shared multicast tree rooted at the core node of the multicast group. The multicast tree is constructed in a distributed manner. Each node in the multicast group sends a JOIN message towards the core node. The JOIN message is forwarded until it reaches the core node or some node which is already part the multicast tree.

The source node sends the multicast message to the core node of the multicast group. The core node assigns a sequence number to the message and initiates multicasting by sending them to its children in the multicast tree which in turn send the message to their children and so on. If the portion of the ad-hoc network which contains the multicast tree does not experience any topology change, then the acknowledgment messages travel up the tree links towards the core node. *Once the core node receives all the acknowledgment messages, it records that the message has stabilized.* This information is then diffused to all the nodes in the multicast tree and the sender (source), which can then delete the message from their memory.

[2] If a node does not have this information, it can get it from a multicast server. In this paper, we do not address the problem of establishment and maintenance of a multicast group; further, the multicast groups are assumed to be open in the sense that any node in the system can multicast a message to any multicast group.

Example 1: Consider the ad-hoc network shown in Figure 1(a). The multicast group $M = \{3, 5, 6, 12, 15\}$ has 5 nodes; nodes in a multicast group are also called *multicast nodes*. Let node 3 be the core node. Figure 1(a) shows three JOIN messages and the paths they travel. The first JOIN message from node 5 establishes a path from 3 to node 5. The second JOIN message from node 12 to node 3 is intercepted at node 5 since node 5 has already joined the multicast tree. Further, node 8 and 13 also become part of the multicast tree. Nodes belonging to the multicast tree but not belonging to the multicast group are called *forwarding nodes*. The third JOIN message establishes the path from node 15 to node 3. Node 6 need not send separate JOIN message since it is already in the multicast tree. The resulting multicast tree is shown in Figure 1(b). □

A multicast tree can get disconnected due to mobility of the nodes. For example, Figure 1(b) shows a multicast tree. Suppose node 3 which is the root of the multicast tree moves as shown in Figure 1(c). After the move, the multicast tree T has been fragmented into three discontented components: node 3 by itself, the subtree rooted at node 4, and the subtree rooted at node 5.

One approach to do multicasting, when a multicast tree is disconnected due to node movements, is to reconstruct the multicast tree and this may need be done repeatedly especially when the nodes move frequently. We develop a new mechanism of repairing the multicasting tree from its fragments by introducing the concept of the *forwarding region* of a node. Informally, a forwarding region of a multicast tree node u is the maximal region (subgraph) of the topology graph around u which consists of only non-tree nodes. In order to flood its forwarding region with message m a node u simply broadcasts m to all its neighbors. Any node which receives m forwards it only if it is not a multicast tree node. Note that the forwarding regions are implicit and dynamic in nature and hence adapt to continuous topology changes. The idea is to flood the message when the multicasting tree is fragmented using the forwarding regions to restrict the flooding to regions where the topology has changed. We now formally define the notion of forwarding regions.

Definition 1 *Given a network graph G and a multicast tree T of G, a* **forwarding region** *of an arbitrary node $u \in G$ with respect to T, is a subgraph $F(u, T)$ of G such that:*

1. *A node v is in $F(n, T)$ iff there exists a path from u to v in G with intermediate nodes only in $G - T$.*
2. *An edge $e = (v, w)$ $(v, w \in F(u, T)$ and $(v, w) \in G)$ is in $F(n, T)$ iff one of the following two conditions is true*

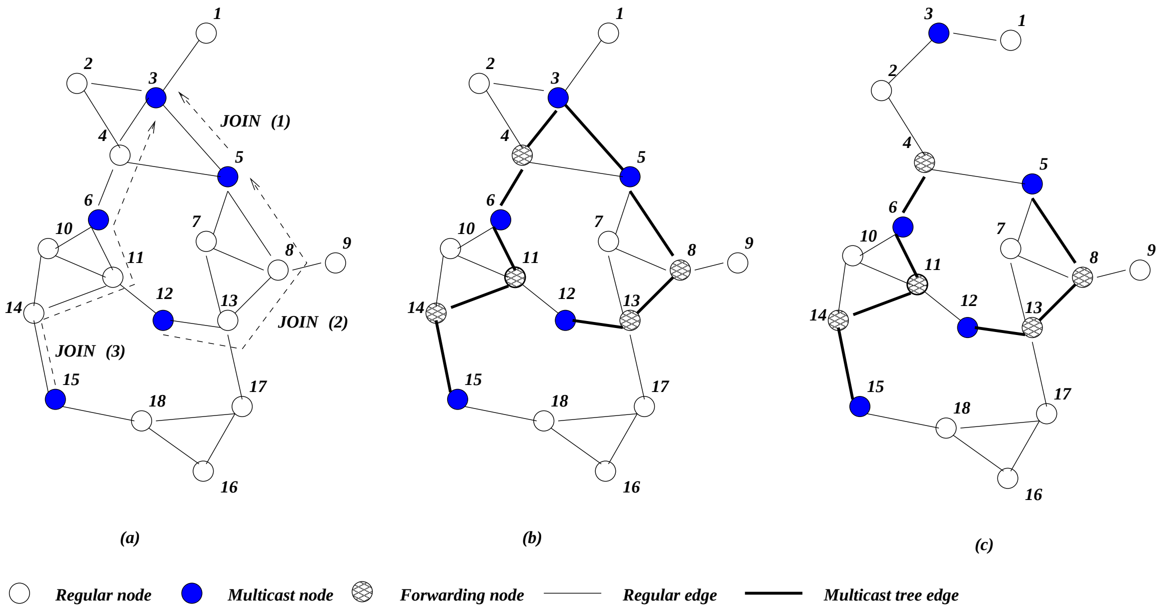

Figure 1. Construction of a multicast tree and its fragmentation due to node movement.

(a) $v \in (G - T) \vee w \in (G - T)$.

(b) $v = u \vee w = u$.

The node u is called the **center** *of forwarding region $F(u,T)$. Any node v in $F(u,T)$ which has an incident edge (v,w) of G such that w does not belong to $F(n,T)$ is called a* **gateway node** *of $F(u,T)$. The set of all gateway nodes of $F(n,T)$ is called the* **boundary** *of $F(n,T)$.*

Example 2: Consider the network graph G and the multicast tree T shown in Figure 2(a). The forwarding regions of nodes nodes 3, 4, and 5, $F(3,T)$, $F(4,T)$, and $F(5,T)$ respectively, are shown. □

Property 1 *Each gateway node of a forwarding region $F(n,T)$ is a node of the multicast tree T.*

We are interested in the forwarding regions of the nodes belonging to the multicast tree. As we noted earlier, any node belonging to the multicast tree either belongs to the multicast group (i.e., is a multicast node), or is a forwarding node. We observe the following.

Property 2 *For any connected graph G and a non null tree T,*

$$G = \bigcup_{u \in T} F(u,T)$$

i.e., the forwarding regions of the nodes in the multicast tree together cover all the nodes in the network graph G.

Remark 1 *The forwarding region is implicit in the way a message is forwarded and need not be explicitly determined. Further, the forwarding regions act as "glue" to link different fragments of a multicast tree together (when the multicast tree gets disconnected due to node movements).*

Example 3: Assume that node 3 is the core node of the multicast group consisting of nodes {3,5,6,12,15}. Consider that in the initial multicast tree node 3 has node 4 (as a forwarding node) and node 5 as its children. Now suppose that node 3 moves such that it gets disconnected from node 4 and 5 as is shown in Figure 2(a). Assume that just after moving node 3 receives a request for multicasting a message m. Since it detects that the multicast tree is disconnected, it floods its forwarding region with the message m. The message m reaches node 4 since it is a gateway node for forwarding region of node 3. Node 4 (it has also detected the move of node 3 due to its link to node 3 going down) also floods the message in its forwarding region $F(4,T)$. Due to flooding of $F(4,T)$, node 5 and 6 receive the multicast message. Similar to node 4, node 5 also floods its forwarding region. However, since node 6 did not witness any change in topology it uses the multicast subtree rooted at it to forward the message m. Similar to node 6, node 13 also forwards the multicast message rooted at itself. Thus, all the nodes in the multicast tree get the multicast message either via flooding of message m in the forwarding region of a node or via links in the multicast tree. □

Remark 2 *Flooding is restricted to the vicinity of the region where topology is changed, and the fragments of the multicast tree are used in the parts of the network where the topology is not changed. Thus, as long as the topology re-*

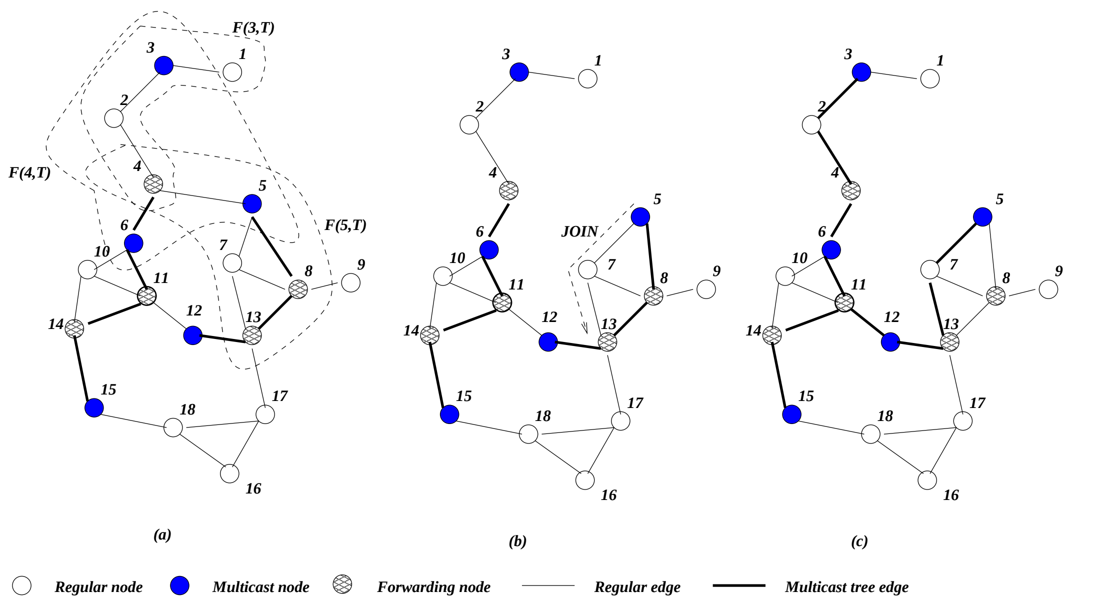

Figure 2. Using combination of multicast tree and forwarding regions for multicasting messages in presence of topology change.

mains connected, forwarding regions can be used for multicasting limiting the need for flooding to a minimum.

Property 3 *(***Gluing Property***) Let T be a multicast tree in a network G. Consider nodes u and v in T such that u is parent of v in T. Then, as long as topology remains connected there is a sequence of nodes $< u_1, u_2, \ldots, u_k >$ such that: $u_1 = u$, $u_k = v$, and $u_{i+1} \in F(u_i, T), 1 \leq i < k$ and u_{i+1} is a gateway node of $F(u_i, T), 1 \leq i < k - 1$.*

It is easy to observe that the message would be flooded in the region which is union of all the forwarding regions of the nodes which witness change in topology.

Definition 2 *(***Witness Nodes***) Suppose the network topology changes from G_1 to G_2 due to node movements (note G_1 and G_2 both have the same set of nodes). The witness nodes in G_2 with respect to G_1, denoted as $W(G_1, G_2)$, are the nodes that do not have same set of neighbors in G_2 as in G_1. Obviously, $W(G_1, G_2) = W(G_2, G_1)$.*

Definition 3 *(***Significant Witness Nodes***) Suppose the network topology changes from G_1 to G_2 due to node movements (note G_1 and G_2 both have the same set of nodes) and T is a multicast tree in G_1. The significant witness nodes in G_2 with respect to G_1 and T, denoted as $SW(G_1, T, G_2)$, is a subset of witness nodes $W(G_1, G_2)$ such that a node u is in $SW(G_1, T, G_2)$ iff no neighbor of u in T is its neighbor in G_2.*

Using the significant witness nodes to do the flooding in the event of a topology change, we can limit the number of nodes which receive the multicast message via flooding. Thus, use of forwarding regions to perform "controlled flooding" of the network in the event of topology change can lead to significant savings in bandwidth used for multicasting.

3.1.2 Avoiding Loops in Multicast Tree due to Rejoining

Although forwarding regions can help glue together some portion of the fragmented multicast tree, other parts of the tree may become unusable due to change in topology. For example, consider the situation in Figure 2(b). It shows a situation in which node 5's parent node has moved away leaving node 5 with no parent node. Now, suppose node 5 sends a JOIN message to rejoin the multicast group. This can cause a loop to be formed as shown in Figure 2(b). In order to avoid such loops being formed, a significant witness node, if it does not receive any flooding message within time t_p after detecting a topology change, sends a REJOIN message along the old tree fragment. The effect of this message is that the old tree fragment is destroyed and each multicast node in that subtree sends a JOIN message to the core node. Figure 2(c) shows one of the possible tree that may result.

3.1.3 Delivery of Acknowledgments

The acknowledgment messages travel up the multicast tree towards the core node. In order to reduce the bandwidth consumed for acknowledgment delivery, the acknowledgments from individual nodes are combined together whenever possible. A node does this by holding back an acknowledgment in a acknowledgment queue for t_a time before forwarding. If another acknowledgment for the same message arrives within this wait period they are combined together. In case the multicast tree gets disconnected, the acknowledgment is sent directly to the core node.

3.1.4 Broadcast of Message Stabilization

Once the core node has received acknowledgments from each node in the multicast group, it records that information and informs the sender (source) of the multicast message. Further, the information about message stability is piggy backed on any future multicast messages. Once a node comes to know that a message has stabilized, it deletes the copy of that message from its memory. This guarantees only the members of multicast group get to know that a message has stabilized. However, the message is also buffered in the forwarding nodes of the multicast tree, who are not multicast nodes. It can so happen that such a node moves and becomes a non tree node and remain so for a long time. Some of the ways to handle this situation are:

- Delete the message some time after becoming a non-tree node.
- The core node broadcasts the information about message stabilization.

The first option saves the bandwidth whereas the second option reduces wastage of node memory. However, the second option is useful to reduce multicast latency. A compromise between the two options is that the core node periodically broadcasts the information about message stabilization.

3.2 Data Structures

We assume that there are maximum of n nodes and g multicast groups in the system. Each node in the system maintains the following local variables and data structures:

- $\boldsymbol{Is\_member_i}[\mathbf{1}:\boldsymbol{g}]$: is a boolean array. The value $Is\_member_i[gid]$ is `true` if node i is a member of group gid, otherwise it is `false`.
- $\boldsymbol{Size_i}[\mathbf{1}:\boldsymbol{g}]$: is an integer array. The value $Size_i[gid]$ is the number of members in the multicast group gid. (Needed only in the core node of a multicast group).
- $\boldsymbol{Core_i}[\mathbf{1}:\boldsymbol{g}]$: is an array of node ids. The value of $Core_i[gid]$ is the node id for core node of the multicast group gid.
- $\boldsymbol{Last_i}[\mathbf{1..}\boldsymbol{m}]$: is an array of integers. $Last[gid] = d$ indicates that node i has received all the messages with sequence number less than or equal to d sent to the multicast group gid.
- $\boldsymbol{Parent_i}[\mathbf{1}:\boldsymbol{g}]$: is an array of node ids. $Parent_i[gid] = k$ if node k is the parent of node i in the multicast tree for group gid, otherwise $Parent_i[gid] = NULL$.
- $\boldsymbol{Children_i}[\mathbf{1}:\boldsymbol{g}]$: is an array of sets of node ids. $Children_i[gid]$ is the set of node ids of children of node i in the multicast tree for group gid.
- $\boldsymbol{Stable_i}[\mathbf{1}:\boldsymbol{g}]$: is an array of integers. The value of $Stable_i[gid]$ is the sequence number of the last multicast message which node i knows to be stable for group gid.
- $\boldsymbol{Unstable_i}[\mathbf{1}:\boldsymbol{g}]$: is an array of ordered list of tagged messages. $Unstable_i[gid]$ is the list of messages received by node i for multicast group gid whose copies node i has to maintain till it stabilizes and is deliverable. A stabilized message with sequence number k for group gid is deliverable only when all message with sequence number less than k have stabilized and have been delivered. This list is ordered according to the multicast sequence number of the messages. The multicast sequence number for a message in $Unstable_i[gid]$ is greater than $Stable_i[gid]$. The tag of a message can be `stable` or `unstable` indicating whether or not node i knows that the message has stabilized. A stable message is in this list only if there is an unstable message with a smaller sequence number.
- $\boldsymbol{AwaitQ_i}[\mathbf{1}:\boldsymbol{g}]$: is an array of ordered list of acknowledgment bit vectors. $AwaitQ_i[gid]$ contains ordered list of acknowledgment bit vectors for group gid. Each of these lists is ordered according to the multicast sequence number of the messages. Node i has a daemon, called process_acks(), which periodically empties all the $AwaitQ_i[]$'s by sending out the acknowledgment messages up the corresponding multicast tree or directly to the corresponding core node. Further, if the message m for group gid has stabilized, i.e., number of one bits in the acknowledgment bit vector for m is equal to $Size_i[gid]$ then that message is marked `stable`. Further, $Stable_i[gid]$ is appropriately updated.
- $\boldsymbol{DeliveryQ_i}$: is a FIFO queue of deliverable messages.

```
Procedure process_acks()
begin
    while(true)
        wait(t_a);
        for gid = 1 to g do
            for each (s, bv) ∈ AwaitQ_i[gid] in increasing order of s do
                if(Parent_i[gid] ≠ NULL)then
                    send MULTICAST_ACK(gid, i, s, bv, Stable_i[gid]) to Parent_i[gid];
                    delete (s, bv) from AwaitQ_i[gid];
                else
                    if(Core_i[gid] = i)
                        if(((s, unstable, m) ∈ Unstable_i[gid]) ∧ (number_of_ones_in(bv) = Size_i[gid])) then
                            m.tag = stable; /* m.tag is the stable/unstable tag of m. */
                    else
                        send MULTICAST_ACK(gid, i, s, bv, Stable_i[gid]) to Core_i[gid];
                        delete (s, bv) from AwaitQ_i[gid];
            if(Core_i[gid] = i) then /* determine stable messages */
                seq = Stable_i[gid] + 1;
                while ((seq, stable, m) ∈ Unstable_i[gid]) do seq = seq + 1;
                seq = seq − 1;
                if(seq > Stable_i[gid]) then process_stable_msgs(gid, seq);
    end while
end
```

Figure 3. Procedure process_acks().

3.3 Message Types

- **JOIN**($gid, i, k, stable, last$) — indicates that the sending node i is requesting to join the multicast group gid; k is the id of the node which initiated this JOIN message; $stable$ is the maximum value of $Stable_l[gid]$, where l belongs to the set of multicast group nodes in the path from node k to node i traversed by this JOIN message; $last$ is maximum value of of $Last_l[gid]$, , where l belongs to the set of multicast group nodes in the path from node k to node i traversed by this JOIN message.

- **JOIN_ACK**($gid, i, j, stable_i$) — indicates that node i will forward any message multicast to group gid it receives to the receiver of this message; node j is the intended receiver of this JOIN_ACK message; $stable_i$ is the value of $Stable_i[gid]$ just before sending this JOIN message.

- **REJOIN**($gid, stable$) — indicates that the receiver of this message should dismantle the current multicast tree rooted at it and if necessary send JOIN message to establish a new path from core node to itself.

- **MULTICAST**($gid, i, seq, m, stable_i$) — indicates that the sender i wants to multicast message m to multicast group gid; seq is the sequence number of this message in the group gid.

- **MULTICAST_ACK**($gid, i, seq, ack\_bv_i, stable_i$) — indicates that the sender i has received multicast seq-th message sent to multicast group gid; ack_bv_i is the acknowledgment bit vector indicating the set of node which have received the seq-th message as known to node i. Evernode runs a demon called **process_ack**() which periodically sends MULTICAST_ACK message including the bit-vectors which aggregate the acknowledgements in the acknowledgment queues. The pseudo-code for process_acks() is given in Figure 3. Procedure process_acks() uses a auxiliary procedure called **process_stable_msgs**() which is given in Figure 4.

- **PULL**($gid, i, stable_i, last_i$) — indicates that node i wants copy of messages with sequence number greater than $last_i$ for multicast group gid.

- **PUSH**($gid, i, seq, m, stable_i$) — indicates that sender i wants to multicast the message m in the push mode to the members of multicast group gid; the sequence number of the message m in the multicast group is seq. As the message is pushed in the forwarding region of a node, say u, a *forwarding tree* rooted at u is dynamically constructed.

- **PUSH_ACK**(gid, i, p) — indicates that sender i has chosen node p as its parent in the *forwarding tree* of group gid.
- **QUIT**($gid, i, stable_i$) — indicates that node i does not need multicast message for group gid anymore.

3.4 System Primitives

The following primitives are used in the pseudo code given in the next subsection. They are used to simplify the presentation of the algorithm and their functions are described below.

- **send** msg to j: is a non-blocking reliable unicast primitive supported by the underlying routing algorithm which sends message msg to node j.
- **mids**(Q, d): returns a message sequence number, $l \geq d$, such that if $l > d$ then all the messages with sequence numbers $d+1, \ldots, l$ are in the message queue Q.
- **wait**(t): suspends the calling process for t seconds.
- **transfer**(Q_1, Q_2, d): transfers all the messages with sequence number less than equal to d from queue Q_1 to queue Q_2.
- **delete**(Q, d): deletes all the messages with sequence number less than equal to d from queue Q.

3.5 Events

Node i wants to Multicast a Message: When a node wants to multicast message m to multicast group gid, it executes the following command: **send** MULTICAST($gid, i, 0, m, Stable_i[gid]$) to $Core_i[gid]$.

Node i receives MULTICAST message: When a node receives a MULTICAST($gid, j, SEQ, MSG, STABLE$) message, it used the SEQ number in the message to filter out duplicate receipt of a MULTICAST message. Further, if node i is the core node of group gid (i.e. $Core_i[gid] = i$), it assigns the message a new sequence number, $Last[i] + 1$. Otherwise, node i preserves the message sequence number SEQ. The MULTICAST message is then propagated down the multicast tree by sending it to all the children of node i. This is in fact just a single broadcast in the coverage area

```
Procedure process_stable_msgs(gid, STABLE)
begin
    if(Stable_i[gid] < STABLE)
    then
        if(Is_member[gid]) then
            /* transfer deliverable messages */
            transfer(Unstable_i[gid],
                            DeliverQ_i, STABLE);
        else
            delete(Unstable_i[gid], STABLE);
        Stable_i[gid] = STABLE;
        if(Last_i[gid] < Stable_i[gid]) then
            /* ensures Last_i[gid] >= Stable_i[gid] */
            Last_i[gid] = Stable_i[gid];
end
```

Figure 4. Procedure process_stable_msgs().

of node i. Further, the message is entered along with its sequence number in the $Unstable_i[gid]$ as an `unstable` message and an acknowledgment bit vector for this message is placed in the queue $AwaitQ_i[gid]$. Finally, $Last_i[gid]$ is updated using the $mids()$ function which is described in Section 3.4. The udates $Last_i[gid]$ to the last consecutive message sequence number received by node i.

Node i receives MULTICAST_ACK message: When a node receives a MULTICAST_ACK($gid, j, SEQ, BV, STABLE$) message, it first adds the acknowledgment bit vector, BV, in the message to $AwaitQ_i[gid]$. The acknowledgments for receipt of message m travels up the tree. A node periodically combines together all the MULTICAST_ACK it receives for a particular message m into one MULTICAST_ACK and sends it to its parent node in the multicast tree. The core node upon receiving acknowledgments from all the nodes in the multicast group marks message m as `stable`. This information is diffused using the stable field of subsequent MULTICAST messages and by periodically broadcasting this information.

Receiving a JOIN($gid, j, k, STABLE, LAST$) Message: The JOIN message works in conjunction with the JOIN_ACK message to establish a path from the multicast tree to the node requesting to JOIN the multicast tree. The JOIN message initiated by a multicast node, say u, is processed by each node in the path from u to a current node in the multicast tree. The path is progressively determined hop-by-hop: the current node (i.e. node i) queries the underlying routing protocol for the next hop node in the path to the $Core_i[gid]$ node from itself. As the JOIN message progress towards the $Core_i[gid]$ node, a tree path in the reverse direction is established by receiver of the JOIN message (i.e. node i) adding the node from which it received the JOIN message as on of its children.

A node sends a JOIN message when it moves to a new location in the network or when it gets (re)connected to the network. Due to mobility or disconnection a node may have not received some of the data messages for the multicast groups it belongs to. Hence, JOIN message is used not only for initiating setting up path for the future multicast data messages but also for sending (re-multicasting) any missing multicast data messages to the joining nodes along the new path being setup. A node checks the condition $Last_i[gid] > LAST$. If this condition holds then it indicates that node i has copies of some the data messages with sequence number greater than $LAST$ and it at least has copies of all data messages with sequence number $LAST + 1$ up to $Last_i[gid]$. Node i initiates the multicast of these messages.

A node i receiving a JOIN message can be of one of the following four types: i) a non-tree multicast group node, ii) a regular node, iii) a multicast node connected to the multicast tree, and iv) a forwarding node. The processing of a JOIN message is illustrated in Figure 5 for each of the four different receiving node types under two conditions i) $Last_i[gid] > LAST$ and ii) $Last_i[gid] \leq LAST$. For example, Figure 5(c) illustrates the case when node i is a regular node and $Last_i[gid] > LAST$. In this case, node i becomes a forwarding node, sends JOIN message to next hop node in the path to $Core_i[gid]$, node q, send JOIN_ACK message to next hop node to the initiating node k, i.e. node l, and sends MULTICAST message for each data message m multicast to group gid for which $LAST < m.seq \leq Last_i[gid]$ to both node j and node l.

Receiving a PULL Message: As shown in Figure 5(f) and (h), a node i which is part of the existing multicast tree sends a PULL message to one of a neighbor q if it learns through a JOIN message that it is missing some of the messages (i.e. when $Last_i[gid] < LAST$). The pull message travels hop-by-hop up the tree links towards the core node in search of node which can re-initiate multicasting of the data messages which the initiator of JOIN message is missing. Note that if node i (in Figure 5(f) and (h)) is a multicast group node then it sends $Last_i[gid]$ in its PULL message otherwise it propagates the $LAST$ value it obtained from the incoming JOIN/PULL message. This ensures that only those messages which have not been received by a multicast group node are re-transmitted.

Receiving a JOIN_ACK Message: As illustrated in Figure 5, a node i receiving a JOIN message sends out a JOIN_ACK message to the next node in the path to the initiator of the JOIN message, unless the node is a regular node for which $Last_i[gid] \leq LAST$. Similar to JOIN message, the JOIN_ACK message also establishes path from the initiator node and the final recipient of the JOIN_ACK message. The rationale for having both JOIN and JOIN_ACK message establish path to multicast tree is as follows. Suppose a node k sends out a JOIN message. In case the topology does not change, the JOIN_ACK will follow the reverse path of the JOIN message (assuming symmetric links). In this case the JOIN_ACK message just helps to propagate the message stabilization information to the nodes in the path. However, in the presence of node mobility the JOIN_ACK message helps to expedite the setting up of paths to mobile nodes. For example, suppose node k moves after sending out the JOIN message. In this case the JOIN_ACK message will be routed to the new location of node k. The portion of the path setup by the JOIN message will be torn down by the QUIT message sent out by a forwarding node which becomes a leaf node after node k moves. If a node which is already in the tree for group gid, receives a group gid's JOIN_ACK message for node k it does two things: i) sends QUIT message up the tree and ii) forwards JOIN_ACK message towards node k. This ensure that every node has a single parent node, i.e., paths established by JOIN_ACK messages in fact construct a tree.

Receiving a QUIT Message: A QUIT message is initiated by a forwarding node in the tree which becomes a leaf node due to node movements. The QUIT message travels hop-by-hop up the multicast tree branch tearing down the multicast path up to the first tree node which is either a non-leaf forwarding node or is a multicast group node. A node receiving a QUIT message deletes the node from which it receives the QUIT message from its children list. If the children list becomes empty and it is a forwarding node then it sends QUIT message to its parent in the multicast tree.

Receiving a PUSH Message: Upon receiving a PUSH message from a neighbor node, a regular node in the tree sends out the PUSH message in the forward direction, i.e., on all the links on which it has not received PUSH m message. Further, the PUSH message also builds a *forwarding tree*. This is done by each node selecting one of the nodes from which it received PUSH message as the parent node in the forwarding tree. All the children node of the node k which receive the PUSH message send acknowledgment message up the forwarding tree using PUSH_ACK message.

A boundary node in the forwarding region of k which is not a child of k, say node b, ignores the PUSH message if it has already received a copy of that message along the multicast tree, otherwise node b treats message m in a

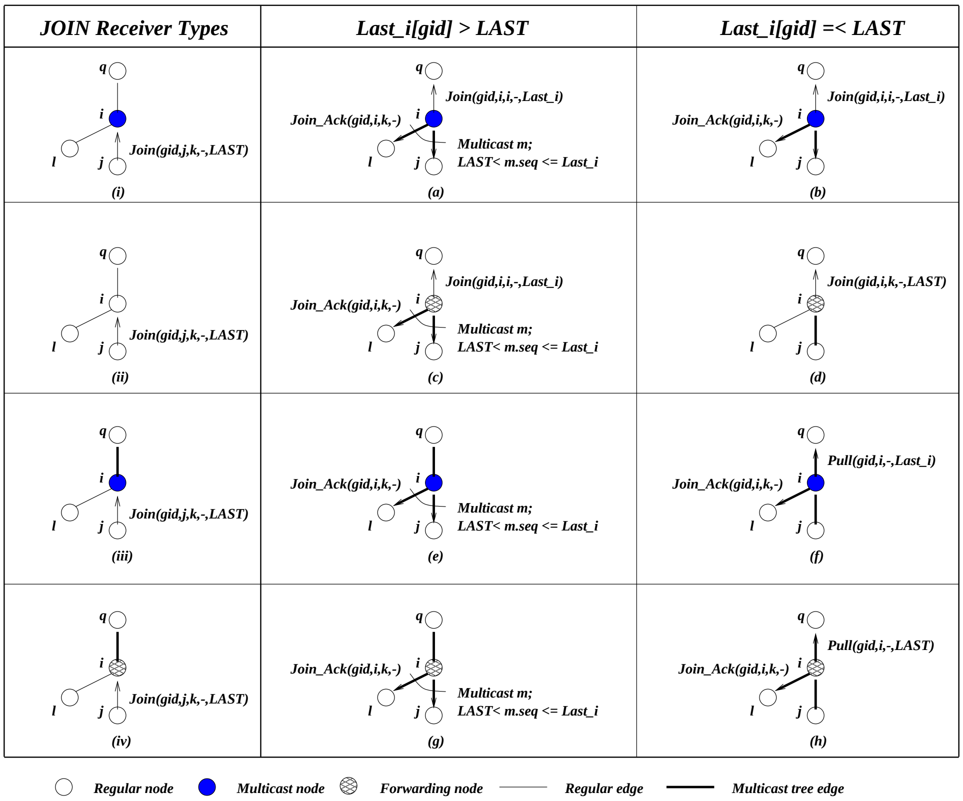

Figure 5. Processing of JOIN message.

manner similar to receiving m via MULTICAST message, i.e., node b PUSH's m to all its children. Some of the branches in the forwarding tree constructed during flooding of forwarding region of node k may end in non multicast nodes. Unless these nodes receive a JOIN message from a multicast group node, they will eventually send QUIT message up the forwarding tree links.

Receiving a PUSH_ACK Message: On receiving PUSH_ACK message from neighbor node j, node i can determine whether node i is parent of node j in the forwarding tree.

Receiving a REJOIN Message: A child node in a multicast tree which does not receive PUSH message after a link to the parent node has failed within a timeout period, t_p, sends out REJOIN message down the multicast subtree rooted at it. This dismantles the subtree and cause all the multicast group nodes in the subtree to resends JOIN message.

New link to node j: When a new link is established to node j, node i simply adds j to $Neighbor_i$ set and sends JOIN message for each multicast group to which it belongs but is not connected to that groups multicast tree.

Failure of link to node j: Node movements are detected via link failures. When a non-leaf node in a multicast tree moves, many of the links Suppose a link to a node, say j, which is the only child of node i in the multicast tree for group gid fails. In this case, node i floods its forwarding region with all unstable multicast messages. This is done using the PUSH message. These message reestablishes paths from node i to some nodes. A child node of k which does not receive PUSH message after a link to the parent node has failed within a timeout period, t_p, will send REJOIN message down the multicast subtree rooted at it. This will dismantle this subtree and cause all the multicast group nodes in the subtree to re-send JOIN message.

4 Correctness Proof

We state the following lemmas (some without proof).

Lemma 1 *The protocol solves the Reliable Multicast Problem in the case of interconnected static topology with transient link failures.*

Proof : The unicast sends are assumed to be reliable. Hence, all protocol messages sent will eventually be received. If nodes do not move then once all the nodes have joined the multicast tree, all the messages which are multicast thereafter will eventually be received since unicast messages are reliable. Hence, we need to only show that the messages that are multicast before all the nodes in the system have joined the multicast tree will be eventually received. Note that every node belonging to the multicast group will eventually send a JOIN message with $LAST = 0$. A multicast message cannot stabilize before every node has joined the multicast tree, received this this message and acknowledged the message. Assume that s messages were multicast before a node i joined the tree. Consider the first node in the path from node i to the multicast tree which has the message with sequence number 1, say node l. Node l will re-multicast this message along the path leading to node i. Hence node i will eventually receive message with sequence number 1. Suppose node l did not also have copy of message with sequence number 2, i.e. $Last_i[gid] = 1$. Node l will either send a JOIN message or a PULL message with $LAST = 1$. This will initiate the re-multicasting of message with sequence number 2 along the path leading to node l and so also along the path leading to node i. Hence, it can be shown that node i will eventually receive all the messages with sequence number up to s. The acknowledgments for these messages from node i will travel up the multicast tree or directly to core node. Eventually these acknowledgments will reach the core node, since i) unicast messages are assumed to be reliable and ii) each node periodically processes its $AwaitQ[]$ and sends out any pending acks. □

Lemma 2 *A multicast message is eventually delivered to every member of a multicast group.*

Proof : Initially the sender has the message m which it wants to multicast to group gid. Since unicast messages are reliable the MULTICAST message containing m will eventually reach the core node of group gid. The core node assigns this message a sequence number, say k. At any time thereafter, let the subset of nodes in the system which have a copy of message # k be denoted as $Have_k$. Initially, the sender and the core node of the group are in the set. We show that the protocol ensures that eventually all the members of the multicast group are in the $Have_k$ set. Note that irrespective of whether or not a node belongs to the multicast group gid, once a node receives a copy of a message for group gid it retains that copy until the message stabilizes. Consider a node u in the multicast group gid which is not in $Have_k$ set.

- If node u is connected to the multicast tree (long enough) then it will receive message # k via the multicast tree; including node u in the $Have_k$ set.
- If the path from u to the tree gets disconnected due to failure of link, say (p, c), then either node p will flood its forwarding region and node c will receive the message m (increasing the size of $Have_k$) or node c will send a REJOIN message down the tree. Eventually either node u will receive REJOIN message or it will itself move. In either case it will send a JOIN message towards the core node. Consider the path this JOIN message takes. Along this path consider the closet node to node u which does not have message # k. This node will receive message # k from its predecessor node, say v, in the tree since v will re-multicast the message. Hence, increasing the size of $Have_k$. Since, there are finite number of nodes in the system, $Have_k$ cannot grow indefinitely. Hence, eventually all the nodes belonging to group gid will be in the $Have\_k$ set and will have a copy of the message in $Unstable_i[gid]$ list.

□

Lemma 3 *The core node of group gid eventually receives acknowledgment for message m from all the members of the multicast group gid.*

Lemma 4 *Every multicast message eventually stabilizes.*

Lemma 5 *After a (data) message stabilizes, eventually no protocol messages are sent concerning m and the copy of m is deleted from every processor's memory.*

Based on the preceding lemmas the following theorem can be established:

Theorem 1 *The protocol solves the Reliable Multicast Problem in the Multihop Adhoc Network which satisfies the* **Liveliness** *property.*

Proof : Any protocol for a reliable multicast in any ad-hoc network satisfies two requirements: 1) Validity and Agreement: if a processor multicasts a message m to group G then all the processors in group G eventually deliver m, and 2) Integrity: for each message m, every processor in G delivers m at most once, and only if m has been multicast to group G by some processor.

Liveness property guarantees that a reliable unicast primitive can be implemented. Further, Lemma 2-4 together establish the Validity and Agreement requirement for a reliable multicast protocol. The Integrity requirement is satisfied since i) reliable unicast protocol guarantees that the message sent by sender to the core node of group gid will be delivered only once, ii) the core node after receiving that message assigns it a unique sequence number, say k, and ii) a message with sequence number k is transferred from $Unstable_i[gid]$ to $DeliveryQ_i$ only once when the first time $Stable_i[gid]$ becomes greater than k. □

5 Conclusions

We have proposed a protocol for performing reliable multicast in mobile multi-hop radio networks. The protocol supports multicasting to a set of predefined multicast groups. The protocol uses a shared core based multicast tree and it is independent of the underlying unicast routing protocol. A notion of forwarding region is introduced which is used to limit the flooding of message (if and when needed) only to the vicinity of the nodes which witness topology change due to host mobility. Our protocol has two distinct advantages: (1) it is reliable; it guarantees message delivery in the event of topology change during multicast; the multicast protocol of [CG97] does not have this property; (2) our protocol is more efficient in conserving network bandwidth especially when the multicast groups are small in size and the multicast group members are localized to a certain region of the network; the implicitly formed forwarding regions can detect the topology changes during multicast and limits flooding only to the part of the network that is affected by the change. As opposed to Pagani and Rossi's protocol [PR97], our protocol not only adapts to topology changes but also to the distribution of the multicast group members to minimize the use of system resources.

Although we have assumed predefined multicast groups in this work, we believe the protocol can be easily extended to handle dynamic multicast groups. In this paper we have not addressed the problem of how to perform reliable multicast in the presence of node failures in mobile ad hoc networks. A reliable core tree based multicasting in presence of node failures may need to use multiple core nodes to make the protocol resilient to core node failures. We plan to address these issues in our future work.

Acknowledgements

We thank the reviewers for their insightful comments which has helped us to improve the quality of this paper.

References

[AB96] A. Acharya and B. R. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. *ACM/Baltzer Journal of Mobile Networks and Applications*, 1:199–219, 1996.

[BFC93] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (cbt): an architecture for scalable inter domain multicast routing. In *SIGCOMM*, pages 85–95, ACM, 1993.

[CB94] K. Chao and K. P. Birman. A group communication approach for mobile communication. In *Proc. Mobile Computing Workshop*, Santa Cruz, CA, December 1994.

[CG97] C. C. Chiang and M. Gerla. Routing and multicast in multihop, mobile wireless networks. In *Proceedings of ICUPC '97*, October 1997.

[CK87] I. Chlamtac and S. Kutten. Tree based broadcasting in multihop radio networks. *IEEE Transactions on Computers*, C-36(10), October 1987.

[EWB87] A. Ephremides, J. E. Wieselthier, and J. D. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of IEEE*, 1(1):56–73, 1987.

[GT95] M. Gerla and J. T. C. Tsai. Multicluster, mobile, multimedia radio network. *ACM Journal on Wireless Networks*, 1(3):255–265, 1995.

[PR97] E. Pagani and G. P. Rossi. Reliable broadcast in mobile multihop packet networks. In *Proceedings of MOBICOM 97*, pages 34–42, Budapest, Hungary, 1997.