IETF MANET Working Group   C.W. Wu
INTERNET-DRAFT   Y.C. Tay
National University of Singapore
C-K. Toh
Georgia Institute of Technology
24 November 1998

Ad hoc Multicast Routing protocol utilizing Increasing id-numberS (AMRIS) Functional Specification

<draft-ietf-manet-amris-spec-00.txt>

Status of this Memo

Abstract

This document introduces a new multicast routing protocol for use over ad hoc networks. The protocol is called AMRIS, short for Ad hoc Multicast Routing protocol utilizing Increasing id-numberS. The conceptual idea behind AMRIS is to assign every node in a multicast session with an id-number. A delivery tree rooted at a particular node called Sid joins up the nodes participating in the multicast session. The relationship between the id-numbers(and the node that owns it) and Sid is that the id-numbers increase in numerical value as they radiate from the root of the delivery tree. The significance of the Sid is that it has the smallest id-number within that multicast session. Utilizing the id-numbers, nodes are able to adapt rapidly to changes in link connectivity. Recovery messages due to link breakages are confined to the region where it occurred.

1. Introduction

Conventional multicast routing protocols for the Internet[1,2,3,4,5] break down in ad hoc networks[6] because of the dynamic nature of the network topology. The dynamically changing topology, coupled with relatively low bandwidth wireless links causes long convergence times(if they ever converge) and possibly formation of loops which rapidly consume the already limited bandwidth.

AMRIS constructs a shared delivery tree among all participant nodes. This is done by first building a DAG rooted at a special node called Sid(Sid has the smallest id within the multicast session and is generally the source node, i.e. the one that expresses the demand for a route). A delivery tree is thus formed by using a subset of the links of the DAG. Multiple senders and receivers are supported on this tree. (We are currently investigating whether the leftover links of the DAG(leftover - those not part of the delivery tree) can be maintained as secondary/backup links. Is the overhead/cost of maintaining them worthwhile vs the advantages that they can bring.)

Nodes participating in AMRIS does not require any globally consistent routing state. Permanent loops which may occur due to node movements etc will not occur. AMRIS allows nodes to recover from broken links rapidly [within one multicast beacon period]. Repairs to damaged links are performed locally without need for any central controlling node thus increasing survivability. We expect that most multicast applications are long-lived, therefore rapid route reconstruction is of greater importance compared to rapid route discovery. This may not necessarily be the case in the context of a unicast routing protocol where short route discovery times may be favoured over recovery times when the application lifespan is short.

2. Terminology

Smallest-ID node (read as Sid)
  We call the node that starts/initiates a multicast session Sid because it has the smallest msm-id compared with all other nodes for that multicast session.

Node
  A device in the ad hoc network willing to participate in the routing protocol.

Potential Parent Node (PPx)
  If a node X receive any NEW-SESSION message from PPx and PPx has a smaller msm-id than node X, then PPx is considered a Potential Parent Node of X. This information is updated in node X's

Neighbour-Status table.

Parent Node
A node X is the parent node of a node Y if node Y successfully joined the multicast session through X. Node X must have a smaller msm-id than node Y. This information is updated in both X and Y's Neighbour-Status table.

Child Node
A node Y is the child node of a node X if node Y successfully joined the multicast session through X. Node Y must have a larger msm-id than node X. This information is updated in both X and Y's Neighbour-Status table.

Bigger-id node
If a node X has a bigger multicast id then Y, then X is called Bigger-id node.

Smaller-id node
If a node X has a smaller multicast id then Y, then X is called Smaller-id node.

Multicast group
Nodes participating in multicast communication which includes the senders/sources, receivers and intermediate nodes.

Multicast Session ID (ms-id)
A unique value identifying a particular multicast session.

Multicast Session Member ID (msm-id)
An ID number that all nodes within a multicast session must have. The ID differs among nodes and generally increases in numerical value the further a nodes is from Sid.

Neighbour-Status Table
This table is a conceptual data structure that we employ to keep information regarding the neighbouring nodes and their status. Eg. Participation in which multicast sessions, etc.

Multicast Beacon
This is a periodic one-hop broadcast sent by all nodes to update neighbouring nodes about its multicast state information. Multicast state information would include the node's unique ID, msm-id as well as it's parent and child msm-ids.

3. Assumptions

We assume that all nodes within the ad hoc network are willing to participate fully in the protocol. Every node has a unique ID at either the network layer (e.g. IP address) or the link layer (e.g. MAC Address for 802.11). We assume that the underlying unicast layer provides information about symmetric links to upper layers. We assume that there is no underlying beaconing mechanism to detect link breakage. Link breakage detection is performed by making use of the periodic multicast beacon but we do not exclude using additional information such as fading link quality, positional information of nodes, etc to ensure link detection is accurate. If the underlying layers also utilize some form of beaconing, AMRIS can efficiently utilize that beaconing instead for the purpose of supporting ad hoc multicast routing.

4. Protocol Overview

Each multicast session is identified a globally unique identifier(eg class D IP address). A single node (called Sid) initiates a multicast session by broadcasting a NEW-SESSION message. Sid has the smallest msm-id. All nodes that receive the NEW-SESSION message generates their own msm-id based on the value found in the message. The new msm-id replaces the old value when the NEW-SESSION message is rebroadcasted. Nodes that receive multiple NEW-SESSION messages(for the same multicast session) will choose only one of the messages to process. The NEW-SESSION message thus travels in an expanding ring fashion outwards from Sid. In the initial phase, nodes closer to Sid will generally have smaller msm-ids than those further away.

The links of the multicast delivery tree are formed as follows:

1) If the requesting node X has a neighbouring node which is already a member of the multicast session, X will join the multicast session by sending a JOIN-REQ to that neighbour. A JOIN-ACK from that neighbour confirms that X is a registered child of the neighbour. X updates its Neighbour-Status table to indicate that that neighbour is a registered parent of X. That neighbour node also updates its own Neighbour-Status to indicate that X is a registered child.

2) If the requesting node X only has neighbouring potential parents(i.e nodes with smaller msm-ids but are not yet members of the multicast session), X will send a JOIN-REQ to one of the potential parents. The potential parent is triggered to join the multicast session when it receives the JOIN-REQ from X. It does so by sending out its own JOIN-REQ to its potential parent, if any are available. If the potential parent is Sid, it does not send out any JOIN-REQ (since it has no parents). If the parent node determines that the requesting node X is allowed to join successfully, it will send a

JOIN-ACK back the to requesting node X, otherwise it will send a JOIN-NACK.

3) If the requesting node X does not have any neighbouring parents or potential parents, it will send a broadcast JOIN-REQ with a ttl of one but with range R. (The ttl determines whether the datagram should be rebroadcast without modification. The range R is another field that specifies if the node should send its own JOIN-REQ or JOIN-REQ.passive in response to the one that was received) All nodes withing range R(number of hops) from X will attempt to join the multicast tree by sending out their own JOIN-REQ.passive. The range R is decremented by one each time a node sends out the JOIN-REQ.passive so that eventually only nodes that are within range R(no. of hops from X) will have generated JOIN-REQ.passive. Nodes that receive a JOIN-REQ.passive with range R equal to zero do not generate any more JOIN-REQ.passve. Note that the contents of the sent JOIN-REQ.passive is different from the JOIN-REQ.passive in that the node will modify the contents with its own information before its sends it out.

A JOIN-REQ.passive received by an off-tree node will trigger that node to send a JOIN-REQ.passive only if the range is greater than zero. An on-tree node that receives a JOIN-REQ.passive will reply with a JOIN-ACK.passive. This sets up a set of passive links between the requesting node to possibly several on-tree nodes. The requesting node may eventually receive several JOIN-ACK.passive messages from different potential parents. It chooses one of the potential parents as its registered parent and sends a JOIN-CONF to that parent. The JOIN-CONF received by the parent node will trigger that parent node to send a JOIN-CONF if necessary to its own parent. The passive links maintained by the other potential parents will eventually time out. Potential parent nodes with passive links are allowed to respond to other nodes that might have sent out JOIN-REQ. In doing so, we "re-use" state information collected and minimize expanding ring type broadcasts.

Link breakage detection is performed in AMRIS by making use of the multicast beacon. The multicast beacon contains the msm-id of the node and the registered parent and child nodes of the node.

5. Protocol Specifics

We will explain the protocol in the following sequence:

1. Types of Membership/Node Classification
2. What is Sid and where does it come from?
3. Tree Initialization

4. Joining the Multicast Session
5. Reaction to mobility/broken links during JOINING phase
6. Forwarding Policy
7. Reactions to mobility/broken links
7.1 Leaf node moves
7.2. Intermediate moves
7.3. Core node moves
8. Reaction to Network partition
9. Group Membership
10. Terminating the Multicast Session

5.1 Types of Membership/Node Classification

Nodes are classified into the following types:

Interested node (I-node)
A node that is interested in a specific multicast session and wishes to join, it does not matter if it is interested to join as either a sender or receiver or both.

Uninterested Node (U-node)
A node that is not interested in a specific multicast session but is "forced" and became part of the session anyway because it is required to be a relay/intermediate node on the multicast delivery path. If a U-node does not have any descendent nodes that are I-nodes, it will prune itself off the tree. This is one of the main difference between a U-node and a I-node.

Leaf node
A node at the edge of the multicast tree. A leaf node maybe a sender or a receiver or both. U-nodes that becomes leaf nodes as a result of node movements will remain as leaf nodes only temporarily before they are pruned off.

Intermediate node
A node in the internal branches of the multicast tree. May be either an I-Node or an U-Node.

5.2 What is Sid and where does it come from?

Sid is defined as the node that initiated the multicast session by broadcasting the NEW-SESSION message to its surrounding nodes. In a single-sender, multiple-receiver environment, the single-sender would most likely assume the role of Sid. In a multiple-sender, multiple-receiver environment, it is most probable that one of the multiple-senders will assume the role of Sid in initiating the multicast session. Any core-election type of algorithm can be used if there are

multiple nodes contending for the role of Sid. Obviously, the choice of the core will have an initial effect on the shape of the delivery tree formed. We hope to have more insight on this matter after analyzing simulation results.

5.3 Initializing the Multicast Session

We assume that Sid has some means of obtaining a unique identifier for the multicast session. Sid creates a new multicast session by broadcasting a NEW-SESSION message to its surrounding neighbours. The NEW-SESSION will contain among other things, Sid's msm-id, multicast session address, and routing metrics. All nodes that receive the NEW-SESSION message will then generate their own msm-id and replace the msm-id in the message with their own, as well as various routing metrics before broadcasting the message again. Information derived from the NEW-SESSION message is kept in the Neighbour-Status table for up to T1 secs. A random delay of T2 secs is inserted between the receipt of a NEW-SESSION message and its subsequent rebroadcast. A node may receive multiple NEW-SESSION messages from different nodes. If it has not rebroadcast any messages yet, it will keep the message that has the best routing metric. That message is then modified and rebroadcast. Otherwise the messages received are dropped.

5.3.1 Bootstrapping the msm-id value

Each node that receives the NEW-SESSION message will calculate the initial value of its msm-id using hop count as a parameter to the function F1 Calculate_Initial_msm-id(Section 6 describes the function in greater detail). In a nutshell, the larger the hop-count value in NEW-SESSION message received, the larger will be its msm-id which is the value returned by Calculate_Initial_msm-id. The function's behaviour is such that there is a large msm-id insertion window between nodes that are one hop-count away. This facilitates the subsequent insertions of other nodes in between these two nodes. The beacon will now include the msm-id. If the beacon is implemented within the multicast routing layer and no other multicast session existed before this, the beacon is started up for the first time.

The NEW-SESSION message thus travels outward from the Sid in an expanding ring fashion. Eventually every node will have assigned to themselves a msm-id. The msm-id is used when the node joins the multicast session. The msm-id will be removed when the multicast session is no longer desired(or expired). This will release the msm-id usage back to the pool. msm-id is related to specific nodes supporting an active on-going multicast sesssion.

5.4 Joining the Multicast Session

When a NEW-SESSION message is received, a node X decides if it wishes to join the session by examining the contents of the message. (The contents of the message will obviously contain information about the multicast session). The joining approach attempts to fultil the join in an adjacent approach rather than resorting to localized broadcast right away. If the immediate neighbouring nodes are nodes already on the multicast tree, then this 1-hop 'peek' approach is very fast and efficient. The presence of msm-id helps in identifying the likelihood of a successful join. Our protocol does not cover the area of notifying the senders the identities of receivers(old or new). A node X joins a multicast session in one of several ways depending on the situation:

1) Node X has a valid msm-id and has a neighbour that also has a valid msm-id. (this is normally the case after tree initialization)

2) Node X does not have a valid msm-id and has a neighbour that also has a valid msm-id. (Probably node X missed the NEW-SESSION broadcast.)

3) Node X has a valid msm-id and does not have any neighbours with valid msm-id. (Probably the neighbours have timed out that msm-ids. X must therefore find other nodes that have valid msm-ids)

4) Node X does not have a valid msm-id and does not have any neighbours with valid msm-id. (Probably node X missed the NEW-SESSION broadcast.)

Case 1

Node X sends a JOIN-REQ to that neighbouring potential parent PP1. Potential parent PP1 checks that X can be a child node (i.e. X has a bigger msm-id than Y) of PP1 and sends a JOIN-ACK back to Node X. Potential parent PP1 updates its Neighbour-Status table to indicate that node X is a registered child of PP1. Node X updates its Neighbour-Status table to indicate that node PP1 is a registered parent of X.

Case 2

If Node X is aware that a neighbour Y can be a potential parent, node X will also know Y's msm-id because it must have received Y's beacon. Node X then assigns itself a msm-id using Y's as a parameter. Node X's situation is now similar to case 1 and node X will behave as in case 1.

Case 3-4

Node X initiates Branch Reconstruction (BR) with the msm-id. (Refer

to Section 5.7 for BR)

A node may also receive a JOIN-NACK instead of a JOIN-ACK, if so, node X will initiate BR based on the error code returned in the JOIN-NACK. If X does not receive any reply with T3 secs, it will initiate BR accordingly.

When a node PP1 receives a JOIN-REQ from a bigger-msm-id node X, PP1 will be in one of the following situations:

1) be part of the multicast session already (as in case 1 above)

or 2) it has a msm-id but has not sent out any JOIN-REQ before.(this happens when the node PP1 received the NEW-SESSION message and determined its own msm-id. However the node PP1 was not interested in joining the multicast session. Thus it did not send out any JOIN-REQ of its own yet.)

or 3) it has a msm-id and has sent out JOIN-REQ pending reply. (as in case 1 or 2 above)

or 4) it does not have a msm-id yet. (as in case 3 or 4 above)

PP1's reaction to receiving a JOIN-REQ (dependent on which situation it is it as discussed above) is as follows:

Case 1
If PP1's msm-id is smaller than the requesting node X, it sends a JOIN-ACK and updates its multicast state tables. if it is bigger or equal, it sends a JOIN-ACK.error="msm id too small/equal" to the requesting node. It updates the neighbour-Status table to indicate the node X is now a registered child node of node PP1. However no multicast traffic is forwarded to X yet until X increases its msm-id to be bigger than PP1. PP1 will know when X has increased its msm-id through X's beacon.

Case 2
PP1 must now join ('coerced' into joining by X) the multicast session. It sends out a JOIN-REQ.passive to its own potential-parent node. If PP1 successfully joins the multicast session, it will send a JOIN-ACK.passive back to requesting node X. Otherwise, it sends a JOIN-NACK with the appropriate error code. Since PP1 was 'coerced' into joining, it will not forward any traffic to X until X sends a JOIN-CONF message to confirm that X has selected PP1 as its parent. Note that this is slightly different from the case 1. (In general, JOIN-REQ/ACK.passive and JOIN-CONF is used when the parent node does not yet have a msm-id) During the time PP1 is

waiting for X's JOIN-CONF, the links through PP1 are considered as passive links. The passive links are converted to active links when a JOIN-CONF from X is received. The JOIN-CONF from X will PP1 to send its its parent a JOIN-CONF as well it the parent sent it a JOIN-ACK.passive previously.

Case 3

PP1 waits until it receives a reply for its own JOIN-REQ or times out. It sends a JOIN-ACK or JOIN-NACK depending on the success of its own JOIN-REQ to node X.

Case 4

If it does not have a msm-id, then no node should have sent it a JOIN-REQ. it sends a JOIN-NACK.error="NO msm-id yet" back to the JOIN-REQ node.

All other nodes that receive a JOIN-REQ with a broadcast address will forward the message provided the time-to-live is not zero yet. The node must keep track of who sent the JOIN-REQ so that a reverse path can be followed subsequently. A node PP1 may also receive a broadcast JOIN-REQ, i.e. a JOIN-REQ with a broadcast address. PP1 will then send out its JOIN-REQ but the ttl is always limited to one. It will follow the behaviour as in case 3.

Only the core node does not try to send a JOIN-REQ out when it itself receives JOIN-REQs.

5.5 Reaction to mobility/broken links during Joining Phase

We discuss how the protocol behaves when links break during Joining Phase.

Case 1

X lookups Neighbour-Status table and sees Y as a Potential Parent. X sends JOIN-REQ to Y. Y's link to X is broken thereafter. Hence, Y is unable to reply. X eventually times out and goes into BR mode. A broadcast JOIN-REQ is subsequently sent by X in BR mode. Y's status as a potential parent is removed from X's Neighbour-Status table.

Case 2

X sends JOIN-REQ to Y. Y is not on multicast tree., so Y must join multicast session as well. Y sends out JOIN-REQ.passive to Z. Z sends JOIN-ACK.passive but at this time link between X and Y breaks. Y tries to send to X but will not succeed. Since Y was a U-node, and has no I-node children. Eventually it will time out and prune itself off the tree. X will eventually go into BR mode as in case 1.

5.6 Forwarding Policy

The forwarding policy rules are as follows for multicast _data_ traffic(not control traffic):

1) When a multicast datagram is received, it first checks if the originator is itself, if so, it will drop the datagram.

2) If a multicast datagram is received from its registered parent node, it will forward to all registered child nodes.

3) If a multicast datagram is received from a registered child node, it will forward to all other registered child nodes and its registered parent node.

4) If a multicast datagram is received from any other node, it is not forwarded to any other nodes.

5.7 Reaction to mobility/broken links

When a link breaks between two nodes, the node with the larger msm-id is supposed to do recovery by going into BR mode. Nodes can also enter BR mode because of unsuccessful joins to its neighbouring nodes. A node X entering BR mode can be classified into having one of the following initial states:

1) X has a valid msm-id but does not have any neighbour nodes that can be potential parent or parent nodes.

2) X has a valid msm-id and has at least one neighbour node that can be potential parent or parent nodes.

3) X does not have a valid msm-id and does not have any neighbouring nodes that can be potential parent or parent nodes.

4) X does not have a valid msm-id and has at least one neighbouring node that can be potential parent or parent nodes.

In cases 1 and 2, X may have child nodes whose parent node is X. i.e. there is a sub-tree below X.

Initial State 1:
X sends a broadcast JOIN-REQ beginning with range R of two. If no reply is received after T3 secs, R is increased by one. This is repeated until the number of incremental attempts exceeds the recover delay that may have been specified. If there is no reply, then it means that the multicast session has either ended a long

time ago or a network partition has taken place. If X has child nodes, it will send a New-Partition-Id message to all its child nodes. If X does not have any child nodes, then X can either begin a new multicast session or X is unable to join the multicast session.

If a JOIN-ACK is received by X, it indicates that X has successfully joined the multicast session. X updates its Neighbour-Status table to indicate that the neighbour node which sent the JOIN-ACK as X's registered parent. If a JOIN-ACK.passive is received, X sends a JOIN-CONF to the neighbour node that send the JOIN-ACK.passive. X updates its Neighbour-Status table that the neigbour is the registered parent of X. A JOIN-ACK.passive is received because the was neighbour 'coerced' to join the multicast session. Since more than one neighbour might do this, a JOIN-CONF is necessary to select the neighbour to be the registered parent.

A node that sends out JOIN-ACK.passive does not forward any traffic until it receives a JOIN-CONF. Other requesting nodes can turn this node's passive links into active links if they send a JOIN-CONF to this node. If a JOIN-ACK.error="msm-id too small" is received, X increases its msm-id as required and sends a multicast beacon. If X is unable to increase its msm-id because of child nodes(the child nodes' msm-id may be equal or smaller then the required new value for X), X sends an Modify-msm-id message to its child nodes before increasing its own msm-id. X can detect if its child nodes have increased their msm-id when it receives their beacons.

Initial State 2:

X sends a JOIN-REQ to the potential parent node PP1. If a JOIN-ACK is received, then X updates its Neighbour-Status table that PP1 is the registered parent of X. If no reply is received from PP1 after T3 secs, X goes into BR mode with initial state 1.

Initial State 3:

X sends a broadcast JOIN-REQ as in Initial State 1. It's msm-id is set to a random value. Any necessary adjustments will be in response to a JOIN-ACK.error="msm-id too small" message.

Initial State 4:

X can determine a msm-id for itself based on its neighbour's msm-id. It then re-enters BR with initial state 1.

In the case of a JOIN-REQ broadcast, the child nodes Cx of the node X executing BR will also receive the JOIN-REQ. When they rebroadcast it, they must keep track of the rebroadcasted JOIN-REQ so that if a reply should return from one of the child node's(C*) neighbor, the child (C*1) must check that the msm-id of the

neighbour is smaller than its own, if not it will send a JOIN-NACK(msm-id error) instead of forwarding the JOIN-ACK to its parent. Child nodes which receive a JOIN-NACK will just forward it along.

If X (the node executing BR) receives a JOIN-NACK(msm-id error) forwarded through an on-tree child node, it is treated differently than if it were received from other non-child nodes. To node X the message means that a route exists to a potential parent but that route runs through the its children's links instead. (This new route may be the only available route to the rest of the multicast tree). If the BR has no other routes to potential parents to choose from, then it must change its msm-id such that it is now larger than its children. This process continues to the descendent node which swapped the JOIN-ACK with the JOIN-NACK(msm-id error). This is a really worse case scenario. The BR node can initiate the change by sending a Reverse-msm-id message to the affected child nodes. The Reverse-msm-id message is forwarded only to downstream neighbours that require the change as a result of the BR node's new msm-id. It will stop at the node which swapped the JOIN-ACK with JOIN-NACK.

We have looked at what happens when the broken link is between a single upstream node and a single downstream node. We want to refine the basic policy if the broken link is between a single upstream node and multiple downstream nodes.

If there originally was a group of neighbouring nodes participating in the same multicast session and their common parent node moves away, then instead of having all the nodes broadcast a JOIN-REQ, it makes more sense to have only a subset of those nodes broadcast the JOIN-REQ. This is done by introducing a short random delay before a JOIN-REQ is sent if a node has more than one neighbour sharing the same parent who is also a member of the multicast session. If the node hears its neighbour JOIN-REQ, it will delay its own JOIN-REQ until either it hears a JOIN-ACK to its neighbour and some timer timouts. A node X will wait random time before transmitting JOIN-REQ. if it hears a JOIN-REQ from a neighbour for a parent node that is also suitable for X. X will delay its JOIN-REQ for Ts. If it hears a JOIN-ACK from the PP1 node, it will send its own JOIN-REQ directly to PP1. If no reply is heard, then it will broadcast its own JOIN-ACK after Ts.

We look at some link breakage scenarios and illustrate how BR works out in each case.

5.7.1 Reaction to leaf node movement

There are two situations of leaf node movement.

5.7.1.1 Situation 1: Leaf node moves to new location where multicast tree is already established

When the leaf node C1 moves, its parent P1 will eventually detect this. P1 will proceed to prune itself from the tree if P1 is a U-node and has no other child nodes. The pruning is as similar to that described in Section 5.9 Group Membership except the parent does not receive explicit notification from the child node. P1 has received implicit notification as a result of broken link to C1.

When C1 moves into new location L2, it can hear node P2's multicast beacon and discover that a multicast tree already exists there. (Another way that C1 can discover P2 is on the multicast tree is hearing multicast traffic for that session being forwarded by P2). If P2 has a smaller msm-id than C1 then C1 can immediately send a JOIN-REQ to P2. If P2 has a larger msm-id, then C1 will increase its msm-id using P2's msm-id as a parameter into Calculate_MSM_ID() to acquire its new msm-id. C1 then sends another JOIN-REQ using its new msm-id. P2 replies with a JOIN-ACK to C1 after it receives the second JOIN-REQ with the new msm-id from C1. It will subsequently forward any multicast traffic to C1 as well. This is similar to the tree initialization process.

[Note: C1 does not send a leave notification control message to its old parent P1. Reason for not sending: Network resource consumed in sending a leave message to P1 from C1's new location, must include resource consumed to establish path to old parent no de etc.]

5.7.1.2 Situation 2: Leaf node moves to new location where no multicast tree exists

The behaviour of the original parent P1 is as in situation 1. When leaf node C1 reaches new location L2, it is unable to discover any existing neighbours who can serve as parent nodes. It then initiates a broadcast BR to its neighbours. Neighbouring nodes which receive the JOIN-REQ will send out their own JOIN-REQ.passive. This is repeated at each node until the range reaches 0. An on-tree node that receives the JOIN-REQ.passive will send a JOIN-ACK.passive. The JOIN-ACK.passive travels along the reverse path taken by the original JOIN-REQ and JOIN-REQ.passive. JOIN-REQ-PENDING table. The multicast delivery tree now extends to C1 through a set of passive links. C1 must send a JOIN-CONF along one of the passive links to turn it into an active parent link. The parent nodes will start to forward traffic only when a JOIN-CONF is received.

5.7.2 Reaction to Intermediate node movement

We explain the action taken by an intermediate node that moves such that the link to is parent is broken but the link to its child nodes are still active. The action taken by the intermediate node is similar to that in 5.7.1. except if it receives a JOIN-ACK.error="msm-id too small". Then the intermediate must send a Modify-msm-id message to all its child nodes. If the intermediate node's child links are broken as well, then intermediate node need only to perform the BR if it is a I-node.

5.7.3 Reaction to Sid node movement

When Sid moves such that its child links are broken, the child nodes will begin a BR process towards the Sid. Only a subset of the child nodes actually broadcast the JOIN-REQ as a result of the optimization performed.

5.8 Reaction to Network partition

A network partition will cause the multicast tree to be divided into different segments. When a partition takes place, the node X at the point of partition will behave as though it has lost the parent link and will behave as discussed in section X and X. If BR does succeed after n-tries, we can assume that a network partition has take place.

Within each partition, multicast traffic continues to be forwarded in accordance to the forwarding rules as stated in Section 6.4, i.e. intra-partition traffic can continue as per normal but inter-partition traffic cannot take place.

When this happens, the child node (who is also the one with the smallest-id within this partition) at the point of link breakage will become the new temporary Sid-temp, i.e. the node with the smallest msm-id within that partition. At a later time, the network may be such that a path now exists either between the Sid-temp or one of its child node back to the original partition.

When a partition takes place, the Sid-temp should send a Partition-Id change message to its decendent nodes. The message can be sent standalone or piggybacked along multicast traffic. This will update all child nodes eventually. The multicast beacon will carry the new partition-id.

Subsequently, any node that hears a beacon for the same multicast session but with a different partition id will send a FOUND_PARTITION message to its own partition's Sid-temp. Only the node with the

smallest msm-id will send the message to its own partition's core. The node in the other partition will not. If there is a tie, the one with largest unique id will inform its core. The core will send a JOIN-REQ back to its child node if after it compares the msm-ids, unique ids, of the reporting nodes and the other partitions reporting nodes.

The temp core may receive several of such messages. It will choose the one with the best routing metric and send a JOIN-REQ message back to the node. Intermediate nodes which receive the JOIN-REQ will react as discussed in Section 5.7. When a JOIN-ACK returns in response to the JOIN-REQ, the temp core may need to send out Modify-msm-id messages to a subset of its child ndoes that lie on the path to the other partition.

So in this situation, we have two partitions with joining together. The node with the smallest msm-id will eventually become the new Sid. However we have a problem if we have two partitions whose core have the same msm-id, then neither partition will join with each other but they are still in different partitions because the original core may be destroyed.

5.9 Group Membership

Group membership is dynamic in nature, any node is free to join or leave the node at any time except for the Sid. If Sid wishes to leave, another node will have to be designated as the new Sid before the present one can leave.

Nodes wishing to join the multicast session after it has already "started" will do so by sending a JOIN-REQ to any of the on-tree nodes as discussed in Section 5.4. On-tree nodes can be detected by listening to its neighbours' beacons and detecting if any of the neighbours are already an on-tree node. If the node is unable to locate any neighboring on-tree nodes, it will begin a N-hop-lifetime JOIN-REQ broadcast to its neighbours. N begins from 2 to MAX-TTL. After sending the broadcast, it will wait for a reply. If no reply is received within time T1, it will increase N by one and send again.

Any node that receives the JOIN-REQ and is not an on-tree node will update its NEIGHBOUR-STATUS table and rebroadcast. The behaviour here is similar to tree initialization except that the intermediate nodes may not yet have a msm-id. Similarly the I-node also does not yet have a msm-id. Eventually, the JOIN-REQ will be received by an on-tree node which will reply with a JOIN-ACK. The nodes along the path from the I-node to the on-tree node will then dynamically program their msm-id using their respective parent node's msm-id as parameter into function Calculate_Initial_MSM_ID.

A leaf node can leave the multicast session by sending a SESSION-LEAVE message to its parent node. If the parent node is a U-node in the multicast session. It too will send a SESSION-LEAVE message to its own parent node provided it does not have other child nodes.

5.10 Terminating the Multicast Session

Only Sid can close a multicast session. Nodes are of course free to leave as and when subject to conditions in Section 5.9 - Group Membership.

Sid closes the multicast session by sending a SESSION-END message down to its child nodes. This is rebroadcasted by intermediate nodes. All nodes that receive the SESSION-END message purge any entries/state associated with the multicast session. The nodes store the session-id for up to time T4 before finally expiring that entry. This is to take into consideration partitioned networks. If the network is partitioned, the original Sid may have closed the session but the other partitions have not since they did not receive the message. For this reason, nodes which receive the END-SESSION message will remember the session-id for time T4. If the node hears a beacon with state about the session. It will forward the END-SESSION message to it.

Each multicast entry in the Neighbour-Status table has an associated timeout value. Session termination can also be done via a soft state approach, i.e. when the entries expire in the Neighbour-Status table.

6. msm-id

The Multicast Session Node ID (msm-id) is a numerical value that identifies a node with a particular multicast session as well as give an indication of the node's relative distance from the root of the tree.

```
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |        Multicast Session Node ID              |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

6.1.1 Calculating the msm-id

Address Mapping Function - Calculate_Initial_MSM_ID

An address mapping function Calculate_Initial_MSM_ID is required to map the node's current hop count from the session initiator into a larger address space. We illustrate with an example of how the

function should perform:

Suppose M is the number of bits of msm-id, let's say 16 bits. The value is calculated using a function that takes in the hop count in the NEW-SESSION message and returns a value that is within the 16bit range.

The value returned must be an increasing function of hop count**. We do not give a specific function that one must use here. A possible function, (though not a very good one is):

```
msm-id=INT((2^(M/2))/hop_count)
```

The idea is the function will give nodes a widely spaced address within the allowable range with the hop count as a initial parameter. The larger the hop count, the larger should be the multicast node ID and thus msm-id as well.

**During multicast delivery path initialization, we just want to use hop count to "jumpstart" the node's msm-id, subsequently the node's msm-id can change as it moves around. Its msm-id would then be a function of the upstream and downstream nodes' msm-ids.

We want to utilize a sparse address space because we foresee such a situation:

```
Node X ===> Node Y ===> Node Z ===> downstream nodes
```

Suppose at time t, node X and node Z can communicate directly. Node Z has a set of downstream nodes. At time t+1, link between node X and node Z breaks. They must now communicate through an intermediate node Y.

If we are just using hop-count directly as msm-id, then Z and its downstream nodes must all increment their msm-id by one since datagrams must go through 1 extra hop through Y. Some overhead will be required to inform Z's downstream nodes to increment their msm-ids. If we use a sparse address space, Y can just acquire an address that is of the nature (X+Z) div 2. The increasing msm-id property is maintained and Z and its sub-tree need not update their msm-ids. There are other situations when this is also useful.

To ensure that the msm-id remains consistent, each node would require a self-check routine. This routine would possibly be based on the msm-id of one's neighbours. Another issue is the scalability and numbering overflow of msm-id. This will be examined in our future work.

7. Multicast Beacon

The Multicast Beacon can be "piggy-backed" on existing underlying layers' beacons, if they are available. Alternatively a separate daemon to periodically broadcast the multicast beacon can be used, existing within the multicast routing layer.

The beacon contains the following fields:

1) Node-unique-id
2) msm-id and status (member/non-member;lifetime)
3) Registered parent msm-id and unique-id
4) Registered child msm-id and unique-id
5) partition_id(initially will be the original core's id)

Every node must implement the beacon mechanism. The beacon is a one-hop broadcast message sent by every node. The main use of the beacon is to detect link breakages within a fixed time interval(this is be closely related to the beacon interval)

8. Routing Metric

We are investigating how routing metrics such as associativity[7], received signal strength[8] and can be incorporated into AMRIS to select "better" routes. This will be discussed further insubsequent drafts.

9. Tables 1. Neighbour-Status Table

[work-in-progress]

Neighbour unique-id	msm-id	Relation-Type (eg. parent or child)	Status	Remaining Timeout Value	Routing Metric

10. Message Formats

[work-in-progress] 10.1 NEW-SESSION

10.2 JOIN-REQ/JOIN-REQ.passive

10.3 JOIN-ACK/JOIN-ACK.passive/JOIN-NACK

10.5 JOIN-CONF

10.5 Modify-msm-id

10.6 LEAVE-SESSION/END-SESSION

11. Detailed psuedocode [work-in-progress]

12. Timers Description

Timer Name	Timer value	Timer Purpose
T1		NEW-SESSION Lifetime
T2		Random delay between receipt of NEW-SESSION and subsequent rebroadcast
T3		Time out value for JOIN-REQ
T4		End-Session message lifetime

13. Future Directions

Perform Simulation to investigate performance and feasibility. Investigate overhead of signalling requirements. Investigate QOS aspects.

14. Applicability Statement

* Does the protocol provide support for unidirectional links? (if so, how?)

- No, bidirectional links are assumed with the first draft.

* Does the protocol require the use of tunneling? (if so, how?)

- No.

* Does the protocol require using some form of source routing? (if so, how?)

- No.

* Does the protocol require the use of periodic messaging? (if so, how?)

   - Yes. The periodic messaging is in the form of a periodic
beacon.

   * Does the protocol require the use of reliable or sequenced
packet
   delivery? (if so, how?)

   - No.

   * Does the protocol provide support for multiple hosts per
router?
   (if so, how?)

   - This will be covered in subsequent drafts.

   * Does the protocol support the IP addressing architecture? (if
so,
   how?)

   - Yes, the protocol is based on the IP multicast host group
model.

   * Does the protocol require link or neighbor status sensing (if
so,
   how?)

   - Yes, this is done either by the periodic beacon or by
underlying layers if such facilities exist there

   * Does the protocol have dependence on a central entity? (if so,
   how?)

   - No.

   * Does the protocol function reactively? (if so, how?)

   - Yes, the protocol reacts accordingly to maintain tree when
links are broken

   * Does the protocol function proactively? (if so, how?)

   - No.

   * Does the protocol provide loop-free routing? (if so, how?)

   - Yes.

   * Does the protocol provide for sleep period operation? (if so,

how?)

- TBD.

* Does the protocol provide some form of security? (if so, how?)

- TBD.

* Does the protocol provide support for utilizing multi-channel, link-layer technologies? (if so, how?)

- Yes. TBD.

15. References

[1] T. Pusateri. Distance Vector Multicast Routing Protocol. Internet-Draft, draft-ietf-idmr-dvmrp-v3-06.txt, March 1998.

[2] John Moy. Multicast extensions to OSPF. RFC1584, March 1994.

[3] A.J. Ballardie. Core Based Trees(CBT) Multicast Routing Architecture, RFC2201 (September 1997).

[4] D.Estrin, D.Farinacci, A. Helmy, D.Thaler; S. Deering, M. Handley, V.Jacobson, C. Liu, P.Sharma, L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC2362, June 1997.

[5] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helmy, and L. Wei. Protocol Independent Multicast Version 2, Dense Mode Specification. Internet-Draft, draft-ietf-idmr-pim-dm-spec-05.ps, May 21, 1997.

[6] Corson, S., and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", Internet Draft, draft-ietf.manet-issues-00.txt, September 1997.

[7] C-K Toh, "Associativity Based Routing For Ad Hoc Mobile Networks" Wireless Personal Communications Journal', Special Issue on Mobile Networking & Computing Systems, Vol. 4, No. 2, March 1997.

[8] Rohit Dube, Cynthia D. Rais, Kuang-Yeh Wang and Satish K. Tripath. Signal Stability based Adaptive Routing (SSA) for Ad-Hoc Mobile

Networks, CS-TR-3646, August 1996.

This work was supported in part by National University of Singapore ARF grant RP960683.

Author's Address

Questions about this document can also be directed to the authors:

C.W. Wu
Mobile Computing Group
School of Computing
National University of Singapore
10 Kent Ridge Crescent
Singapore 119260
(65) 472-2628
wuchunwe@comp.nus.edu.sg

Y.C. Tay
Department of Mathematics
National University of Singapore
10 Kent Ridge Crescent
Singapore 119260
(65) 874-2949
tay@acm.org

C-K. Toh
Mobile Multimedia & HiSpeed Networking Lab
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia GA 30332, USA
C-K.Toh@acm.org