Journal of Theoretical and Applied Information Technology
31st May 2022. Vol.100. No 10

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A NEAR OPTIMAL MULTICAST SCHEME FOR MOBILE AD-HOC NETWORKS – AN IMPLEMENTATION PERSPECTIVE

MR. P ANIL KUMAR[1] , Prof. CH KAVITHA[2] , Prof. M BABURAO[3] , Dr. CH SURESHBABU[4]

[1]Department of CSE, PVP Siddhartha Institute Of Technology, Vijayawada, Andhra Pradesh, India
[2]Department of IT, S R Gudlavalleru Engineering College, Gudlavalleru, Andhra Pradesh, India
[3]Department of CSE, S R Gudlavalleru Engineering College, Gudlavalleru, Andhra Pradesh, India
[4]Department of IT, S.R.Gudlavalleru Engineering College, Gudlavalleru, Andhra Pradesh, India

Email: [1]anilkumar.pallikonda@gmail.com, [2]kavithachaduvula12@gmail.com, [3]baburaompd@gmail.com, [4]dr.sureshbabuch@gmail.com

ABSTRACT

An ad-hoc mobile network is a collection of mobile nodes that are dynamically and arbitrarily located in such a manner that the interconnections between nodes are capable of changing on a continual basis. The primary goal of such an ad-hoc network routing protocol is correct and efficient route establishment between a pair of nodes so that messages may be delivered in a timely manner.**Multicasting** is to send single copy of a packet to all of those of clients that requested it, and not to send multiple copies of a packet over the same portion of the network, nor to send packets to clients who don't want it.The **Adhoc Multicast Routing Protocol (AMRoute)** presents a novel approach for robust IP Multicast in mobile ad-hoc networks by exploiting user-multicast trees and dynamic logical cores. It creates a bi-directional, shared tree for data distribution using only group senders and receivers as tree nodes. Unicast tunnels are used as tree links to connect neighbors on the **User-multicast tree**. Thus AMRoute does not need to be supported by network nodes that are not interested/capable of multicast, and group State Cost is incurred only by group senders and receivers. Also, the use of tunnels as tree links implies that tree structure does not need to change even in case of a dynamic network topology, which reduces the signaling traffic and packet loss. Thus AMRoute does not need to track network dynamics; the underlying Unicast protocol is solely responsible for this function. AMRoute does not require a specific Unicast routing protocol; therefore, it can operate seamlessly over separate domains with different Unicast protocols. We have tried to overcome the transient loops in the mesh creation. Also we have implemented the Dynamic core migration technique by using a timer which periodically changes the current core node, so that the efficiency of the protocol can be improved.

Keywords*: Adhoc, Multicast, AMRoute, Multicast Tree*

1. INTRODUCTION:

Networking has stormed the world today with its multi-faceted applications. Communication today has reached great heights with the help of Networking.

1.1 Types of Network

Networks can be broadly classified into two types depending on the type of connectivity.

1.1.1 Wired Networks:

As the name implies the various terminals of the wired network are connected together with a wire. Thus data is transferred through this physically existing wire links.

1.1.2 Wireless Networks:

These are networks that do not have a physical (namely a wire) connection between the various terminals. Wireless networks find their importance in those areas where it is difficult to establish a connection through a wire. This was the initial motive behind the development of wireless networks. But once the flexibility and advantages were realized these networks started ruling the world. The basic reason is the ability to communicate without any physical connection. Wireless networks can be further classified into two types.

Journal of Theoretical and Applied Information Technology
31st May 2022. Vol.100. No 10

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

1.2 Infrastructure Wireless Network

This is the traditional cellular network model that supports the current mobile computing needs by installing base stations and access points. In such networks, communications between two mobile hosts completely rely on the wired backbone and the fixed base stations. A typical infrastructured network is shown in the figure.

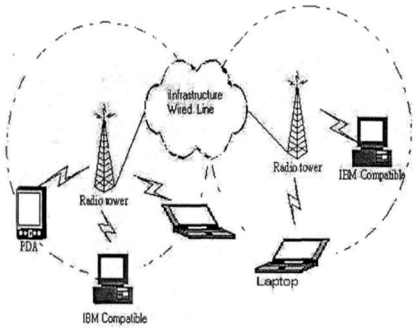

FIGURE 1.1 Infrastructured Network.

1.3 Infrastructureless Networks or ADHOC Networks:

In Ad Hoc wireless networks, there exists no base stations and each mobile host is smart enough to act as a router to forward packets from one to the other until the packet reaches its destination. The intelligent communications software enable these mobile computers and devices to form and deform networks on the fly, in real-time.

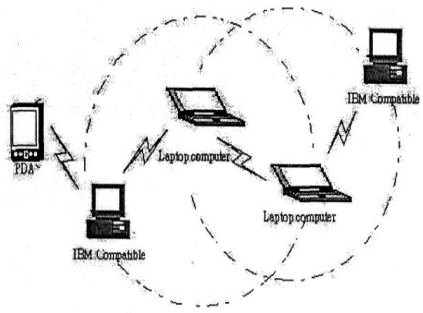

FIGURE 1.2 Infrastructureless Networks ADHOC Networks

Ad Hoc networks are highly dynamic in nature since they can form and deform quickly, without the need for any infrastructure setup and system administration. They can be deployed anytime and anywhere (indoors and outdoors), be it battlefields or conference rooms.

1.4 Multicasting in Ad Hoc Networks:

Multicast is a type of communication used for communicating between groups of computers, as opposed to unicast, which is intended for communication between pairs of computers. In a typical ad hoc environment, network hosts work in groups to carry out a given task. Hence, multicast plays an important role in ad hoc networks. The main advantage of multicasting is that a sender only needs to send the data once so that significant resources (e.g., network transmission bandwidth) can be saved. The following functions are to be performed.

1. Sender sends each set of data only once.
2. Receiver can participate in the multicast at any time by joining the multicast group and receiving the sent packets.

To support multicasting, routers within the networks need to maintain much information such as

1. Membership for each multicast group.
2. Input/output ports for each multicast group so as to perform packet forwarding.
3. Packet error recovery within a multicast group.

1.5 Multicast Protocols:

Existing multicast protocols do not work well in ad hoc networks as the frequent tree reorganization can cause excessive signaling overhead and frequent loss of datagrams. Hence there is a need to develop a separate multicasting protocol for Ad-Hoc networks.

There are two categories of Multicast protocol

1.5.1 Proactive Protocol:

These algorithms maintain a route to the destination much like the traditional fixed networks. When the packet is to be transmitted it just picks up a root from the cache and uses it. This leads to the instant transmission of the first packet, but the nodes have to work hard in the background wasting precious radio resources.

E.g.: AMRoute – Ad-Hoc Multicast Routing Protocol

1.5.2 Reactive Protocol:

As the name suggests these types of algorithm try to evolve the route to the destination only when they need it to transmit a data packet to the destination. Usually a route discovery phase proceeds the actual send phase. This on the fly reaction may take a long time for transmission of the first packet, but is efficient in terms of the data packet to routing control packet ration saving battery power in the mobile nodes.

E.g.: AODV – Ad-Hoc On demand Distant Vector Protocol.

In our project we analyze the AMRoute protocol and study the various characteristics of the protocol.

1.6. Goal of the work

To perform multicast routing in ad hoc networks .

1.6.1 Problem Identification

Existing multicast protocols do not work well in ad hoc networks as the frequent tree reorganization can cause excessive signaling overhead and frequent loss of datagram and core node is not identified to pass the segment.

1.6.2 Problem Solution

Finding the core node, each message will be sent by segments. Message Queues are used to establish Inter Process Communication (IPC CALLS).

1.7 Applications of Ad Hoc Networks:

- Emergency search-and-rescue operations.
- Meetings or conventions in which persons wish to quickly share information.
- Data acquisition operations in inhospitable terrain.
- Local area networks in the future.

2. WORK DESCRIPTION

2.1 Algorithm

- A mesh is created between the members of the group by a Mesh Creation technique, which involves broadcasting a Control Packet to identify the members of the Group. This is an " Expanded Ring Search" algorithm.
- Each of the mesh created consists of a Logical Core node, which is responsible for maintaining the tree and its members. The core is selected by using a "Core Resolution" algorithm.
- Once a Mesh is created a User Multicast Tree is built from it. This tree is formed in such a way that the nodes of the tree are the members of the group.
- The next step is to maintain the Tree created. This is done by periodically sending a message to all the members of the group. The core node is responsible for sending this packet. It maintains a TREE_CREATE_TIMER.
  - ✓ To improve the efficiency of the AMRoute protocol a Core Migration technique is used in our algorithm. A new core is being elected periodically so that the core migrates and thereby the tree is maintained effectively.

A description of each of the above steps is given in detail in the following pages.

2.1.1 Expanding Ring Search Algorithm

This algorithm is used to find out the members of a group in AMRoute. The members found are logically linked to each other in the form of a mesh. Hence each of the nodes in the mesh can be identified as the neighboring node.

The algorithm can be explained as follows:

An expanding ring search works by searching larger areas, centered around the source node, until a node with a route to the destination is located. The basic premise behind the expanding ring search is to find some local node with a route to the destination and thereby avoid flooding the entire network in search of a route. Using an expanding ring search, the initial JOIN_REQ has a small time to live (TTL) i.e., two hops. Each time the JOIN_REQ is rebroadcast, the sending node decrements the TTL. Once the TTL reaches zero, the JOIN_REQ is no longer forwarded. The source node waits the discovery period for a JOIN_ACK to be returned. If it has not received a JOIN_ACK by the end of the discovery time, it initiates a new JOIN_ACK with the TTL increased by an increment. This process continues until a threshold TTL value is reached. After this point, if no route has been located the JOIN_REQ is flooded across the network. The following figures illustrate the expanding ring search algorithm. The shaded nodes indicate nodes, which have a route to the destination. In a larger network with more than that illustrated the number of node undisturbed by the request query would be likewise greater. When re-discovering a route after a link break, the source places the last known hop count to the destination in the TTL is increased by the increment value. The TTL is increased on each subsequent route discovery attempt until the TTL threshold is reached. After this point the JOIN_REQ is just flooded to the entire network.

Journal of Theoretical and Applied Information Technology
31st May 2022. Vol.100. No 10

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

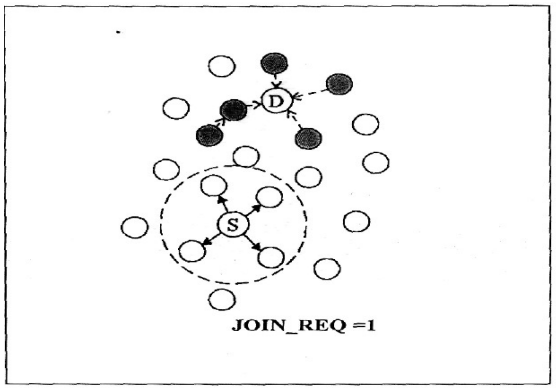

FIGURE 2.1: Request of mesh=1

The above three figures illustrate the Expanding Ring Search algorithm. In the first figure the JOIN_REQ packet is transferred with a TTL value of 1. Hence when the neighbors of the source receive that packet they try to decrement the value of TTL and rebroadcast it. But the value of TTL becomes Zero after decrementing. Hence they don't forward the message. Meanwhile the source waits till the discovery time. Once the timer expired the source again transmits the JOIN_REQ message incrementing the TTL value.

The second figure illustrates the state with a hop count of 3. With this hop count the JOIN_REQ packet reaches the neighbors of the destination node. Thus the destination node receives the JOIN_REQ packet and it transmits the JOIN_ACK packet. The third figure depicts the route from source to destination.

The following section explains how a mesh is actually created in the AM Route protocol.

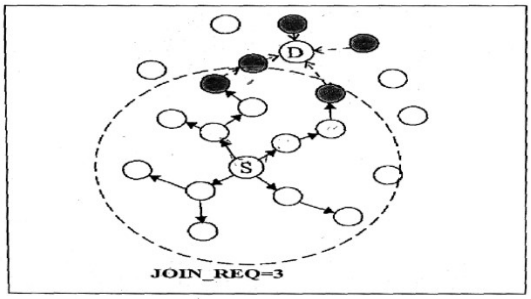

FIGURE 2.2: Request of mesh=3

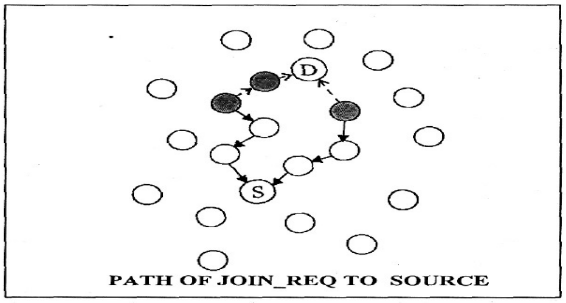

FIGURE 2.3: Path of request to source

2.2 Mesh creation

An AMRoute mesh is a graph where each node is a member (sender or receiver) of the group and every link is a bi-directional unicast tunnel. While the mesh establishes connectivity across all the members of a group, a tree is needed for forwarding the data. We use a two step process of creating a mesh before the tree because it is simpler and more robust.

A mesh is much simpler to maintain a mesh (that could potentially have loops) than a tree (with no loops) at every state of member mutual discovery phase. For example, a very naive merging algorithm could result in a loop when three disjoint trees discover each other. In addition, the redundant mesh links contribute towards increased robustness in the case of node/link failures. (Note that the use of unicast tunnels between neighboring nodes of the mesh itself contributes towards robustness in the face of intermediate node/link failures along routes between them as the unicast protocol is expected to establish a separate route around the failed network node/link).

2.2.1 Periodic JOIN_REQ Broadcast

To create a mesh encompassing all the members (senders or receivers) of a specific group, mechanisms are needed to allow members to discover each other. The expanding ring search mechanism based on TTL limited responds back with a JOIN_ACK. A new bi-directional tunnel is established between the core and the responding node of the other mesh. As a consequence of mesh mergers, a mesh will have multiple cores. One of the cores will emerge as the "winning" core of the unified mesh as a result of the core resolution algorithm.

2.2.2 Becoming a Passive Non-core node

Only logical core nodes initiate the discovery of other disjoint meshes, while both core and non-core nodes respond to the discovery messages. It is simpler and more scalable (in bandwidth usage) to have only the core node initiate discovery as

against every node of the mesh. However, to avoid the situation where every merger adds a link to a core (which might result in too many links from the core), non-core nodes can participate in the mergers by responding to discovery messages.

2.2.3 Leaving the group

If a node leaves a group, they send out a single JOIN-NAK message to their neighboring nodes. If they subsequently receive any data or signaling message for that group they can send out further JOIN-NAK messages.

2.2.4 Limitation on link connectivity

Whenever, the number of links adjacent to a node exceeds LINK THRESHOLD, a node must break one or more of its links. Each of the links could be associated with a weight representing the "distance" (example, a number of hops) from the neighbor. The links chosen to be broken could be the ones with the farthest neighbors. The farthest neighbor is notified about the link breakage using a JOIN-BAK message. If the message is lost and data is received from this non-neighbor, the JOIN-NAK message will be resent. Periodic mesh reconfiguration is necessary to maintain a reasonably optimal mesh in the face of mobility; however, removing links may result in temporary loss of data and additional overhead. When the links are broken, the mesh might be fragmented into disjoint meshes. Fragmentation is handled in the same manner as node failures.

2.2.5 Dynamic Core Migration

There might be a need for dynamically migrating the logical core so as to make the tree more optimal. If the core is "closer" to the senders, then the tree may be a close approximation of a source-based tree (which is shared). If the core is at the "center" of the mesh, then TREE-CREATE messages (described in the next section) reach all the nodes of the mesh faster than when the core is at the "edge". In addition, as the core is involved in discovering and merging with other disjoint meshes, dynamically changing the core might help in avoiding the situation where there are excessive links adjacent to a core node. Policies and mechanisms for node changes are still under research.

2.3Core resolution Algorithm

Before starting with the core resolution algorithm, the operations performed by a core node are explained. In the AMRoute protocol each group has at least one logical core that is responsible for initiating signaling actions, specifically:

- ✓ **Mesh joins** (discovering new group members and disjoint mesh Segments) and
- ✓ **Multicast tree creation**.

A non-core node cannot initiate these two operations, acting only as a passive responding agent. Limiting the number of nodes that perform these two functions (ultimately to a single logical core) ensures that AMRoute can scale, without causing excessive signaling or processing overhead.

An immediate reaction on hearing about using core nodes is that "Yes, a core improves scalability (as shown by CBT or PIM-SM, but it also causes robustness problems in dynamic networks. " However, the AMRoute logical core node is different from a CBT core and PIM-SM RP in several fundamental aspects. In particular, an AMRoute core node:

- ✓ is not a central point for all data. Forwarding can continue on working branches of the tree irrespective of the status of the logical core and links to the logical core.
- ✓ Is not a preset node. Each multicast tree segment designates one of its nodes o be the core based on the "core resolution algorithm".
- ✓ Changes dynamically. The core node migrates according to group membership and network connectivity.

The core resolution algorithm is run in a distributed fashion by all nodes. The goal of the algorithm is to ensure that any group segment has exactly one core node and that the core node migrates to maintain a robust and efficient multicast tree.

An AMRoute segment can temporarily have more than one core node for group after new nodes join or disjoint segment emerge together. A network node designates itself as a core when first joining a group. As a logical core a node can quickly discover new group members and join the mesh and tree with its closest neighbors (not just to the existing core). When multiple core nodes exist in a segment, they will advertise their presence by sending out tree creation messages. Core nodes use the reception of tree creation messages from other cores to decide whether to remain as a core.

An AMRoute segment can also have no core nodes because the core node disappears (e.g., leaves the group) or an existing segment is split into multiple disjoint segments (e.g., because of link or node failure). If a segment does not have a core node, one of the nodes will designate itself as the core node at some random time, on not receiving any join or tree creation messages.

A key issue with any algorithm that assigns a single core node is that it can centralize

the multicast tree and indeed the mesh links on itself. AMRoute prevents centralization in a number of ways:

- A non-core node is not allowed to graft to its own logical core. Without this limitation all group members would ultimately be connected to the core.
- All nodes, including the core, are only allowed to have a limited number of tree links. If the limit is reached the node must drop the link furthest (at highest cost) from its current location.
- A logical core will only take responsibility as core for a limited time or until some event makes changing the core desirable. A new logical core can be picked, for example when the core's mesh connectivity limit is reached.

Clearly the core resolution and change algorithms are key to the robustness and performance of the AMRoute protocol. However, it is also desirable to contain the complexity of the algorithms. Simulations are hence being planned to determine the tradeoffs between simplicity, robustness and efficiency.

2.3.1 Algorithm

In the event of mesh mergers, there might be multiple active cores in the new mesh. Nodes in the mesh become aware of this situation when they receive TREE-CREATE messages from multiple cores. The nodes execute a core resolution algorithm to decide on a unique core for the mesh and forward TREE-CREATE messages arriving from the unique core and discard TREE-CREATE messages from other cores. As the multiple cores in the mesh will also become aware of the existence of other cores, they will also execute the same core resolution algorithm. All the cores except the "winning" core will demote themselves to non-core state. One simple core resolution algorithm could pick the winning core to be the one with the highest IP address. This is the technique used in our protocol. The Core resolution algorithm picks up the node, which has the highest IP address.

Once the core has been picked up the next step is to create a tree using the mesh links. The creation of user multicast tree is explained in the next section.

2.4 Tree Creation

This section discusses the creation of a tree for data forwarding purposes once a mesh has been established. The core is responsible for initiating the tree creation process. From the point of view of individual nodes of the mesh, this phase involves identifying the subset of links adjacent to it that belong to the tree.

2.4.1 Periodic TREE-CREATE Broadcast

The core sends out periodic TREE-CREATE messages along all the links adjacent to it in the mesh. (Note that TREE-CREATE messages are sent along the unicast tunnels in the mesh and are processed by group members only, while JOIN-REQ messages are broadcast messages that are processed by all network nodes).

The periodicity of the TREE-CREATE messages depends on the size of the mesh and also on the mobility of the nodes of the mesh. As the mesh nodes are mobile, the number of hops between neighbors keeps changing dynamically. Thus, newer and more optimal trees might be created when TREE-CREATE messages are sent out. Group members receiving non-duplicate TREE-CREATEs forward it on all mesh links except the incoming, and mark the incoming and outgoing links as tree links. If a node has a collection of neighbors all 1-hop away on the same broadcast capable interface, then the node can send a single broadcast message to all 1-hop neighbors simultaneously.

2.4.2 TREE-CREATE-NAK

If a link is not going to be used as part of the tree, the TREE-CREATE message is discarded and a TREE-CREATE-NAK is sent back along the incoming links. On receiving a TREE-CREATE-NAK, a group member marks the incoming link as a mesh link and not a tree link. Thus each non-core node considers the link along which a non-duplicate TREE-CREATE message was received and every other link along which no TREE-CREATE-NAK message was received to be part of the tree for a specific group. (Core considers every link adjacent to it to be part of the tree). Note that all these tree links are bi-directional tunnels.

The choice of using ACK or NAK in response to the TREE-CREATE messages is dictated by whether robustness or saving bandwidth is more important. If an ACK-based (positive acknowledgement) scheme is use, then data may not be delivered along links where ACKs were lost. This results in loss of data, but no wasting of bandwidth. However, if a NAK (negative acknowledgement) based scheme is use, loss of NAKs can only result in some data being forwarded more than once (which is discarded by the downstream node on reception).When data arrives

at a node along one of the tree links, the node forwards the data along all other tree links. However, if data arrives along a non-tree link a TREE-CREATE-NAK message is (again) sent back along that link and the data is discarded.

2.4.3 Transient Loops

The tree created by the <n>th TREE-CREATE message might not be the same as the one created by <n-1>th message. A situation may exist where some nodes are forwarding data according to the older tree and some according to the newer tree, which may result in loops or data loss. Such a phase is to be expected due to the dynamic nature of ad hoc networks. However it is considered to be transient and AMRoute recovers from it as soon as the network reduces its dynamicity.

2.4.4 Node Failures and Group Leaves

Nodes leaving a group or node failures are only partially handled by the redundant links in the mesh. In some situation, node failures might result in splitting the mesh into multiple disjoint meshes, where only one of these meshes has the core. Each node in the mesh expects to periodically receive TREE-CREATE messages. In case this message is not received within a specific timeout, the node designates itself to be the core after a random time. The node whose timer expires the earliest succeeds in becoming the core and initiates the process of discovering other disjoint meshes as well as tree creation. Multiple cores that may arise in this case are resolved by the core resolution procedure.

2.4.5 Picking which branch to use for the Tree

There are several possible algorithms that can be used to decide which mesh branches to use for the tree. The simplest approach is to simply accept the first TREE-CREATE message that is received and discard and duplicate TREE-CREATE message. This results is a reasonable tree, but it is not necessarily the most bandwidth efficient (e.g., using minimum number of total hops) or lowest latency. We are therefore investigating the tradeoff of increasing the complexity of branch selection in order to improve bandwidth efficiency or reduce latency. To improve bandwidth efficiency a node can select the branch from which it received a TREE-CREATE from its closest neighbor(based on the TTL value in the outer IP header). However, in order to prevent the tree from becoming broken (not connecting all group members), the algorithm must

- be able to tell when the TREE-CREATE message has been received before, and
- not change the initial selection until the next round of TREE-CREATE messages are received.

To detect duplicates it necessitates the use of a path-vector field in the TREE-CREATE message. Also the duplicate messages can be identified with the help of a sequence no field in the packet. Every time the core sends a TREE-CREATE packet at increments the sequence no field. Initially the value is 0. Each time a packet is duplicated it's sequence No is changed.

The following figure explains the above statements.

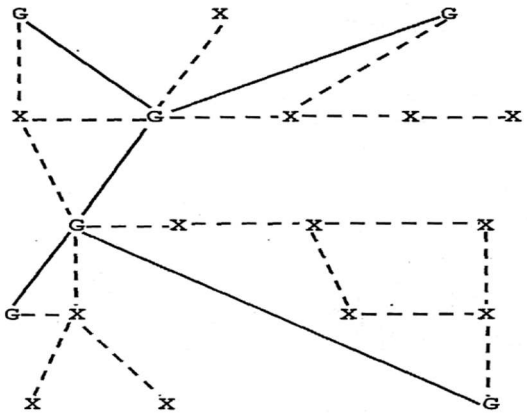

FIGURE 2.4: Virtual User-Multicast Tree

G = group member router (AMRoute capable)
X = non-member router (need not be multicast or AM Route capable
- - - - - --- = Link
_____ = Virtual user-multicast tree

These are the various steps involved in creating a tree in the AMRoute Protocol.

2.5 Implementation Details

The implementation details include the various types of messages used and their formats. These messages and their formats are given below.

2.5.1 JOIN_REQ message:

The logical core node periodically to all the nodes nearer to it broadcasts this message. These nodes forward the message to their neighbors and so on. The structure of the packet is as follows.

Table 2.1: Join_Req

Version	Message-ID	Unused	Initial-TTL
Source IP ADDRESS		IP Multicast Address	

2.5.2 JOIN_ACK message:

A node in response to a JOIN_REQ from a logical core generates this. This message indicates that the node is interested in becoming a member of the group.

Table 2.2: Join Ack

Version	Message-ID	Unused	Initial-TTL
Source IP ADDRESS		IP Multicast Address	

2.5.3 JOIN_NAK message:

A node in response to a JOIN_REQ from a logical core generates this. This message indicates that the node is not interested in becoming a member of the group.

Table 2.3: Join-Nack

Version	Message-ID	Unused	Initial-TTL
Source IP ADDRESS		IP Multicast Address	

2.5.4 TREE_CREATE message:

The logical core node generates this message. This is used to create a tree out of the mesh. The logical core node generates this message periodically.

Table 2.4 Tree_Create

Version	Message-ID	SEQ-NO	Initial-TTL
Source IP ADDRESS		IP Multicast Address	

2.5.5 TREE_CREATE_NAK message:

The logical core node in response to the TREE_CREATE message generates this message. This indicates that the link cannot be used as a tree link.

TABLE 2.5 TREE CREATE NAK

Version	Message-ID	SEQ-NO	Initial -TTL
Source IP ADDRESS		IP Multicast Address	

Field DescriptionVersion: This field represents the version of the AMRoute protocol.

Message-ID: This field is used to identify the type of packet.

Message-ID = 1, a **JOIN_REQ** packet

Message-ID = 2, a **JOIN_ACK** packet

Message-ID = 3, a **JOIN_NAK** packet

Message-ID = 4, a **JOIN_CREATE** packet

Message-ID = 5, a **JOIN_CREATE_NAK** packet

Message-ID = 6, a **Data** packet

Initial TTL: This field indicates the TTL value (i.e.,) the no. of hop counts allowed to that particular packet. This field when used along with the TTL value in the IP header can be used to find out the remaining hops for the packet.**Source IP Address:** This field indicates the IP address of the source of that packet. The node, which is sending the packet, fills up this field with its address.

IP Multicast Address: This field indicates the multicast group address. This is used to distinguish from one group address to another group address.

SEQ_NO: This field is used to identify the duplicate TREE_CREATE message. For the first message this field is set to zero and for every duplicate message this is incremented by one.

Path Vector: This is used to distinguish among replicated versions of the same TREE_CREATE messages when more bandwidth or latency optimal trees are desired. Initially it is set to 0. Each node that performs any replication modifies the value at each replication.

These are the various message formats and their descriptions.

Timer: A logical core keeps two timers, namely the JOIN-REQ-SEND timer and the TREE-CREATE-SEND timer. The expiry of JOIN-REQ-SEND causes the node to compute the new TTL value to use for the expanding ring search, broadcast a new JOIN-REQ with this TTL value and reset the timer. The TREE-CREATE-SEND timer is kept to send out periodic TREE-CREATE messages. A non-core member uses a TREE-CREATE-RCV timer. When it expires, the node waits for some random amount of time before it resets itself to be a core, and starts sending out JOIN-REQs and TREE-CREATEs. This period is set to be random to prevent multiple non-core nodes from becoming cores simultaneously.

Data structures

Each member keeps two tables, each containing a set of neighbors, the neighbors on the mesh and the neighbors on the multicast tree. Note that these neighbors are connected by unicast tunnels rather than being physical neighbors. The member also keeps a “hop-count” associated with each mesh link. This hop-count is obtained at the

time of mesh link creation, and is updated by the periodic control messages (control messages have original TTL value used by source in the headers). A node tracks the ID of the logical core it currently recognizes, which can be itself. We allow the existence of multiple logical cores in a same group. However, each node only recognizes one logical core at any instant. This information is updated when the node receives a TREE-CREATE from a logical core it did not recognize, or when the node resets itself to be core, as described in detail in the next section.

So the data structures involves basically a Mesh table and a Tree table. These include the corresponding neighbors either of the tree or of the mesh. Also many other details can be stored in this table. This table is implemented using a singly linked list data structure. This is chosen because of the flexibility in inserting or removing an entry from the table. Also the complexity involved is less in this case.

2.6 MODULES

The various modules in the protocol are as follows.

2.6.1 Module 1:

Receiving Control packets:

This module deals with those routines, which are to be executed on receiving some of the control packets. They are as follows:

Receiving a JOIN_REQ message:

Core Node:

- If this core itself (because of the broadcast medium) transmitted the JOIN-REQ, it is ignored.
- If this core is in the same group as the JOIN-REQ indicates and does not have a mesh or tree link to the source of the JOIN-REQ, it returns a JOIN-ACK and drops the JOIN-REQ. (Note that the JOIN-ACK is unicast to the source of the JOIN-REQ). The core marks the source of the JOIN-ACK as its mesh neighbor.
- Otherwise the core decrements the TTL of the JOIN-REQ and re-broadcasts it.

Non-core Node:

- If this JOIN-REQ comes from a logical core that this node recognizes (which means they are already on the same mesh), decrement the TTL value of the JOIN-REQ and rebroadcast it.
- If this JOIN-REQ comes form a different logical core from that recognized by this node, the node returns a JOIN-ACK to the source and marks the source as a mesh neighbor.
- If this JOIN-REQ is for a different group, the node decrements the TTL, of the JOIN-REQ and re-broadcasts it(acting as an intermediate router).

Receiving a JOIN_ACK packet

Core Node:

If the JOIN-ACK is responding to a JOIN-REQ sent out by this core, the logical core marks the source of the JOIN-ACK as its mesh neighbor and drops the JOIN-ACK.

Non-Core Node:

A non-core member will NEVER receive a JOIN-ACK message since JOIN-ACK is sent to the source of a JOIN-REQ, which can only be a logical core. Hence any JOIN-ACK received is dropped.

Receiving a JOIN_NAK Packet

Core Node:

The logical core deletes any existing mesh and tree links with the source of the JOIN-NAK.

Non-Core Node:

The node deletes any existing mesh and tree links with the source of the JOIN-NAK.

Receiving a TREE_CREATE Packet

Core Node:

- A logical core can only generate TREE-CREATE messages.
- On receiving the message, the logical core uses the core resolution algorithm to determine the winner. If the core itself wins, the TREE-CREATE is dropped. If the source of this message wins, the receiving node becomes a non-core member and transitions into the NON-CORE State. The receiving node also marks the winning core as a mesh neighbor, and recognizes it as the new logical core. The TREE-CREATE message is forwarded.
- Onto downstream mesh links, which are then marked as tree links?

Non Core Node:

- If the source of this TREE-CREATE is the same core as that recognized by this node, and the message has a new sequence number (not a duplicate), the node marks the incoming link as a tree link. It also forwards the message to neighbors along its downstream mesh links and marks them as tree links.
- If this TREE-CREATE is a duplicate from the same core as that recognized by this node, the node marks the incoming link as a mesh link. It also sends a TREE-CREATE-NAK message to the upstream node and drops this TREE-CREATE.
- If this TREE-CREATE comes from a different logical core from that recognized by this node, it runs the core resolution algorithm to decide the new core. If the new core wins, the node sets the CORE ID to the new core, forwards

the TREE-CREATE onto all its downstream mesh links, and marks the incoming and outgoing links as tree links. If the source loses in the core resolution, this TREE-CREATE is dropped.

The TREE-CREATE path vector is modified if a node performs does any replication. The node reads the current content of path vector looking for the least significant "1" (Initially the path vector is set to zero).

Received a TREE_CREATE_NAK:

Core Node:

The incoming link is marked as a mesh link, instead of a tree link, and the message is dropped.

Non Core Node:

The incoming link is marked as a mesh link, instead of a tree link, and the message is dropped.

Timer Events

JOIN_REQ Timer

This message is used to detect other group members. An expanding ring search is used whereby successive JOIN-REQs are broadcast with increasing TTL values every JOIN-REQ-SEND time units. The TTL value can only be increased to a specific upper bound, after which all JOIN-REQs are broadcast using the same value.

TREE_CREATE Timer

This message is used to build the multicast tree. This message is transmitted over the existing mesh links, i.e., this message is sent through unicast tunnels to mesh neighbors. On transmission of a TREE-CREATE, logical core considers its mesh neighbors as its tree neighbors also until informed otherwise by a TREE-CREATE-NAK. Note that when a node joins a group and sets itself to be a logical core, it has no mesh neighbors prior to receiving a JOIN-ACK or JOIN-REQ. Therefore, the TREE-CREATE message is not transmitted until one or more mesh neighbors exist.The two timer events and Receiving the control packets or the data packets are executed simultaneously. There are three processes, each of which executes its routine independently.

2.6.2 Module 2:

Sending Data Packets

This module is used to get data from the user and the data is sent from a separate program. Hence the two programs are to be executed separately. Message Queues are used to establish Inter Process Communication (IPC CALLS) between the two modules. Whenever the data is to be sent by the user, the user types the data and then this module will check if there are any members in the tree. If there is a node a message queue is used to transfer the data from one routine to another routine. After this the module sends the data to the address specified in the queue. Thus this module is used to send data from the user.

2.6.3 Module 3:

Tree Table:

A tree table is actually a Linked List, which is used to store some information Such as the IP address of the neighboring node the number of hopcounts required to reach the corresponding node and some of the other entries like the core nodes id total no of nodes in the tree etc.,

This module is used to perform the following operations.

- To add an entry to the tree table
- To remove an entry from the tree table
- To search for the given entry in the tree table
- To display the contents of the tree table

2.6.4 Module 4:

Mesh Table

A mesh table is also implemented using a linked list and it is used to store information about the neighboring nodes in the mesh. These informations include IP address of the neighbor, Hopcount required to reach him, Total no. of neighbors and the current core node.

The manipulations made in the mesh table are same as those done in tree table.

- To add an entry to the mesh table
- To remove an entry from the mesh table
- To search for the given entry in the mesh table
- To display the contents of the mesh table

3. RESULTS

The below result screens explain the entire process in fully qualified manner.

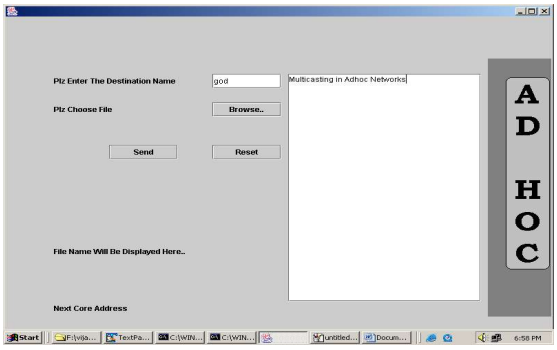

Fig 3.1 Source node sending Message frame

Journal of Theoretical and Applied Information Technology
31st May 2022. Vol.100. No 10

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

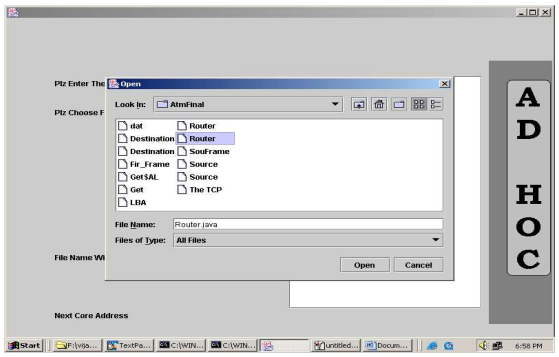

Fig 3.2 Browsing the FILE message to be send to destination

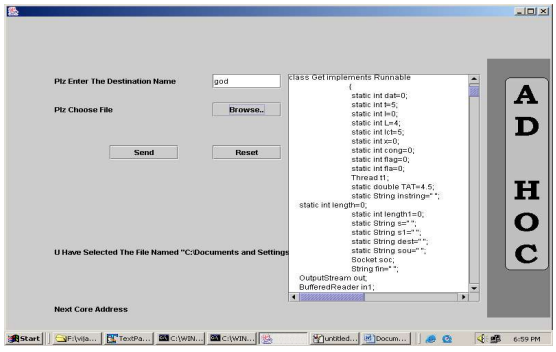

Fig 3.3 sending File

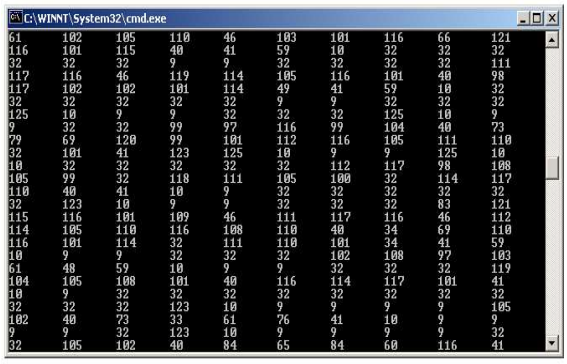

Fig 3.4 Encrypted Message

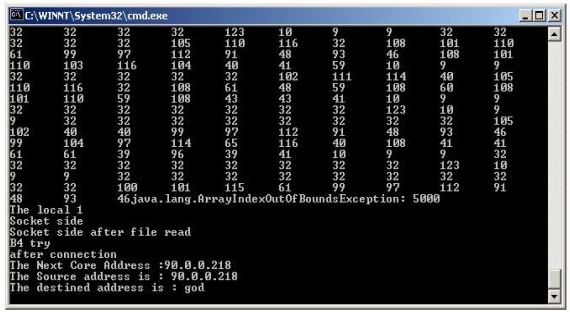

Fig 3.5 Encrypted Message

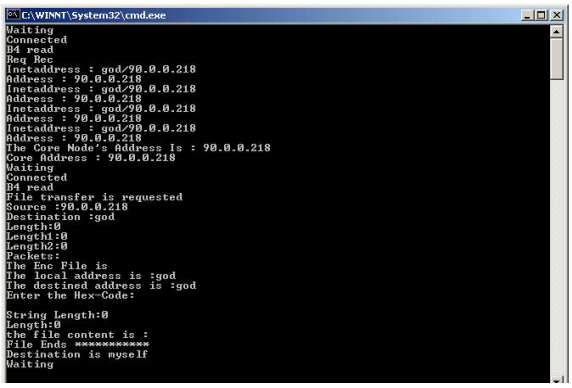

Fig 3.6 Finding Core node to send the data

Fig 3.7. Decrypting the message (file)

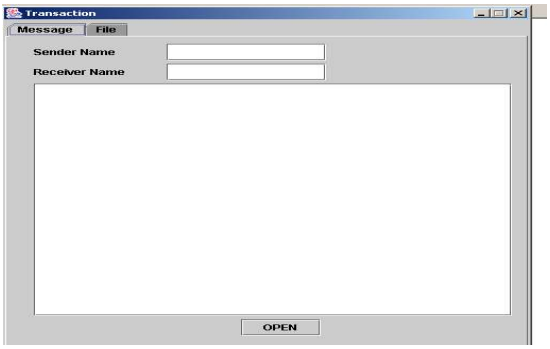

Fig 3.8 Destination frame

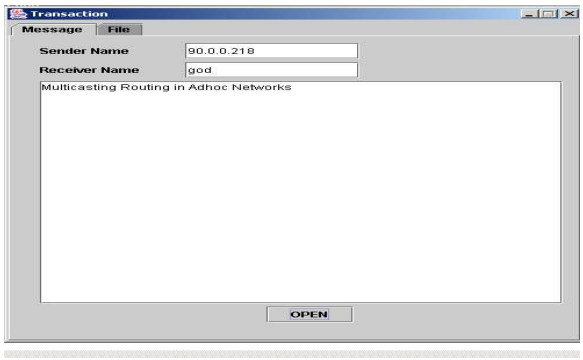

Fig .3.9 receiving the message (file)

4. CONCLUSION

Analysis of earlier Adhoc routing protocols states that existing multicast protocols do not work well in ad hoc networks as the frequent tree reorganizations can cause excessive signaling overhead and frequent loss of datagram and core node is not identified to pass the segment. The solution for this is to design a routing protocol that involves in first Finding core node and then send each through it. Message Queues are used to establish Inter Process Communication (IPC CALLS).

The **Optimal Adhoc Multicast Routing Protocol (AMRoute)** presents a novel approach for robust IP Multicast in mobile ad-hoc networks by exploiting user-multicast trees and dynamic logical cores. It creates a bi-directional, shared tree for data distribution using only group senders and receivers as tree nodes. Unicast tunnels are used as tree links to connect neighbors on the **User-multicast tree**. Thus AMRoute does not need to be supported by network nodes that are not interested/capable of multicast, and group State Cost is incurred only by group senders and receivers. Also, the use of tunnels as tree links implies that tree structure does not need to change even in case of a dynamic network topology, which reduces the signaling traffic and packet loss. Thus AMRoute does not need to track network dynamics; the underlying Unicast protocol is solely responsible for this function. AMRoute does not require a specific Unicast routing protocol; therefore, it can operate seamlessly over separate domains with different Unicast protocols. We have tried to overcome the transient loops in the mesh creation. The output achieved shows that the protocol worked well and I conclude that this protocol can be outperform well than earlier protocols. The project achieved its main goal 'To perform multicast routing in adhoc networks'.

5. FUTURE WORK

AMRoute showed some promise in its simplicity and scalability in the number of senders. However, the presence of unidirectional "critical" links prevented reliable data delivery. The problem became worse as mobility was increased. Other drawbacks of AMRoute were the existence of loops and inefficient formation of trees. A possible improvement for AMRoute is to take reachability information (i.e., packets sent to neighbor/packets received from neighbor) into account when selecting tree links. Using this method, the impact of unidirectional critical links can be reduced. In addition, introducing adaptively into the protocol can build more optimal trees most importantly, a loop prevention mechanism must be utilized for AMRoute to be efficient.

REFERENCES:

[1] Bommaiah, McAuley, Taplade and Liu ""AMRoute:Adhoc Multicast Routing Protocol", draft-talpade-manet-amroute-00.txt, August 6, 1998

[2] Ballardie T., "Core based Trees(CBT) Mulitcast Routing Architecture", RFC 2201, September, 1997.

[3] Deering, S., et al, "Protocol independent Mulitcast Sparse Mode (PIM-SM): Motivation and Architecture", Internet Draft, draft-ietf-idmr-pim-arch-04.txt, October, 1996.

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

[5] 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.

[6] Perkins, C., "Mobile Ad hoc Networking Terminology", Internet Draft, draft-ietf-manet-term-00.txt, October, 1997.

[7] Sung.Ju Lee, William Su, Julian Hsu, Mario Gerla, and Rajive Bagrodia "A Performance Comparision Study of Ad Hoc Wireless Multicast Protocols"
[8] "Unix Network Programming" by Richard Stevens.
[9] "The Desing of Unix Operating Systems" By Maurice J Bach.
[10] "Design of Unix Operating System" by Maurice J Bach.