A MODULAR MULTICASTING ALGORITHM WITH A CONTOUR DISCOVERY APPROACH

by

ALBERTO APONTE

A Dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2006

UMI Number: 3232036

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Professor Tarek Saadawi

9-18-2006

Date | **Chair of Examining Committee**

Professor Gabriel Tardos

9-18-2006

Date | **Executive Officer /Deputy**

Professor Joseph Barba

Professor Umit Uyar

Professor Jizhong Xiao

Professor Mehmet Ulema

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Acknowledgements

I wish to express my sincerest gratitude to Professor Saadawi for his advice and guidance during my journey through this wonderful world of research; his time and support were indispensable. I would also like to thank Professor Barba for opening the doors of research to me and waking me up to the importance of computers.

This work is dedicated to:

My mother Adela, who guided by her maternal instincts and love gave me and taught me everything, I miss her.

My wife Dora, who patiently stayed home during so many long evenings and weekends taking excellent care of our kids, I love you.

My kids Elisa and Juancarlos who always understood why Dad was not home, I love you guys.

A special thanks to Osama Hussein, and Lisa Hu for providing me with the "kick start" during my early days with Opnet.

Abstract

A modular Multicasting Algorithm with a Contour Discovery Approach

by

Alberto Aponte

Adviser: Professor Tarek Saadawi

This thesis presents an innovative approach to multicasting algorithms design. Our approach herein is based on two main components; the first is our proposed contour discovery algorithm, while the second is the proposed multicasting algorithm that is based on the contour discovery algorithm.

We develop an algorithm for finding the shortest distance -in terms of the number of hops- from a node identified as a source to unknown nodes located by the outskirts of a network. Several distances are found through each neighbor of the source node to determine the contour of the network. The accuracy of the found contour relies on our outward propagation mechanism which propagates discovery packets strictly in an outward direction relative to the source node. This algorithm is based on the deployment and return of discovery packets.

Based on the found contour we develop a Distributed Multicast Algorithm which uses that information to create modules on appropriate regions of a network. This algorithm deals with the following issues: (a) neighbor discovery, (b) neighborhood exchange, (c) multicast advertisement, (d) multicast registration, and (e) data distribution. The main goal of this algorithm is to minimize the multicast delivery delay, that is, the time between the beginning of the reception of a packet by the first group member and the end of the reception of the same packet by the last member.

Performance metrics covered include coverage time, multicast delivery delay, and network congestion. Performance evaluation through simulation observations show that the probability of missing a node during the discovery process is near zero for several topologies with different network sizes and different origination points. It is also shown that the multicast delivery delay in our distributed multicasting decreases on the average by a factor of 0.6 compared to that in non-distributed multicasting when the two algorithms are tested on the same topologies and network sizes. On the other hand, the packet delivery ratio, that is, the number of multicasting packets delivered versus the number of multicasting packets supposed to be received is near one for moderate multicast group sizes and data rates. Other outcomes achieved by the implementation of modularity are: Distribution of acknowledgement implosion among several nodes in the network, obtain multicast source registration, and avoid broadcast storming.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Multicasting is a communication process where a single source transmits user data to more than one receiver. In a multicast network, sources send data traffic to a group of hosts represented by a multicast group address. A multicast wireless host cannot base its forwarding decision on a destination address; a host in a wireless network must forward multicast data in a broadcasting mode in order to reach all possible receivers.

In order to forward multicast data to appropriate receivers, multicast routing algorithm protocols have to deal with many of the same issues as a unicast routing algorithm, such as learning the inter-network topology, detecting changes in the topology, and computing delivery paths in the topology. Multicast forwarding makes use of the source address, while unicast forwarding makes use of the destination address. Forwarding mechanisms of multicasting are based on the source address, and to prevent loops, packets are discarded if they do not arrive from a designated forwarder host.

The goals of a multicast algorithm are: i) minimize the number of steps needed to complete the multicast process, and ii) minimize the multicast delivery delay, i.e., the time between the beginning of the reception of a packet by the first member and the end of the reception of the same packet by the last member. In our multicast distributed algorithm we emphasize on the reduction of the multicast delivery delay by creating techniques which reduce this delay to values that are on the average 40% below those values found in other non-distributed algorithms.

A multicast algorithm may be state based [38]. However these types of algorithms (either hard or soft) are not suitable for wireless networks because these are characterized by their random mobility. In some multicast algorithms for wireless networks, states or

tables are kept for very short periods of time, therefore they are considered stateless. In order to deliver the multicast data, some protocols, such as the On-Demand Multicast Routing Protocol (ODMRP) [12], construct a multicast mesh. Others, such as the Multicast Ad-hoc On-Demand Distance Vector protocol (MAODV) [13], construct a Distributed Shared Tree. The Distributed Multicast Algorithm presented herein construct distributed trees to keep redundancy factors as low as possible. Redundancy is defined as the ratio of the number of nodes visited once or more than once versus the number of nodes visited only once. Furthermore, the Distributed Algorithm for Multicasting proposed herein does not depend on any unicast routing protocol; it is capable of discovering all nodes interested in the communication process and deliver the multicast data to them. Our multicasting algorithm relies on a discovery algorithm which is also developed in this work; this discovery algorithm finds the distances from a source node to nodes located by the outskirts of the network by using our own outward propagation mechanism. These two algorithms working together allow us to reduce the multicasting delivery delay, and maintain a near one value for the packet delivery ratio while maintaining moderate values of group members and user data rate. Some issues inherent to multicasting such as network discovery, broadcast storm, acknowledgment implosion, source registration and unfair delivery delay, along with proposed solutions are presented in the following sections 1.1 through 1.5.

The rest of this work is organized as follows: chapter two describes some multicast techniques used in wired and wireless networks. Chapter three describes the first part of our original work which is the design of an algorithm to discover the contour of a network and the use of these findings to create modularity; simulations and obtained

results such as network coverage, and redundancy are presented at the end of the chapter. Chapter four describes the second part of our original work which is the design of a multicasting algorithm capable of advertisement, registration, and data distribution; simulations and obtained results such as multicasting delivery delay, and data delivery ratio are presented at the end of the chapter. Chapter five discusses some required mechanisms to control and maintain the performance of the algorithms presented in chapters three and four. Chapter six presents some of the processes and projects used to test the validity of the algorithms developed in chapters three and four. Chapter seven presents a summary and future work.

1.1 MULTICASTING ISSUES

Regardless of the network application, multicasting communications [41] provide an efficient way to support the dissemination of data from a sender to a group of receivers. However, distributing data in applications such as financial information requires a fair multicast delivery delay, data distribution in other applications such as software distribution, and billing records require reliable delivery, while multimedia communications [35] require quality of service [29]. These types of problems, and some others related to bandwidth consumption, and network unpredictability are presented along with a proposed solution in the following sections.

1.1.1 NETWORK SIZE AND CONTOUR DISCOVERY

The components and resources of mobile wireless networks are not known in advance. Their characteristics can change in an unpredictable way, communication

capabilities can break down, and therefore necessary services can be become unavailable at any time. Communicating in mobile networks is a complex problem, because networks can become very large, and mobility and battery usage can make the components of the network unpredictable. Discovery packets are tools that can be used periodically to explore the network to determine the number of nodes in different directions. Figure 1.1 shows the trajectory followed by discovery packets from a source node to the outskirts of the network.

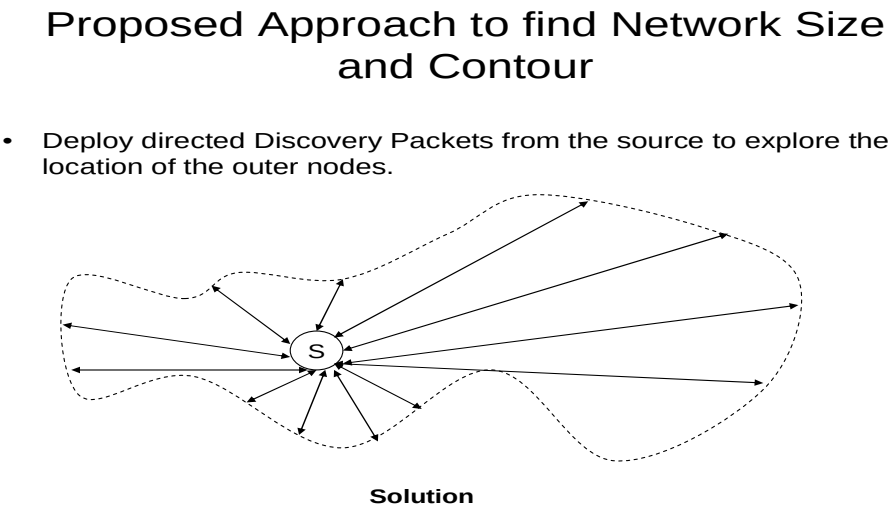


Fig. 1.1: Discovery packets travel to the perimeter of the network and return to the source

1.1.2 BROADCASTING

Multicast communications involve flooding and broadcasting [36], especially when node mobility needs to be taken into consideration. These processes create contention and collision problems, and the redundancy involved leads to poor scalability. The problem of serious redundancy creates what is referred to as the Broadcast Storm Problem.

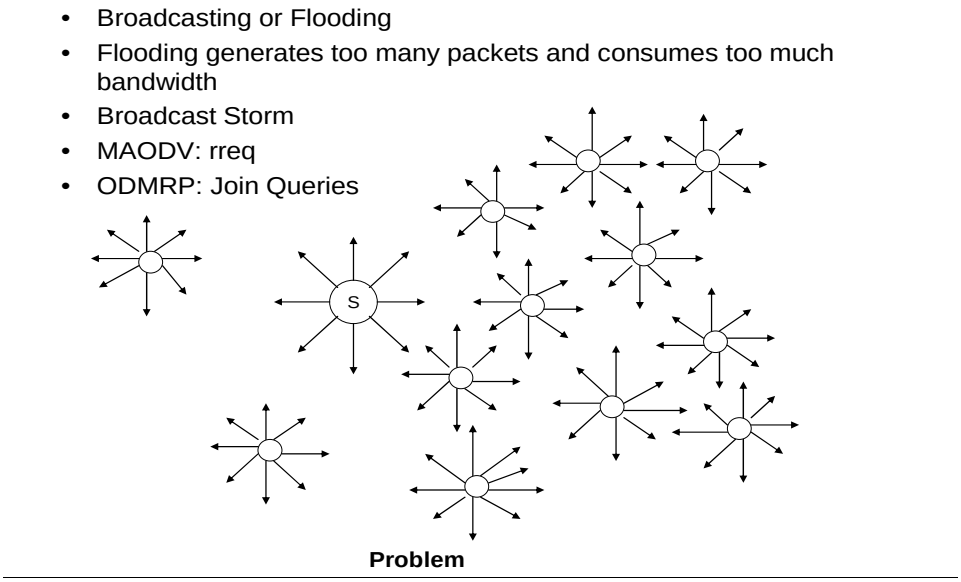


Fig. 1.2: Every node in the network re-broadcasts received packets

Multicast protocols such as Multicast Ad Hoc On Demand Distance Vector broadcasts route requests, while the On Demand Multicast Routing Protocol broadcasts join requests. In the Distributed Algorithm for Multicasting presented here, neither flooding nor broadcasting are used during the discovery, advertisement or registration sessions. Instead, the information originates at a multicasting source and is passed from node to node in a sequential manner. Figure 1.2 shows nodes re-broadcasting received packets. Figure 1.3 shows the use of controlled broadcasting to reduce the redundancy of control packets and improve the bandwidth efficiency.

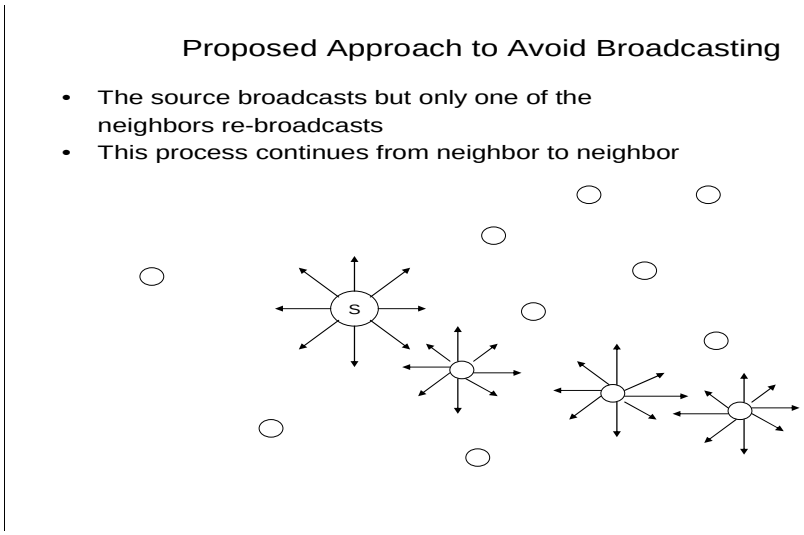


Fig. 1.3: Only designated nodes re-broadcast packets

1.1.3 SOURCE REGISTRATION

Unreliable multicast algorithms [28] delivering data throughout distribution trees do not expect any acknowledgement from the group members. As a result, the multicast source does not have a record of the actual registered members receiving data. In the distributed algorithm proposed herein, the source registration of group members is distributed among all registration center nodes and the source communicates only with those nodes.

Figure 1.4 shows a multicast source node (S), a few group members (M), and some nodes being used as forwarders. As data is propagated through forwarders to the member nodes, the source does not get information about the number of group members or their identification.

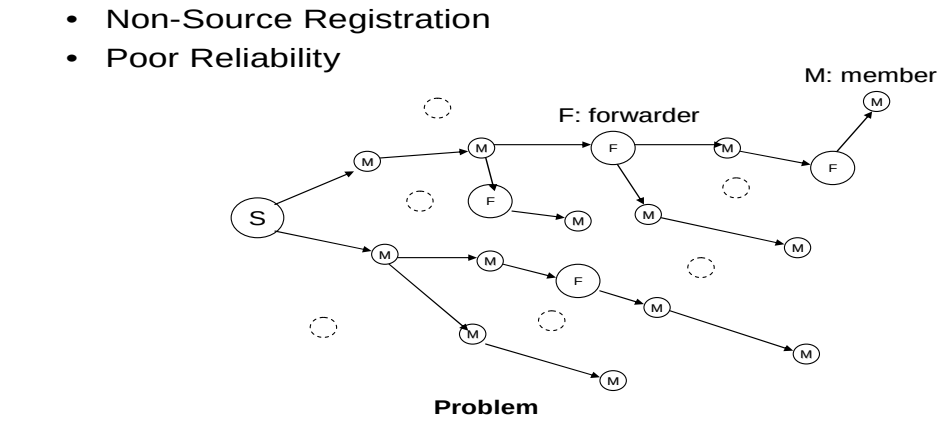


Fig. 1.4: A multicast source does not get registration information

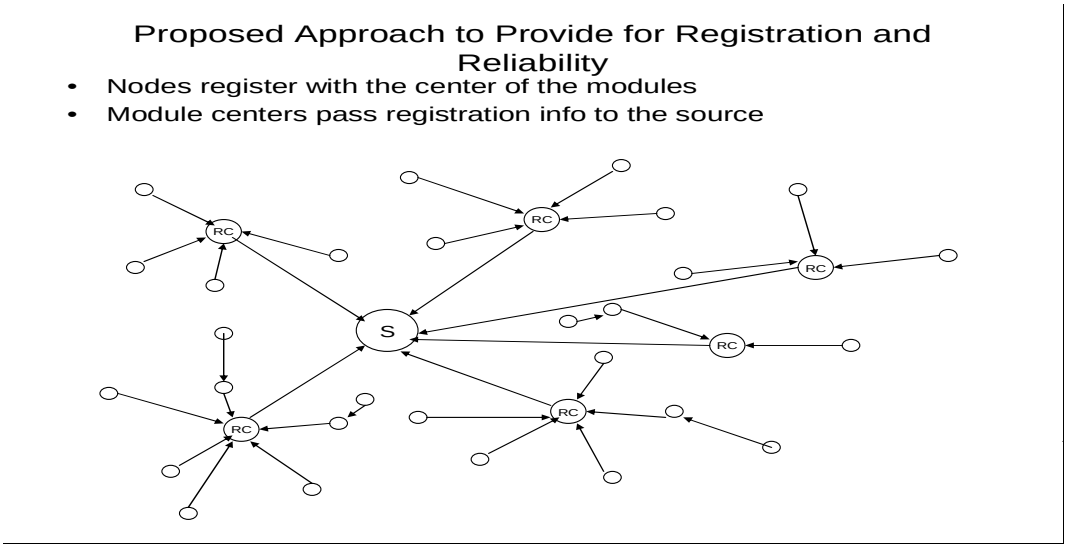


Fig. 1.5: The source gets registration information from registration centers

Figure 1.5 shows a source node (S), and several registration centers (RC) nodes. Nodes around a registration center can register for the multicast session locally at the closest center, and the multicast source communicates only with those centers.

1.1.4 ACKNOWLEDGEMENT IMPLOSION

Reliable multicast protocols [24] [26] work efficiently in small area networks but they don't scale well in large area networks. A large number of group members sending acknowledgements or data requests to the multicast source create what is referred as acknowledgement implosion. This implosion [43] does not allow the implementation of reliability. Figure 1.6 shows a multicast source (S) and a large number of member (m) nodes returning acknowledgement packets.

The distributed algorithm proposed herein is reliable for multicasting. The implementation of modularity allows for acknowledgement return packets to be distributed among all registration centers. Figure 1.7 shows a multicasting source (S), a few registration centers (RC), and some group members returning acknowledgement packets to their respective centers. The return of packets confirming the delivery of data implements reliability.

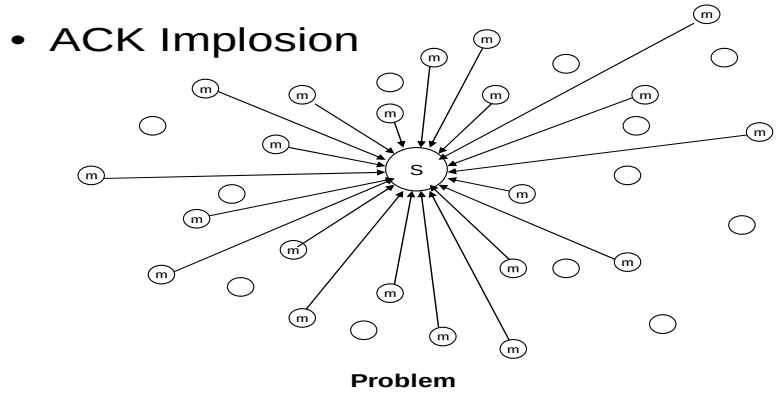


Fig. 1.6: Non-Distributed Acknowledgement Implosion

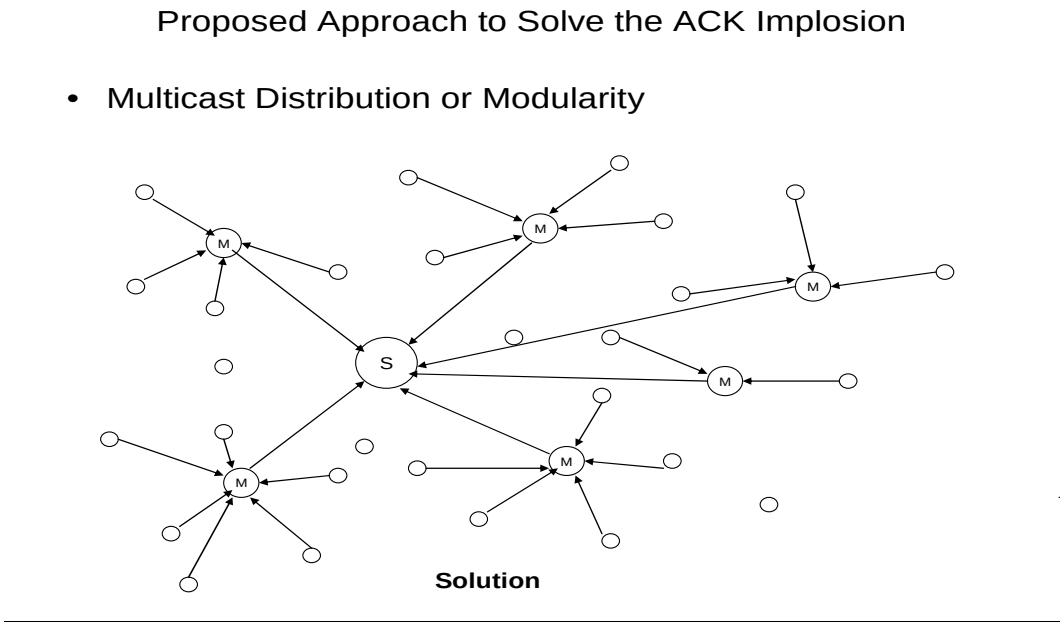


Fig. 1.7: Distributed acknowledgement to avoid implosion at the source

1.1.5 MULTICAST DELIVERY DELAY

We have defined the multicast delivery delay as the time between the beginning of the reception of a packet by the first group member and the end of the reception of the same packet by the last member. Obviously, nodes located near the multicast source will receive data information before nodes located at a long distance. This situation is depicted in Figure 1.8. The delivery time is also affected by the number of nodes in between the source and the group member. The approach proposed in this work considers the selection of a node in the middle of a path between the first and last node from a source and distribute the data from this middle node. This procedure will effectively reduce the multicast delivery delay by a factor of 0.5. Chapter 4 shows this procedure in fine detail.

Figure 1.9 shows the comparison of the delivery delay when data is multicast in sequence from the first to the last node, against the delivery delay when data is sent to a node in the center of the network.

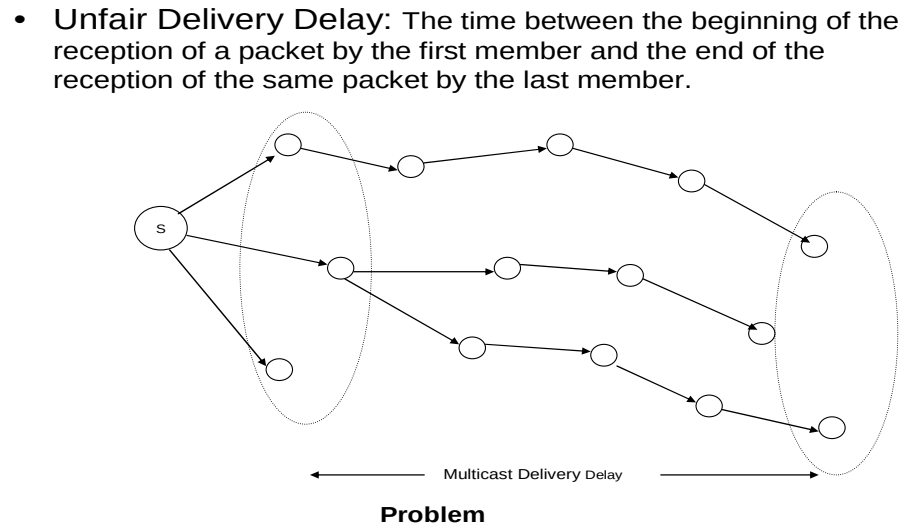


Fig. 1.8: Unfair multicast delivery delay in Non-Distributed Multicasting Algorithms

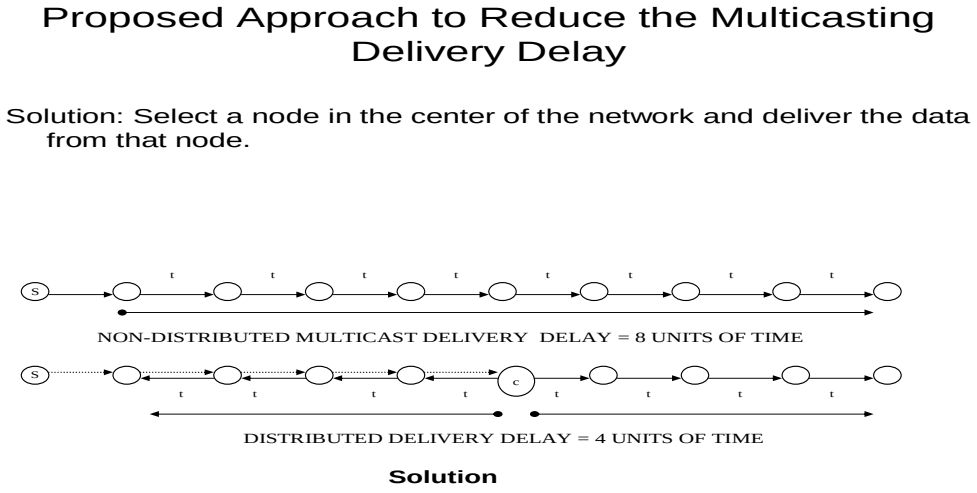


Fig. 1.9: Multicast delivery delay comparison between Non-Distributed and Distributed Algorithms

2. MULTICASTING COMMUNICATIONS

2.1 MULTICASTING MODES

2.1.1 RECEIVER BASED MULTICASTING MODE

Receiver based multicasting uses a multicast source distribution tree where the root is the source node generating the data to be multicast, and the branches are formed by either routers or receivers. In this type of multicasting the sender does not need to maintain a list of receivers. The routers save only their next receivers in the tree; this makes it easier to make groups dynamic, letting receiving members join and leave multicast sessions without the need to update the sender information.

A simple source tree is shown in Figure 2.1.

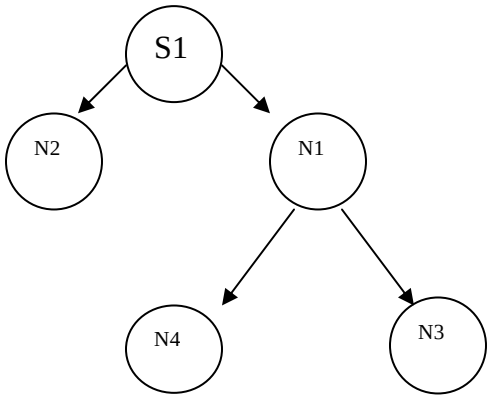


Fig. 2.1: Simple Source Tree for Multicasting

A multicast session can have more than one source with each source forming a separate distribution tree. Any node in the network with the exception of sources can be a receiver for the multicast session or it can be a forwarder for the multicast traffic. This type of

distribution trees is used for dense networks where group members are densely distributed across the network. Protocols used in this type of networks use an approach called data-driven (flood and prune) to construct the multicast distribution tree.

A network with two sources, five receivers, and a forwarder is shown in Figure 2.2.

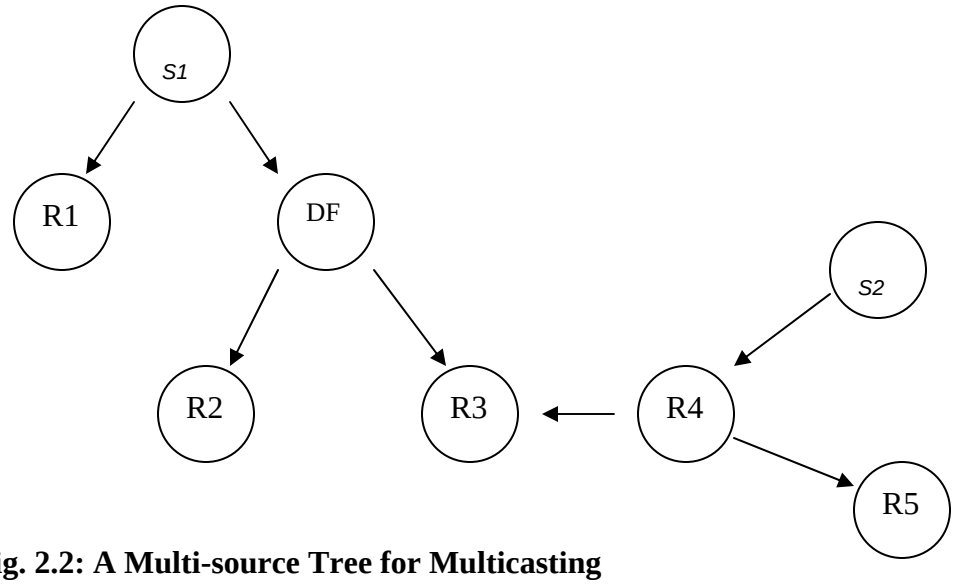


Fig. 2.2: A Multi-source Tree for Multicasting

2.1.2 SENDER BASED MULTICASTING MODE

Sender based multicasting mode uses the multicast shared distribution tree where the root of the tree is not at the source, instead it is located at a selected node in the network. There are many algorithms that can be applied for the best selection of this node based on its location. This node is often called RP (rendezvous Point or core). The multicast traffic for the session is sent from the source or sources to this node and then it is distributed from here to the receivers or to the designated forwarders. Sources can unicast the traffic to the RP if they know its address and location; otherwise they have to discover the route to it by using a unicast routing table. In this mode of multicasting every new member has

to notify every potential sender within the group to be added to the recipient list. This approach works well for small groups, but scales poorly to large and dynamic groups.

In Figure 2.3, sources S1 and S2 send multicast traffic to the RP node (root).

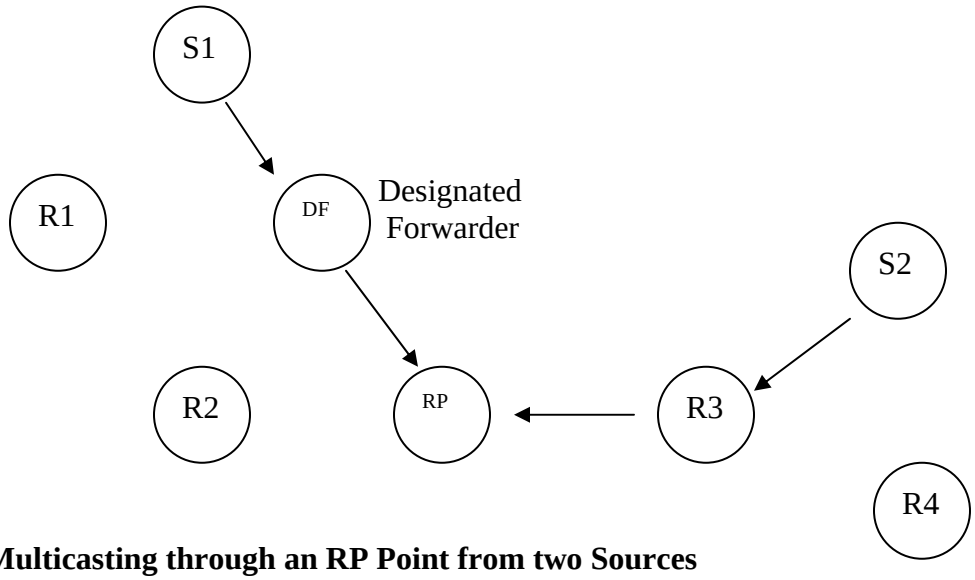


Fig. 2.3: Multicasting through an RP Point from two Sources

The shared tree is formed from this root to the receivers as shown in Figure 2.4. It is assumed that links are bi-directional; this allows the traffic to flow down from S1 to the RP on the way to the root and up from the RP to the S1 on the way to R1.

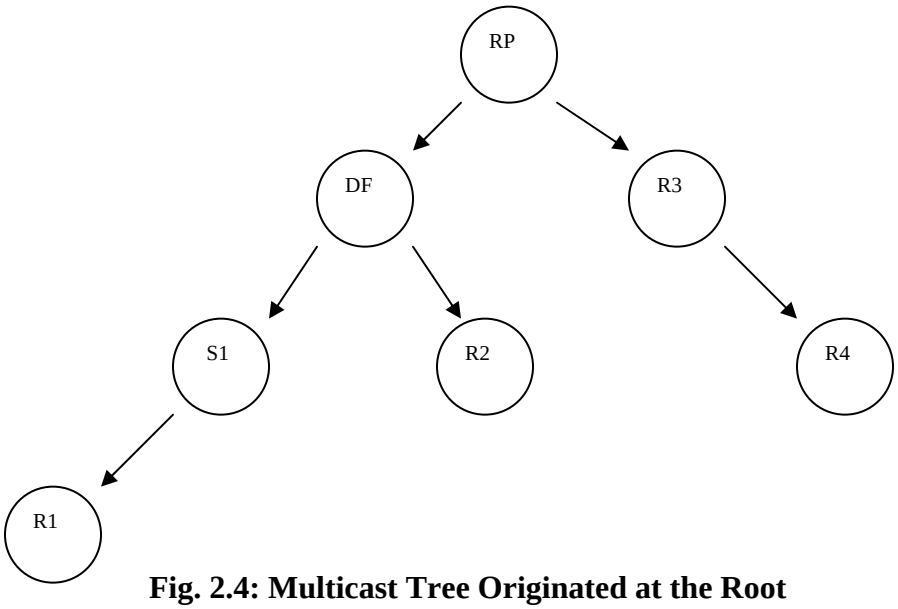


Fig. 2.4: Multicast Tree Originated at the Root

An interesting situation arises in the topology shown in Figure 2.3 and the shared tree in Figure 2.4. S1 is a source of multicast traffic and as such it sends its data to the root. The root then returns the same traffic to S1 to be delivered to receiver one (R1). This flow of traffic introduces a delay in the traffic to R1 and also consumes unnecessary bandwidth; this problem can be solved by making R1 to join the multicast session from S1 directly and pruning itself from the root.

2.2 MULTICASTING MECHANISMS

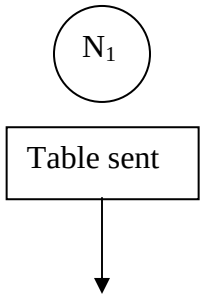
2.2.1 MULTICAST FORWARDING TABLE FORMATION

As a node discovers a neighbor, its address is added to a table; after all neighbors have been discovered, a table with their addresses is formed and maintained at that node. This table is kept until a packet discovers that a node has moved away from the neighborhood or a new node has entered into the neighborhood. The exchange of these tables is necessary for building and maintaining the distribution tree, that is, to keep the connectivity among nodes in the tree. Figure 2.5 shows the exchange of forwarding tables and the creation of designated forwarders.

The last table after being received by node N_1 allows the designation of this node as the multicast forwarder of node N_2 for traffic coming from source one and two. In other words, this table states that the best route to source one and source two is through node N_1 and the best route to source three is through node N_2. The formation of the tables shown above assumes a maximum number of hops of 16.

Multicast Forwarding table at node N_1

Source Address	Hop Count
Source One	2
Source Two	5



Multicast Forwarding Table at node N_2

Source Address	Hop Count
Source One	4
Source Three	2

N_2

New table at node N_2

Source Address	Hop Count
Source One	3
Source Two	6
Source Three	2

Table sent from node N_2 to node N_1

Source Address	Hop Count
Source One	19
Source Two	21
Source Three	2

Fig. 2.5: Exchange of Forwarding Tables

2.2.2 SOURCE TREE FORMATION

Multicast data is distributed to all destinations through a Distribution Tree originated at the source. The tree is formed as follows: The source (S) in Figure 2.6 advertises itself to

its immediate neighbors N1 and N2. These two nodes in turn broadcast to their neighbors N3 and N4.

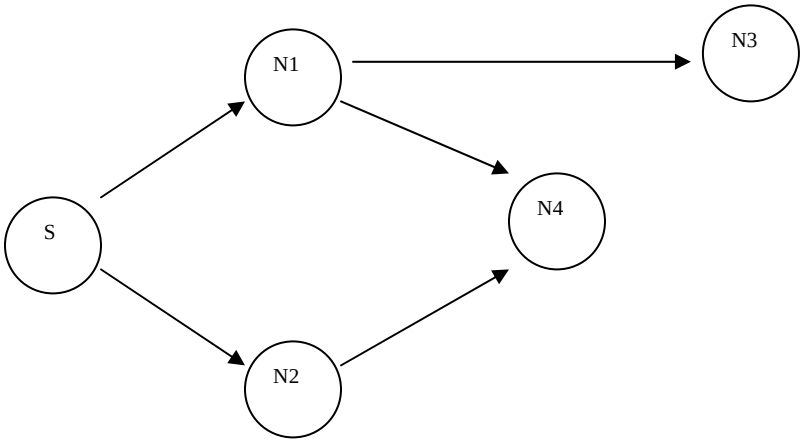


Fig. 2.6: The Advertisement Process Starting at the Source

Assuming N3 is downstream from N1, it will receive data from the source through N1. N4 receives the advertisement from N1 and N2 almost simultaneously. N4 designates its forwarder based on the lowest metric (hop count) and lowest address number. In this case the hop count is the same, so N1 is selected as a forwarder based on its address. From this point on, N4 will process data from N1 and will discard data from N2. In this sense, N1 becomes the designated router or forwarder for N4. The resulting tree is shown in Figure 2.7.

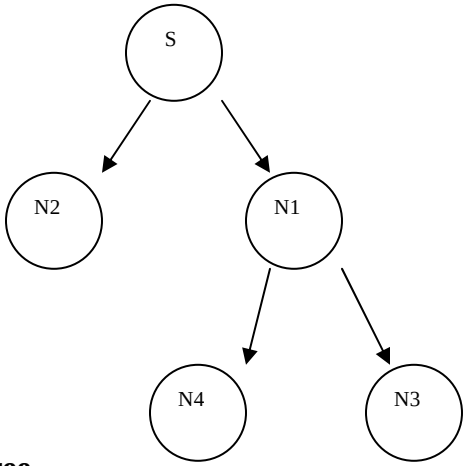


Fig. 2.7: The Formed Tree

Each source can be associated with a distribution tree. In fact, simultaneous sources multicasting to a session can take place in the network and when a receiver wants to switchover across sources the trees involved have to be reconfigured.

2.2.3 MULTICAST TRAFFIC

Trees are used to distribute the multicast traffic to all receivers and to reject traffic coming from a node that is not a designated forwarder for a particular node. In Figure 2.6, it was shown that N4 should reject or discard traffic coming from N2. On a well constructed tree, nodes are not supposed to forward information to a node that is not on its forwarder list, however this situation may arrive before the tree is fully constructed. When a node receives a multicast packet, this packet undergoes a check to ensure that it arrived from the correct forwarder node in the direction of the source before the packet is forwarded.

In Figure 2.8, N4 must reject or discard traffic from N2 and N3, and it must forward traffic received from N1.

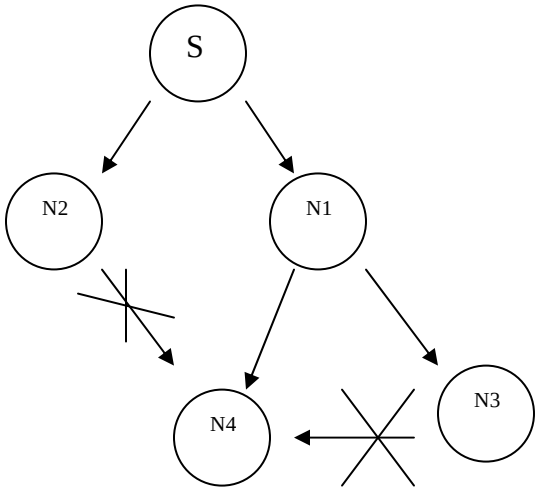


Fig. 2.8: Traffic not coming from forwarders is rejected

This process is similar to Reverse Path Forwarding check where routers check to make sure that the packet is coming from the proper interface.

2.2.4 PRUNING

Initially, traffic is sent down the tree from the source or from the root to every possible receiver. However to conserve bandwidth the traffic flowing through branches with no receivers should be shut off. Removing nodes that are not participating in the multicast session is called pruning.

If a node is not interested in receiving traffic from a multicast session it should investigate if a node down the branch is. If there are no receivers down the branch, the node must stop sending traffic down and inform the upstream node about its intention to leave the multicast group.

2.2.5 NEIGHBOR DISCOVERY

The initial step in implementing a multicast algorithm at any node is to select a neighbor to send a packet to; therefore nodes must have a list of neighbors at all times. The packets exchanged between any two neighbors can contain, besides the identification, the following information: available energy for distance calculations, processing capabilities, and location information if available.

This information allows the creation of the distribution trees and the forwarding of multicast traffic for data delivery. Nodes will maintain a database of neighbors by performing neighbor discovery. To initiate the process, a node broadcasts discovery packets with a hop count of one; every node starts with a null in the neighbors list. When a node hears a discovery packet from another node, it adds the received address to its neighbors list. This node replies with a discovery packet including its own address. After

those two messages have been successfully exchanged, an adjacency between the two nodes has been established and the two nodes are considered neighbors of each other. Figure 2.9 shows the exchange of those messages.

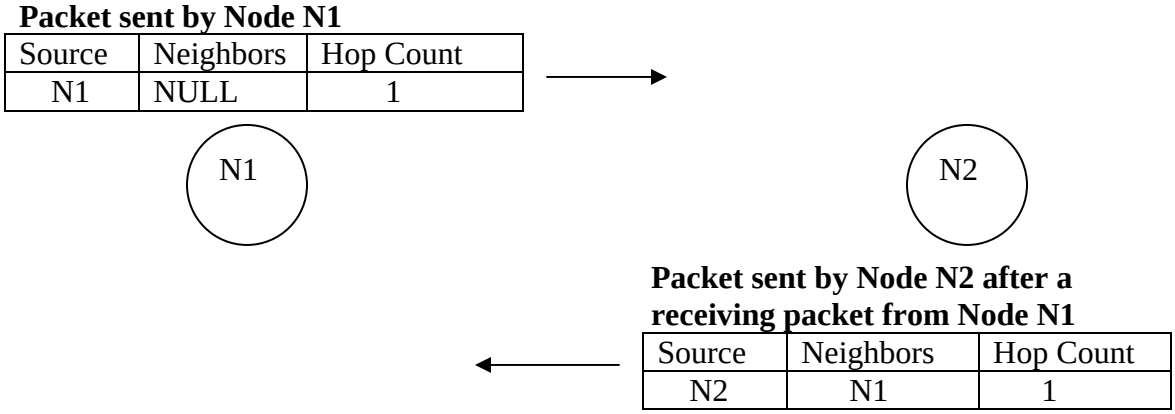


Fig. 2.9: Neighbor Discovery Packets

Discovery packets are sent periodically to check for new or lost neighbors.

2.2.6 PROMISCUOUS LISTENING

In wireless networks, nodes can listen to and process data from packets destined for nearby nodes. This promiscuous listening can be implemented in both the advertisement and registration sessions of a multicasting algorithm. Nodes that receive advertisement packets promiscuously must relay (process and resend) the packets in new advertisement rounds. Nodes that register promiscuously do not act as parents or forwarders; they are just children of the node they register with. Figure 2.10 shows nodes receiving advertisement packets by promiscuous mode, and Figure 2.11 shows nodes that may decide to register promiscuously.

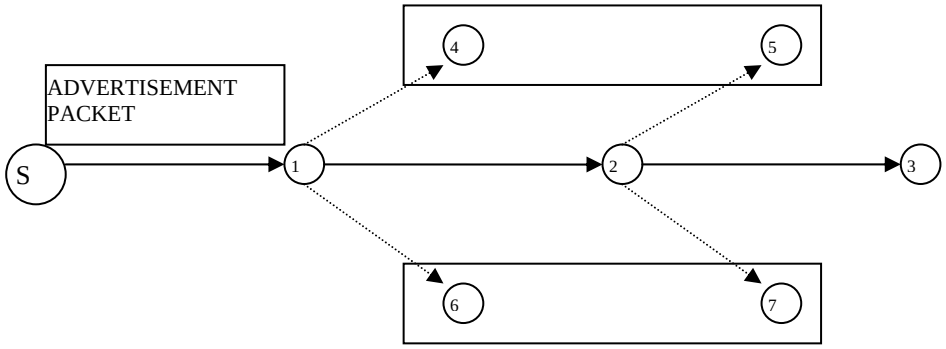


Fig. 2.10: Nodes 4, 5, 6, and 7 shown inside the boxes, are receiving the advertisement packets promiscuously

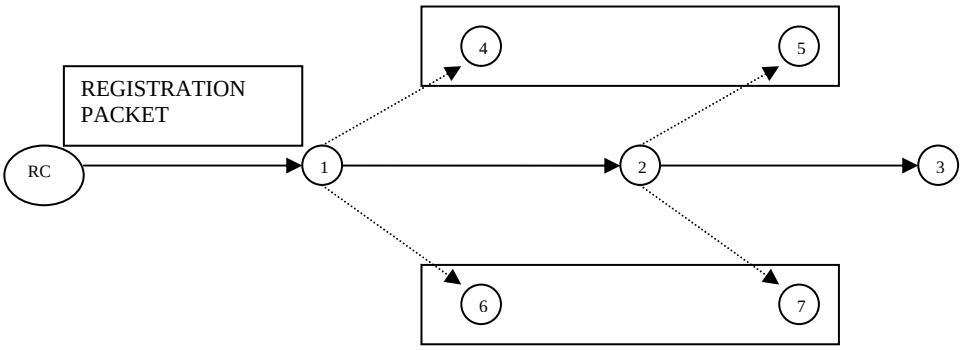


Fig. 2.11: Nodes 4, 5, 6, and 7 shown inside the boxes, may decide to register promiscuously

2.3 MULTICASTING PROTOCOLS FOR AD HOC NETWORKS

When designing protocols for Ad Hoc Networks we have to consider that the physical topology can change continuously, and nodes may join or leave the multicasting group at any time.

Several approaches have been proposed for multicasting in ad Hoc Networks: Multicast Ad Hoc On Demand Distance Vector (MAODV) a tree based protocol, and On Demand Multicast Routing Protocol (ODMRP) a mesh based protocol, are among them. These two are reactive (on demand) protocols. Tree based protocols are efficient for low mobile

networks, but they become fragile in highly mobile networks. Mesh-based protocols are robust for highly mobile networks at the cost of redundancy.

We propose a distributed protocol using discovery packets that can be implemented in dynamic dense mode wireless networks.

2.3.1 MULTICAST AD HOC ON DEMAND DISTANCE VECTOR PROTOCOL

This is a reactive protocol. Multicast routes are discovered on demand by using a broadcast route discovery mechanism. The protocol uses a Route Request/Route Reply/Multicast Activation (RREQ/RREP/MACT) message cycle to add a node to the multicast group. To distribute the multicast data, the protocol forms a shared tree composed of group members and connecting nodes (non-members).

The group membership is dynamic; a node can join or leave the group at any time. Group leaders (first node to request route to the multicast group) are used to maintain and distribute sequence numbers. These numbers prevent loops and help to refresh routes.

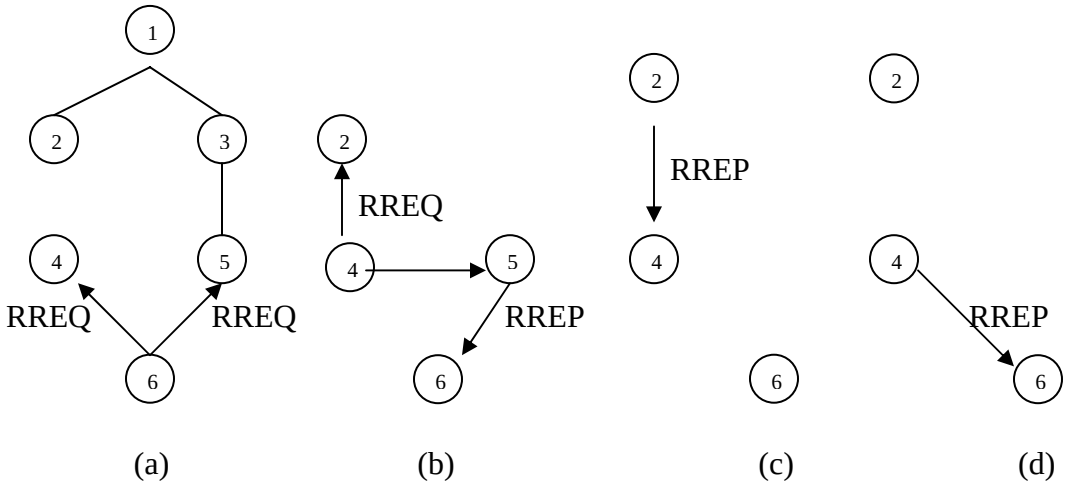


Fig. 2.12: Multicast membership request in MAODV

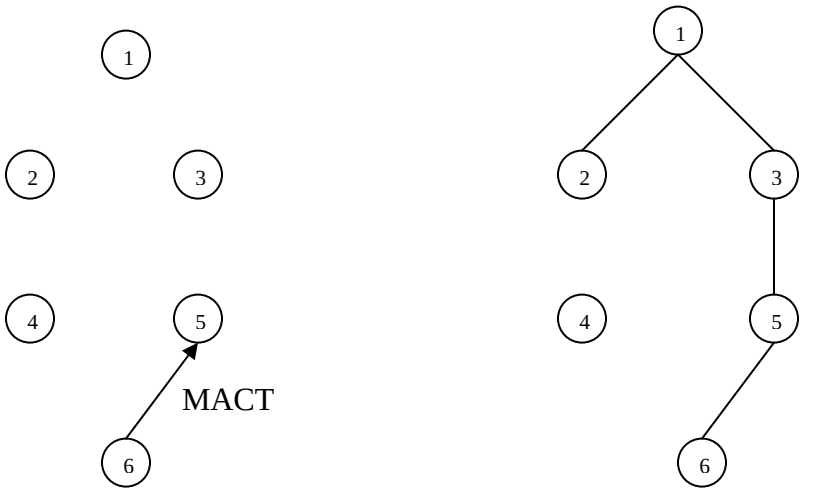
In the sequence shown in Figure 2.12, (a) shows a shared tree with nodes 1, 2, 3, and 5 as members of the tree, nodes 4 and 6 are non-members; node 6 is broadcasting a route request to join the group. In (b), node 5 which is part of the tree is replying to node 6, and node 4 which is not part of the tree is re-broadcasting the request to nodes 2 and 5. In (c), node 2 which is a member, is replying to node 4, and (d) shows node 4 sending the reply to node 6. After waiting a discovery period, node 6 has collected two replies, one from node 5 and one from node 4; node 6 selects the route with the least number of hops to the multicast tree, in this case node 5. To complete the cycle node 6 sends a Multicast Activation (MACT) message to node 5.

Figure 2.13 shows node 6 sending the activation message in (a), and the tree with node 6 as a new member in (b). In the event that a node receives two replies with the same number of hops to the multicast tree, the requester selects the reply with the highest sequence number.

Each node running MAODV maintains two routing tables: a routing table used to record the next hop for routes to other nodes in the network, and a multicast routing table that contains entries for multicast groups of which the node is a member.

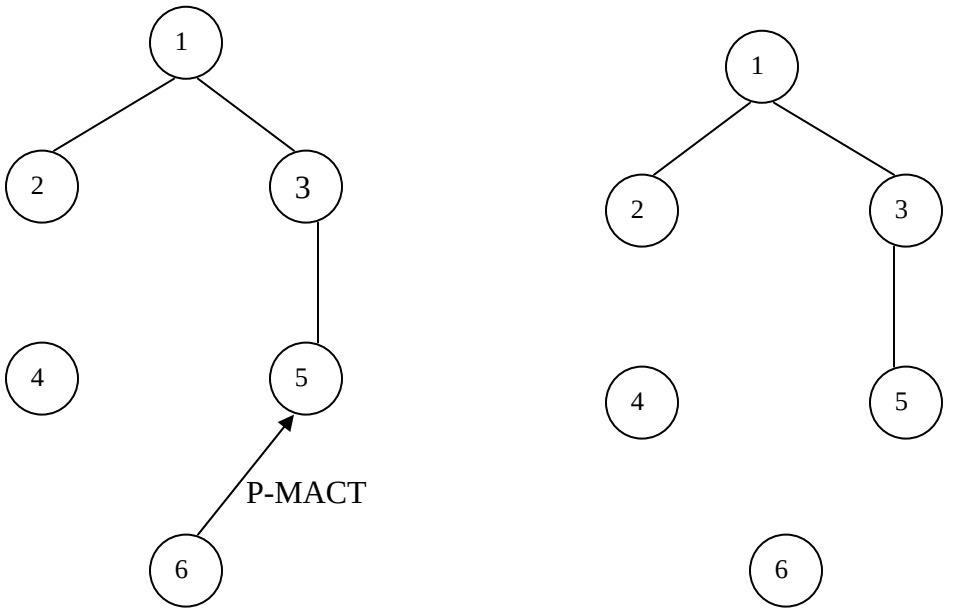
To leave a multicast group, the node that wishes to leave sends a Prune-MACT message to its next hop in the tree. The node that receives the message deletes all multicast group information and deletes the leaving node from its next hop table.

Figure 2.14 shows the pruning process; (a) shows node 6 sending the Prune-MACT message to node 5, and (b) shows the new tree with node 6 as non-member. Only leaf nodes can actually leave the multicast tree.



(a) **(b)**

Fig. 2.13: The Multicast Activation in MAODV



(a) (b)

Fig. 2.14: Pruning process in MAODV

2.4 DISTRIBUTED MULTICASTING WITH DISCOVERY PACKETS

The components of mobile wireless networks are not known in advance, their characteristics can change in an unpredictable way, communication capabilities can break down, and therefore needed services can be become unavailable at any time. Multicasting in Ad Hoc networks is a complex problem [27], because networks can become very large, and mobility and battery usage make the components of the network unpredictable. A system with these types of constraints can be handled by developing a number of modular components specialized at solving particular aspects of it. Discovery packets are tools that can be used to make systems either modular or distributed. Discovery packets act on behalf of their sources; they can meet each other in order to communicate; they can be assigned destinations to involve mobility; they can collect objects inside a module and take ownership, and they can be assigned a time to live or maximum distance to travel. Discovery packets can be transmitted from a source and relayed from place to place; they can be executed within a secure environment at remote locations to obtain the information sought, and they can then be transmitted back to their original source.

Discovery packets can encapsulate data packets, control packets, and information provided by each node they visit. This information can include neighbors, particularly parents and children, nodes that have been visited, node capabilities in terms of energy and processing, location if available, data requests, etc.

By applying modularity, and using discovery packets, the congestion caused by overhead data, and the problem referred to as “acknowledgement implosion,” which causes performance bottlenecking in multicasting, can be distributed among many nodes

in the network. The multicast delivery delay can be reduced by distributing the data to all modules at the same time.

Initially, discovery packets are sent out from the multicast source with a double purpose: first, they must advertise a multicast session, and second, they must explore the network in order to find appropriate locations for nodes to be used as module centers. After modularity has been established, the source distributes the multicast problem among the modules by using their centers as points for registration to the multicasting session, and as hubs for data distribution. The multicast algorithm can be recursively implemented at each module, and the source interacts with its registration centers only.

The general approach of the distributed multicast algorithm is as follows. First, a graph containing the paths from the source to nodes located on the outskirts of the network is constructed, and then the source contacts nodes in the middle of the paths. Then, each path can be recursively broken into two sub paths, each with a multicast informed node.

When a node in the network has data that needs to be multicast, it does the following: 1) it designates itself as a multicast source, and multicasts a discovery-advertisement packet node to its neighbors, 2) it initiates the formation of the distribution tree by sending registration packets to selected nodes, and 3) it multicasts the data to the members in the distribution tree.

3. NETWORK CONTOUR DISCOVERY ALGORITHM

3.1 INTRODUCTION

We consider the problem of finding the shortest distance, in terms of the number of hops, from a source node to unknown nodes located on the outskirts of a network. At least one distance is found through each neighbor of the source node to determine the contour of the network. The algorithm is based on the deployment and return of traveling discovery packets. The accuracy of the found contour relies on the availability of a mechanism that propagates discovery packets strictly in an outward direction relative to the source node.

The contour of the network outlines its shape. The knowledge of the shape allows a source node to estimate the density of nodes throughout different regions of the network. By implementing the outward propagation mechanism presented in this work, a source node can learn the distribution of nodes in the network, allowing it to determine that in the direction of each of its neighbor nodes, the farthest node is located N hops away and therefore there are N-1 nodes in between. Source nodes do not learn the actual location of other nodes; what they learn is how far a node is located in terms of number of hops and how to reach a node that has been visited by a returning backward packet. The network distribution unfolded by this mechanism can be effectively used to create dynamic routing tables at a source node, or to create modules for a distributed multicasting algorithm.

The mechanism presented here controls the forward and backward propagation of packets of mobile packet-type deployed from a source node. It should not be expected

that discovery packets deployed from a source node without a known destination propagate away in a straight line. In fact, the probability that they take curved trajectories is higher than the probability that they take a straight trajectory. To increase the probability of straight paths, discovery packets can embed in them the sender's neighbors list. This information is passed from node to node so that new packets will not visit those neighbors.

The process for outward propagation is illustrated in Figure 3.1. Node A has selected node B among its neighbors as the new destination. Node A passes a list of its neighbors to node B. Even though nodes C and D are neighbors of B, node B will not use them to propagate the packet. It is expected that by providing this neighbor information, the trajectory of the forward packets follows an outward direction. Even though the propagation of the sender's neighbors reduces the probability of inward paths it still does not guarantee the formation of outward trajectories. In section 3.4.3 we add the following constrain into the algorithm: if a neighbor of a receiver can be detected as a neighbor of the sender's neighbors, this neighbor can not be selected as a new destination. It will be shown that this process effectively improves the formation of outward trajectories.

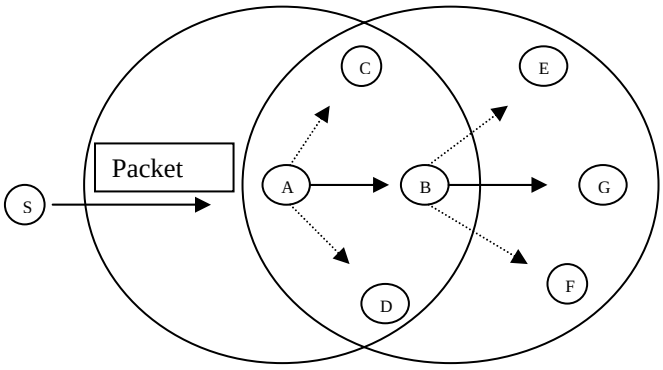


Fig 3.1: Outward Propagation

The mechanism to build paths that grow strictly outward from the source requires the implementation of two stages prior to the contour discovery stage: 1) neighbor discovery and 2) neighbor exchange. The first stage allows every node in the network to discover a set of nodes that are in their transmission range, and the second one allows every node to exchange their neighbor databases. A node in the contour discovery stage uses the information from those two stages to select the most appropriate node as the new receiver for the packet. The selected node can neither be a neighbor of the sender nor a neighbor of a sender's neighbor. Following these restrictions in the selection process for a new destination will send discovery packets from node to node in an outward direction from the source and inward paths are not expected to be formed. Figure 3.2 shows the stages implemented in the outward propagation mechanism.

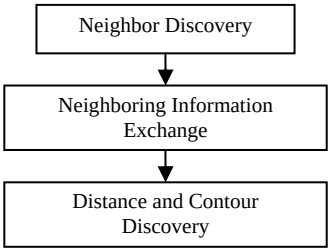


Fig 3.2: Three stages of the outward propagation mechanism

3.2 THE USE OF DISCOVERY PACKETS

Traveling discovery packets hop from node to node to collect information from them and then they gift their collected data to the source node from where they were deployed.

These packets act on behalf of their sources; they can be assigned destinations to involve mobility; they can collect information from inside a node, and they can be

assigned a time to live or a maximum distance to travel. These packets can be transmitted from a source and relayed from node to node; they can be transmitted to remote locations to obtain the information sought, and they can then be transmitted back to their original source.

Traveling packets can encapsulate data packets, control packets, and any knowledge provided by each node they visit. This information can include neighbors, parents and children in a tree, which nodes were visited, node capabilities in terms of energy, processing and memory capacity, location if available, and data requests.

3.3 RELATED WORK

The work presented in this chapter touches on the following related topics: Topology Discovery, Routing, Loops, Mobile Agents, Shortest Path, and Ant Colony Algorithms.

There is a number of existing protocols related to network discovery, including topology discovery and path discovery. Mechanisms to avoid infinite loops and algorithms to find the shortest path between a source and a terminating point are always used in any discovery protocol. Mobile agents [37] are currently being used to explore a network by collecting information from visited nodes and distribute it among other nodes in the network.

An algorithm that relies on standard SNMP Management Information Basis for topology discovery in Heterogeneous IP Networks is presented in [3]. Mtrace [4] is a multicast version of the traceroute utility; it is used to discover the multicast path between a given receiver and a source in a multicast group.

A scalable mechanism to discover multicast tree topologies in the internet is presented in [5]. The basic idea of using mobile agents for topology discovery has been explored in MIT Media Lab [6]. Mobile agents [1] that hop around the network are a novel solution to the problem of topology discovery. The agents hop from node to node, collect information from these nodes and distribute it among other nodes in the network. In recent work R. RoyChoudhury et al. [7] proposed a distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents. An Active Approach to Multicasting in Mobile Networks is presented in [30].

Mobile packet systems have been simulated in close resemblance to an ant colony [8], [9]. A routing protocol for ad hoc networks using ant algorithms is presented in [2].

Some of the mechanisms used by network discovery protocols include tools to deal with loops. Split Horizon [10] is a tool used to eliminate loops and speed up convergence in routing protocols. The rule is that sending information about a route back in the direction from which the original update came is never useful. Finally, the Shortest Path Algorithm [11] is an algorithm to determine the shortest distance from a source node to all other nodes in the network

Our primary concern in this work is to find the shortest distances from a source node to various unknown nodes located by the outskirts of the network. At least one distance is found through each neighbor of the source. The mechanism employed to propagate the discovery packets guarantees that the distance found between the source and any of the unknown nodes is the shortest. Moreover, we use forward and backward packets of mobile packet-type, these packets carry information about neighbors and nodes visited allowing a source to create routing tables to visited nodes. Furthermore, data collected at

the source provides information about the contour of the network, as well as necessary information to create and allocate modules for a distributed multicast algorithm.

3.4 OUTWARD PROPAGATION AND CONTOUR DISCOVERY

The goal in this session is to minimize the discovery time, which is the time required to propagate outward discovery (forward and backward) packets through a high percentage of nodes in the network. In order to achieve this goal, discovery packets will be deployed from a source at a periodic rate. The maximum number of hops such packets have to travel is specified in the packet itself. Initially, this number has a unity value, and then it is adjusted following one the following three expansions:

a) Unity Expansion: This expansion which is used by many discovery protocols in ad hoc networks increases the radius of expansion rings by one hop. Even though it provides updated information at short time intervals, simulations show very high link utilization, and high redundancy factors. Redundancy is defined as the ratio of the number of nodes visited once or more than once versus the number of nodes visited only once. This expansion technique should be used only in very small networks.

The next return is found by executing the following operation:

New Next Return = (Current Next Return) + 1 **(3.1)**

b) Binary Expansion: This expansion technique starts with a unity value as the number of hops for the next return. Subsequent values in the expansion are found by increasing the number of hops by a factor which forces the packet to return when the number of hops reaches a value in the binary distribution.

This expansion technique increases the number of hops by a factor of two every time this number reaches a value which is equal to a power of two. Link utilization and redundancy factors for this expansion technique are lower than those in the unity expansion. Even though the link utilization and redundancy factors improve as the network size increases, this technique should not be used for very large networks since the spacing between any two powers of two becomes larger. A large value for a next return in a discovery packet might give place to inward paths defeating the purpose of this expansion technique.

The next return in this binary technique is found by executing the following operation:

New Next Return =2*(Current Next Return) **(3.2)**

c) Square Expansion: This expansion technique starts with a unity value as the number of hops for the next return. Subsequent values in the expansion are found by increasing the number of hops by a factor which forces the packet to return when the number of hops reaches a value in the quadratic distribution.

The next return is found by executing the following operation:

$$\textbf{New Next Return} = (\textbf{Current Next Return})^{1/2} + 1)^2 \quad (3.3)$$

Link utilization and redundancy factors for this expansion technique are lower than those in the unity, and comparable to those found in the binary expansion for small networks. However, for large networks the quadratic expansion offers less spacing between any two values in the distribution when we compare it to the binary distribution. Those lower spacing values reduce the possibility of creating inward paths, since packets

will be forced to return before loops can be created. Keeping in mind that the essence of the algorithm presented here is based on outward propagation with direct trajectories, we have selected the quadratic expansion for most of our theoretical work as well as for our simulations. Figure 3.3, shows forward packets propagating away from the source and backward packets returning when there are no more eligible neighbors to receive the packet or when the number of hops at a node reaches a number in the quadratic distribution.

When a node using any distribution, receives a forward packet from a source, it selects one of its neighbors as the destination to propagate it. As the packet is propagated, all of its neighbors will listen to the information it contains, but only the destined node will process it and retransmit it. A neighbor node that is visited by a discovery packet increases the number of hops by one before the packet is relayed. When the number of hops reaches the maximum, or is equal to the next return number as specified in the packet, the node being visited generates a new packet called a backward packet, and sends it to the source following the reverse path of the forward packet. The backward packet travels back to the source node and provides it with the following information: visited nodes, the path and number of hops to reach those nodes, and the number of hops the packet has traveled.

The propagation of a forward discovery packet can be stopped at any node for one of the following reasons: a node does not have any neighbors, or all neighbors of the node have already processed a forward packet for the same discovery session. Any one of these events will trigger the generation of a backward packet to be sent back to the source.

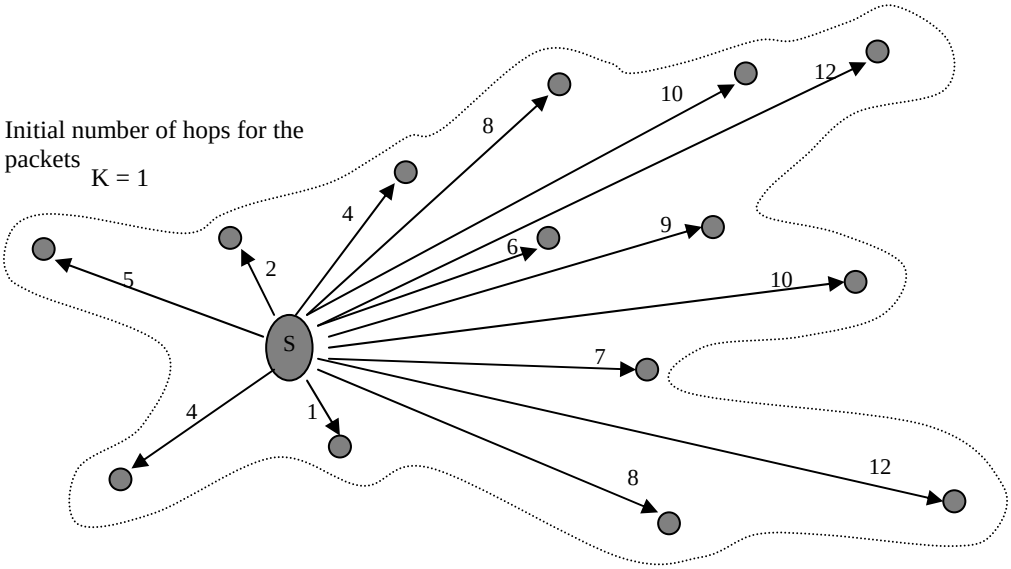


Fig 3.3: Quadratic Distribution Expansion

The number of forward packets emitted from the source depends initially on the number of neighbors, and later on the information being received from backward packets. If the number of neighbors is small, more than one packet will be sent through each neighbor. In the extreme case that the source has only one neighbor, multiple packets with instructions to follow different paths will be sent through that neighbor.

Each discovery packet will have the following information: source node, time stamp, sender's neighbors, maximum hop count, and a log of nodes visited. As the process progresses, the source keeps sending discovery packets in a sequential order throughout all of its neighbors and waits for backward packets.

When the source receives a backward packet, it first checks the number of hops that the packet has traveled. If the number of hops traveled by the packet is less than the number specified when the packet was released, the source learns that there are no more nodes to be visited in this direction and determines the length of the network in that

specific direction. If the number of hops traveled by a forward packet is equal to the maximum number of hops indicated in the packet, the source learns about the possibility of more nodes to be visited in that direction and releases another packet with a new next return value to follow the same path. Figure 3.4 shows a flowchart for the propagation of forward and backward packets.

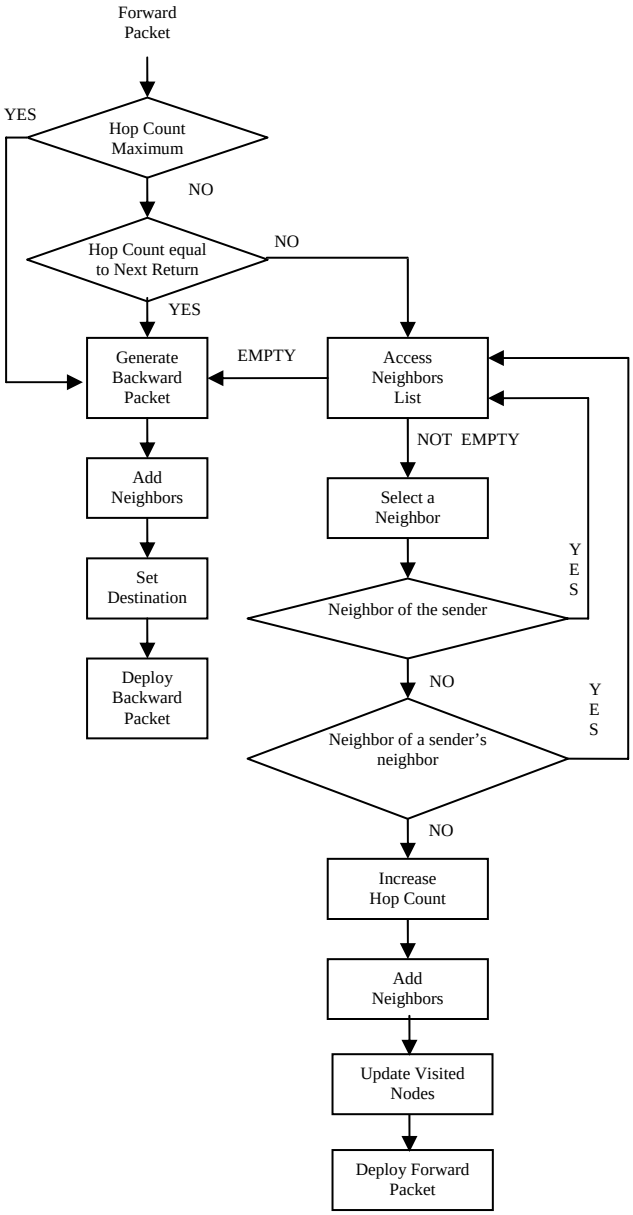


Fig 3.4: Flowchart for Forward and Backward packets

To increase coverage, the source can release a second round of packets throughout its neighbors to be propagated through different paths from those of the first round. Even though all of the source's neighbors have processed forward packets during a first round of released packets, they will be forced to re-process them during a second round and send them to their unvisited neighbor nodes.

Particular situations may require the release of a third round of discovery packets. This may occur when the number of neighbors is small or the network coverage is not satisfactory. The source's neighbors and neighbors of the source's neighbors are then instructed to re-process the packet and send it to their unvisited neighbors.

In this way, those packets in the third round will be launched and forced to go away from the source searching for unvisited nodes. In the simulation section 3.5 we show the coverage for two and three rounds. Figure 3.5 illustrates a process where nodes 1 and 2 are shown reprocessing forward packets for second and third round of packets respectively.

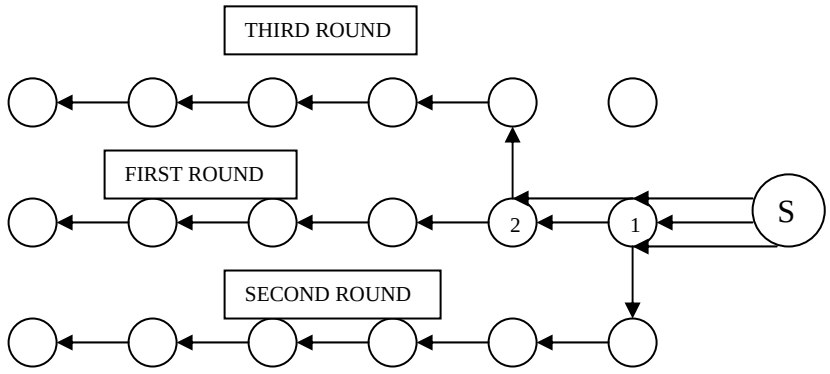


Fig 3.5: Node 1 and Node 2 reprocessing forward packets. Even though these nodes have processed the discovery packet during round one, they reprocess the packet for round two and three respectively. Forcing these nodes to reprocess those packets increases the coverage of the network.

The number of rounds used by the algorithm in the discovery process is determined by the following factors: a) the number of neighbors at the source, and b) the number of hops the discovery packets are traveling (this information is provided by backward packets).

3.4.1 NEIGHBOR DISCOVERY

Nodes will maintain a database of neighbors by performing neighbor discovery. To initiate the process, a node broadcasts discovery packets with a hop count of one; every node starts with a null in the neighbors list. When a node hears a discovery packet from another node, it adds the received address to its neighbors list. This node replies with a discovery packet including its own address. After those two messages have been successfully exchanged, an adjacency between the two nodes is established and the two nodes are considered neighbors of each other. Based upon this procedure, Program 3.1 gives the pseudo-code for this process. The complete code is appended in appendix A.

```
If (packet is coming from own source)
        {
         Set node's id into the packet and send it to all of its neighbors
         }
  else
     if (packet is arriving from another source)
         {
          if (packet is a neighbor discovery packet)
          {set node's id, change packet type, and send it back}
              else
                   if (packet is a neighbor reply packet)
                      { read incoming port
                        get neighbor's id
                        save neighbor and port of arrival }
          }
         }
```

Program 3.1: Pseudo-Code for Neighbor Discovery

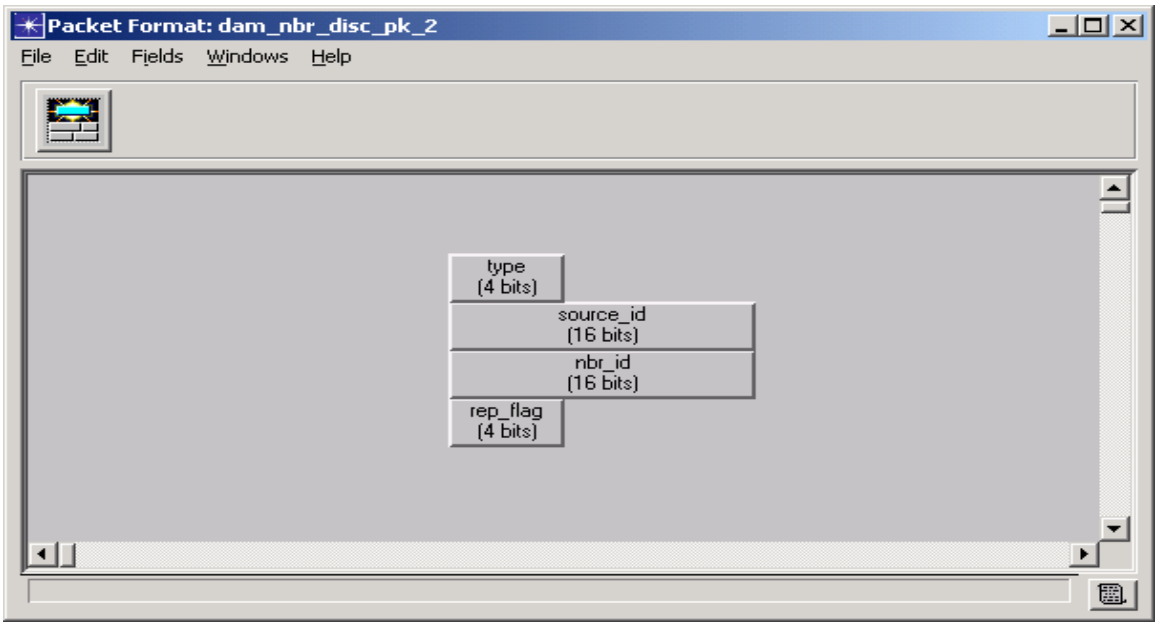


Fig. 3.6: Neighbor Discovery Packet

Figure 3.6 shows a neighbor discovery packet. These packets have the following fields:

Type: This field indicates the packet type.

Source_id: This field indicates the identification of a node generating this packet.

Nbr_id: This field indicates the identification of a node replying to a neighbor request.

Rep_flag: A value of zero in this field indicates that a node is requesting neighbor information, and a value of one indicate that a node is replying to a request.

3.4.2 NEIGHBOR EXCHANGE

The basic assumption in the analysis of the algorithm presented herein is that the growth of paths or trajectories is considered strictly outward. In practice, exceptions to this outward growth are possible. In order to minimize the possibility of inward paths, the mechanism requires the implementation of a stage where every node in the network receives a request to exchange its neighboring information within its set of neighbors. This information allows a node to select the appropriate destination for a packet in a way

that keeps this packet moving in an outward direction from the source. Program 3.2 gives the pseudo-code for the neighbor exchange stage. The complete code is appended in appendix B.

```
if (packet is coming from own source)
  {
   Set node's id
   Set own neighbors into the packet
   Send the packet to all its neighbors
   }
else if (neighbor information is coming from another  node)
            {
             Read port of arrival
             Read sender's id
            Read sender's neighbors
            Update database
            }
```

Program 3.2: Pseudo-Code for Neighbor Exchange Information

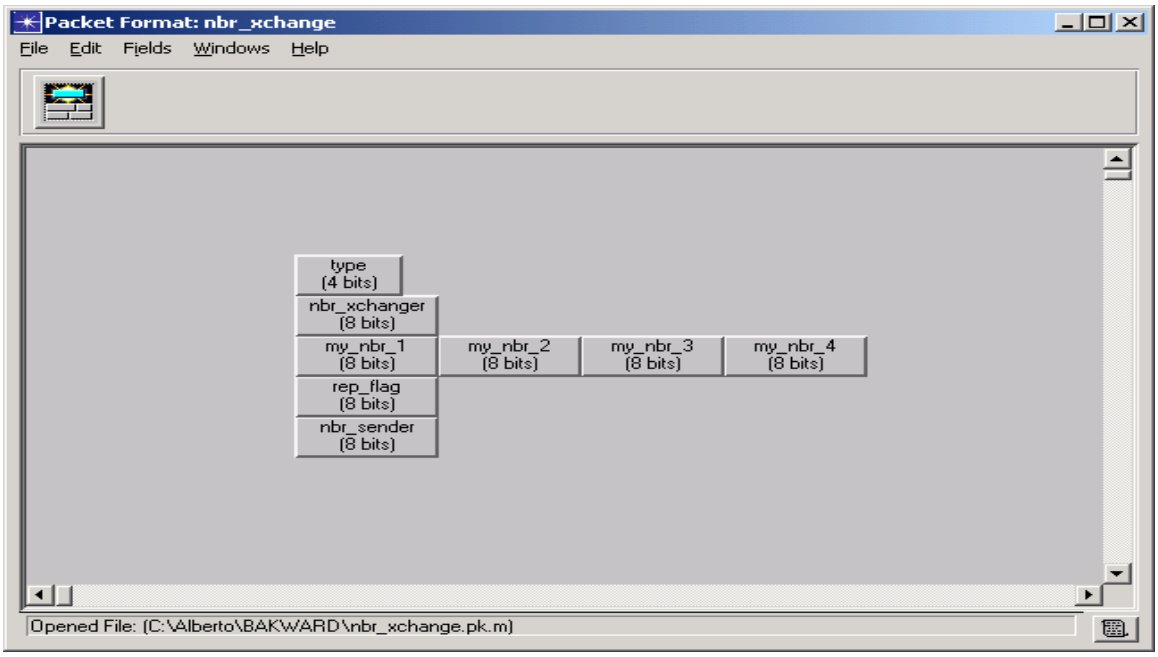


Fig. 3.7: Neighbor Exchange Packet

Figure 3.7 shows a neighbor exchange packet. These packets are used to exchange neighborhood information, and they have the following fields:

Type: This field identifies the packet type.

My_nbr_1 through My_nbr_3 fields contains the neighbors of the sender node.

Rep_flag: A zero in this field indicates a request for neighbor exchange, and a one indicates a reply.

Nbr_sender: This field carries the identification of the node sending its neighbors information.

3.4.3 PACKET OUTWARD PROPAGATION

The accuracy and reliability in the output of this algorithm are based on the growth of the path or trajectory for any discovery packet which must be strictly in an outward direction from the source node. Program 3.3 gives the steps of the algorithm including the basic variables in use.

Algorithm: Outward Propagation

Given a collection of Nodes 1, 2, 3, …,M each with a collection of neighbor nodes 1, 2, 3, …,N; this algorithm revolves around four variables and two basic arrays in addition to other arrays per each neighbor up to a maximum of M. These variables and arrays are identified as follows:

number of nodes (M): integer
number of neighbors (N): integer
sender: integer
my_node_id: integer

Array of Neighbors (my_nbr[i]): integer; where i varies from 0 to N
Array of Sender's Neighbors (sender_nbr[i]): integer; where i varies from 0 to N
Array of Neighbors' Neighbors:
    my_nbr_nbr_1[i]: integer; where i varies from 0 to N
    my_nbr_nbr_2[i]: integer; where i varies from 0 to N

```
        my_nbr_nbr_N[i]: Integer;  where i varies from 0 to N

REPEAT
   FOR  i ← 1  to  N
        IF  my_nbr_nbr(i)= my_node_Id
            my_nbr_nbr(i)=-1
   ENDFOR
   FOR  i  ← 1  to  N
       FOR j  ← 1  to  N
            IF  my_nbr_nbr(i)=sender_nbr(j)
                 Or
                my_nbr_nbr(i)=sender
                BREAK
                     ELSE
                           send packet to i
            ENDIF
       ENDFOR
  ENDFOR

UNTIL ALL my_nbr_nbr_N ARRAYS ARE INCLUDED
```

Program 3.3: The algorithm for Outward Propagation

It should be noted that a value of -1 is used as a sentinel to eliminate the id of the node searching for a neighbor from the neighbors' neighbors vectors.

To illustrate the outward propagation mechanism, let's refer to the situation in Figure 3.8, where a source node **s** and a termination node **t** are shown along with 14 other nodes. Assuming that the source **s** has selected node D from among its neighbors as the receiver for the packet, node D in turn has to select one of its neighbors to send the packet to.

The packet cannot be sent back to the source, and it can be sent neither to A nor to B because these are neighbors of the source.

F	J	M	**t**
C	G	K	N
A	D	H	L
s	B	E	I

Neighbors of the source: [A D B]
Neighbors of the sender D: [A C G K H E B s]
Neighbors of the sender's neighbors:
Neighbors of Node A: [C G D B s]
Neighbors of Node C: [F J G D A]
Neighbors of Node G: [C F J M K H D A]
Neighbors of Node K: [G J M t N L H D]
Neighbors of Node H: [D G K N L I E B]
Neighbors of Node E: [B D H L I]
Neighbors of Node B: [s A D H E]

Fig. 3.8: A packet released from node s has to end up at node t passing only through nodes D and K.

Furthermore, node D should not select nodes C, or G because they are neighbors of A, and node A is a neighbor of the sender s. Nodes H, and E also cannot be selected because they are neighbors of B, which is also a neighbor of the sender s. The only node eligible to receive the packet is node K; so node D proceeds to release the packet destined to node K.

Now node K has to select one node from among its neighbors as the receiver for the packet. Nodes G and H are not eligible to receive the packet because they are neighbors of the sender D. Nodes J and M cannot be selected because they are neighbors of G, and G is a neighbor of the sender D. Nodes N and L are not eligible to receive the packet since they are neighbors of node H and this node is a neighbor of the sender D. The only node eligible to receive the packet from node K is node **t,** so node K proceeds to release

the packet destined to node t. As a result, the packet has propagated in an outward direction from node **s** to node **t,** passing only through nodes D and K.

A similar analysis can be used to show that a packet sent from node **s** to node A will end up at node F through the s→A→C→F outward path, and a packet sent from **s** to node B will end up at node I through the s→B→E→I outward path. This mechanism does not allow the creation of inward paths.

3.5 SIMULATION OBSERVATIONS AND RESULTS

We have selected OPNET 11.0 as the platform to implement our simulations and to study the performance and accuracy of the outward propagation algorithm.

String and Star topologies have been used to test the deployment of forward packets throughout all branches of the network, and the propagation of forward and backward packets. When using these topologies, the algorithm performs as expected; forward packets effectively propagate following an outward trajectory from the source, and when the next return according to the expansion being used or maximum number of hops is reached, the current node generates a backward packet and sends it back to the source. The backward packet propagates along the reverse path of the forward packet with the original source as its destination. The source detects the arrival of backward packets and logs visited nodes in the direction of that neighbor to determine the distance to the last visited node.

The grid topology has been used to test the propagation of forward packets with and without sender's neighbor information, and the network coverage. Using this topology, the algorithm performs as expected. Loops are minimized when the neighbor information

is included in the forward packets. We have also tested the utilization of the links and the delay time for all three expansion types: Unity, Quadratic, and Binary. The performance in terms of link utilization and time delay is very poor for the unity expansion, while the quadratic and the binary expansion perform very close to each other at least for a small number of hops. When the number of hops is increased, the performance of the binary expansion improves compared to the quadratic. However we prefer the quadratic expansion for contour discovery because the decrease in its performance is compensated by its ability to decrease the possibility for the unwanted inward propagation paths.

In order to test the coverage of a network we have deployed discovery packets on different topologies. One hundred percent of the network is covered when one round of packets is deployed in the string topology shown on Figure 3.9. The coverage for this topology is shown in Figure 3.10.

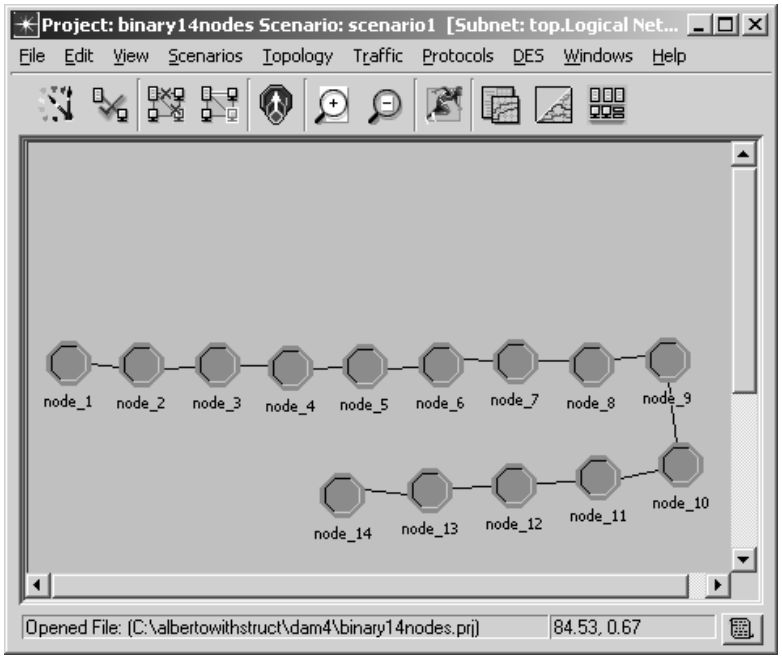


Fig. 3.9: String Topology with 14 Nodes

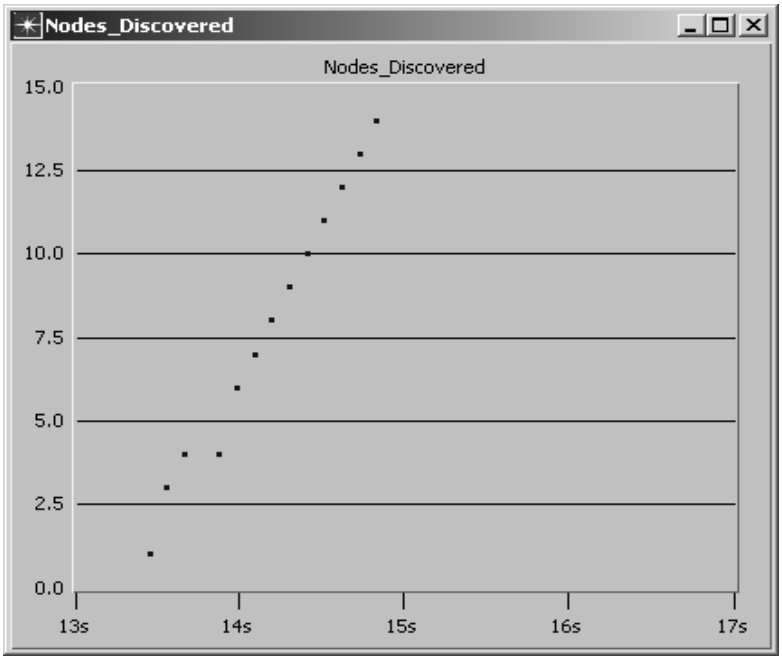


Fig. 3.10: Coverage for a 14 Nodes String Topology with Source at Node7

Shown in Figure 3.11 is a non-symmetric network with 25 nodes. This network has been used to test the coverage when discovery packets are deployed from nodes at different locations.

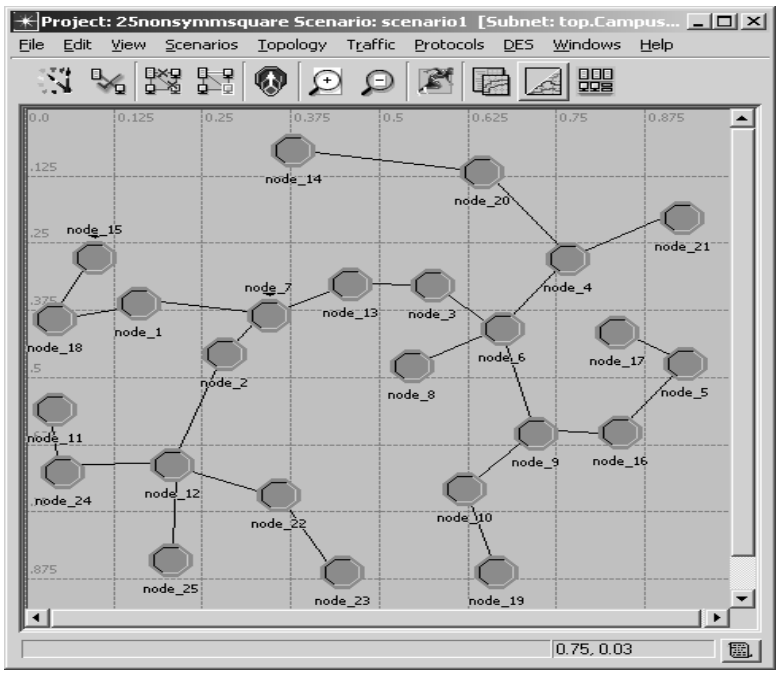


Fig. 3.11: 25-Nodes Non-Symmetric Topology

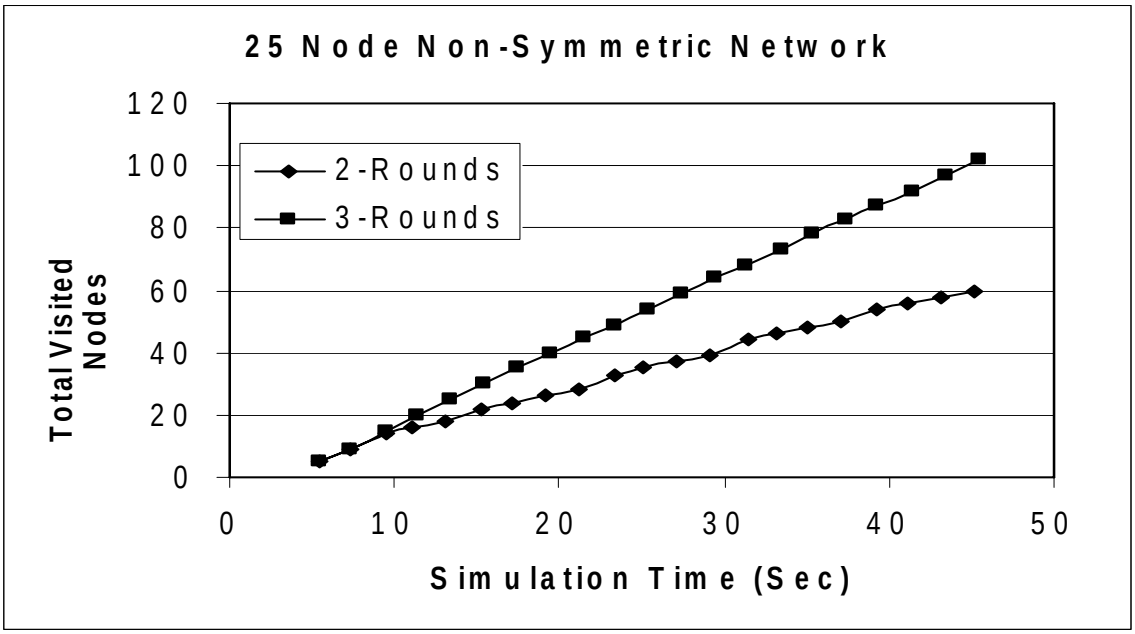


Fig. 3.12(a): Total Visited Nodes with redundancy by 2 and 3 rounds as a function of time

Figure 3.12 shows visited nodes and the redundancy factor for two and three rounds of packets as a function of time in the network with 25 nodes shown in Figure 3.11. Redundancy is defined as the ratio of the number of nodes visited once or more than once versus the number of nodes visited only once. Two and three rounds mean sending a forward discovery packet through the same neighbor node two and three times respectively. These packets were left to propagate without any expansion control mechanism until they found no more neighbors, or the maximum number of hops specified in the packet was reached. Figure 3.12(a) shows the total number of visits including redundancy by those forward packets, Figure 3.12(b) shows the effective net number with no redundancy of visited nodes, and Figure 3.12(c) shows the redundancy factor.

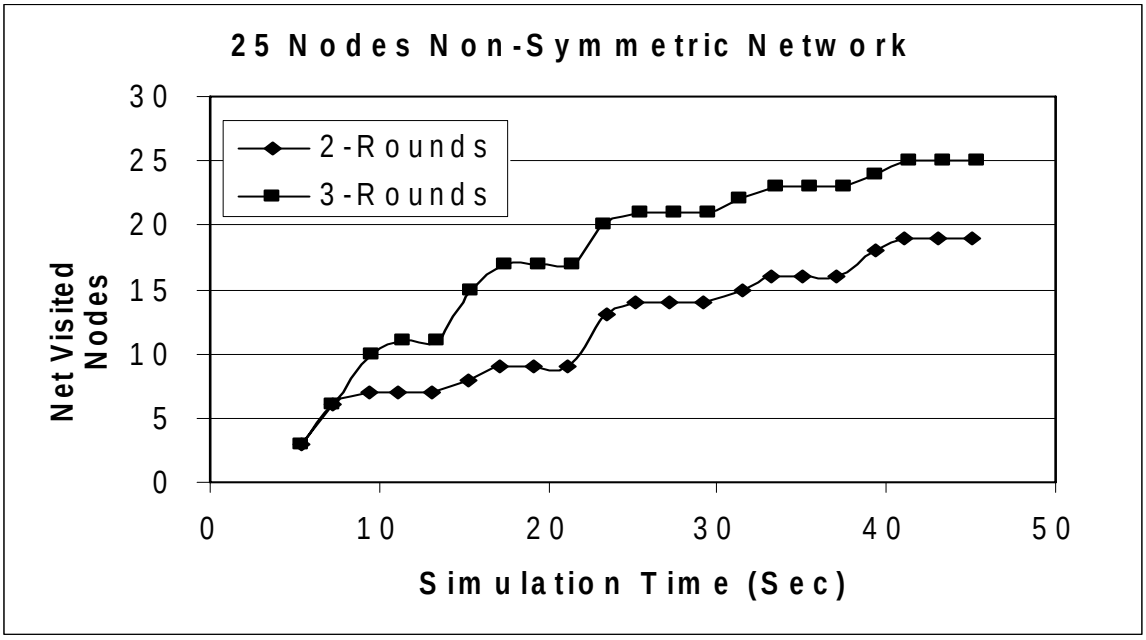


Fig. 3.12(b): Effective Visited Nodes by 2 and 3 rounds as a function of time

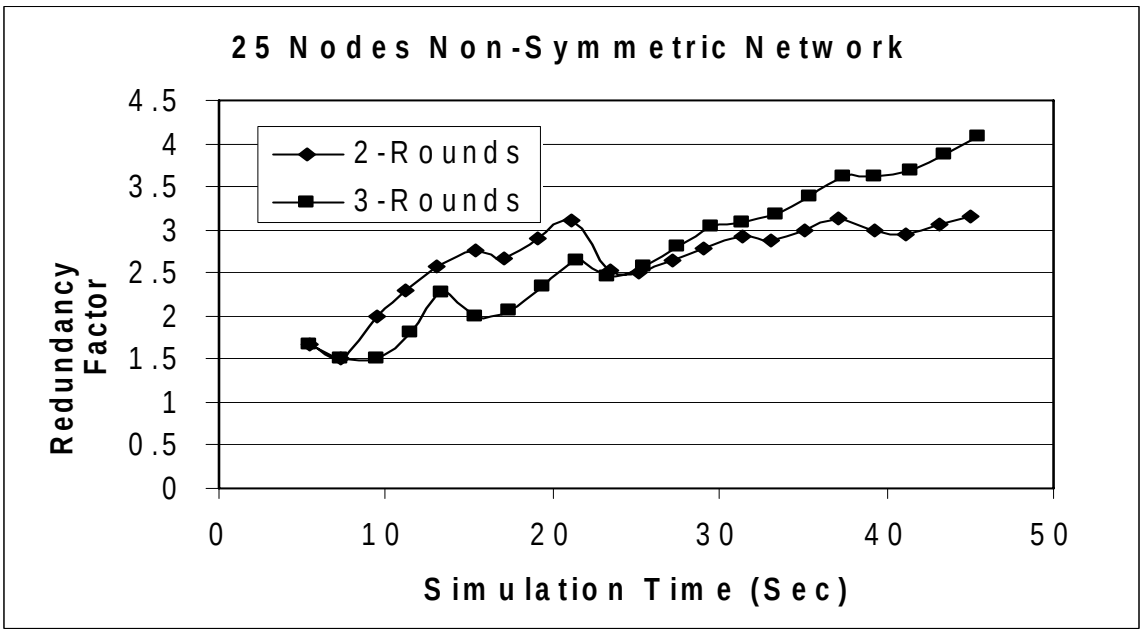


Fig. 3.12(c): Redundancy Factor from 2 and 3 rounds as a function of time

Figure 3.13 shows the total number of visits and the redundancy factor of the three expansion distributions (Unity, Binary, and Square) as a function of time for the network

with 25 nodes shown in Figure 3.11. Figure 3.13(a) shows the total number of visited nodes including redundancy, and Figure 3.13(b) shows the redundancy factor for the three expansion techniques simulated.

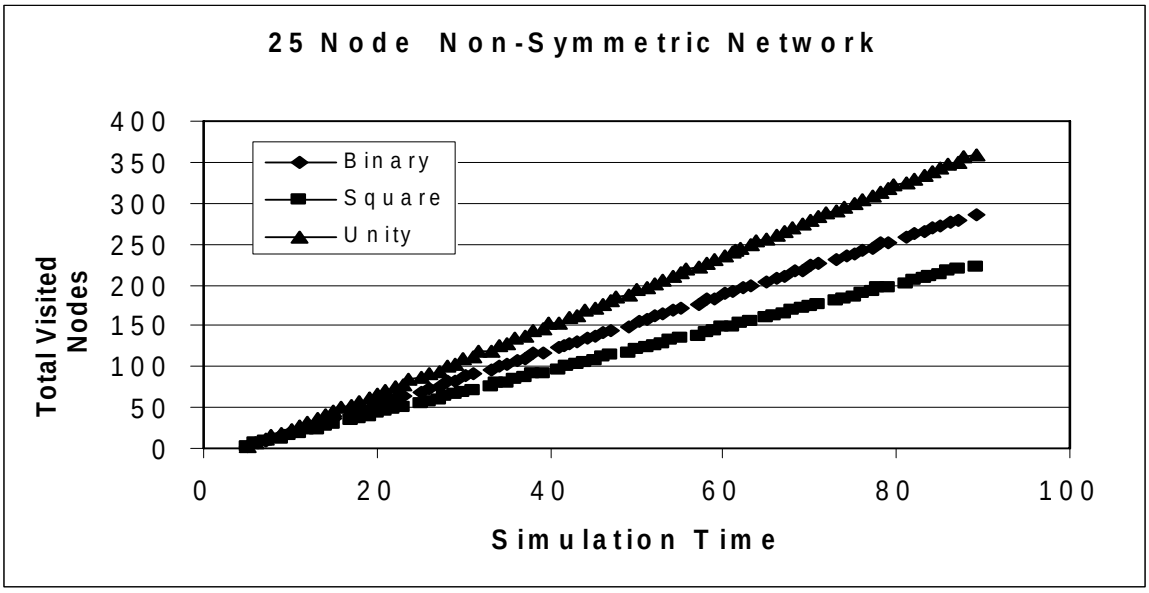


Fig. 3.13(a): Total number of visited nodes for Binary, Square, and Unity expansions as a function of time

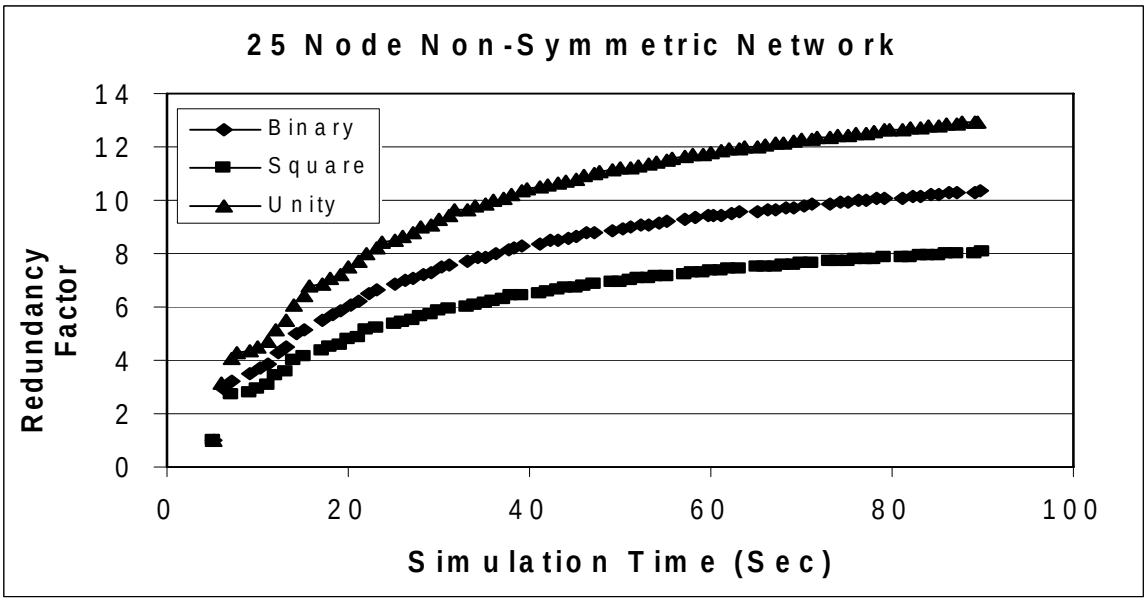


Fig. 3.13(b): Redundancy Factor for Unity, Binary, and Square Expansions as a function of time

The same network with 25 nodes network has been used to test the coverage as a function of time when discovery packets are deployed from nodes at different locations. Figure 3.12 and Figure 3.13 show the coverage when packets are deployed from the middle of the network at node 3. Figure 3.14 shows the coverage when the discovery packets are deployed from one side of the network at node 12. In order to obtain full coverage from this location it would be necessary to deploy three rounds of discovery packets. A similar situation to that presented in Figure 3.14 is shown in Figure 3.15, when discovery packets are deployed from the other side of the network at node 9.

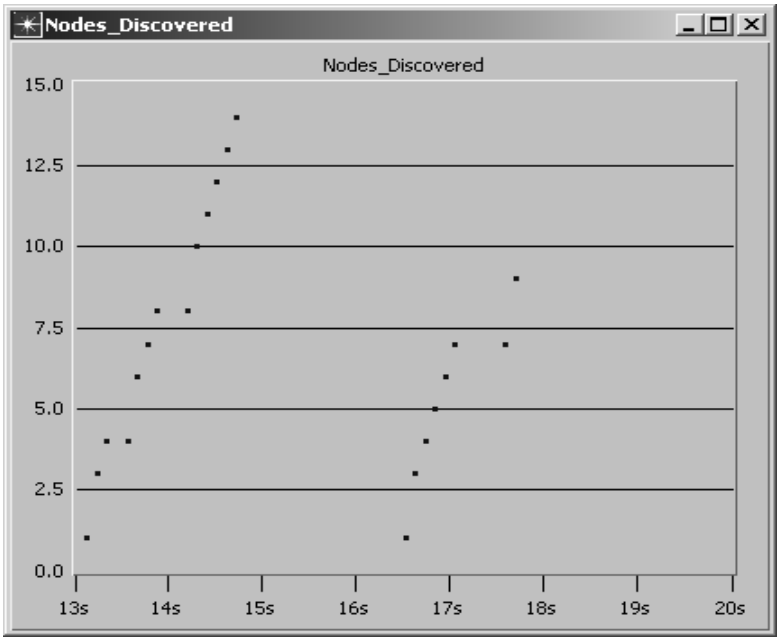


Fig. 3.14: Coverage for the 25 Nodes Non Symmetric with the Source at Node12

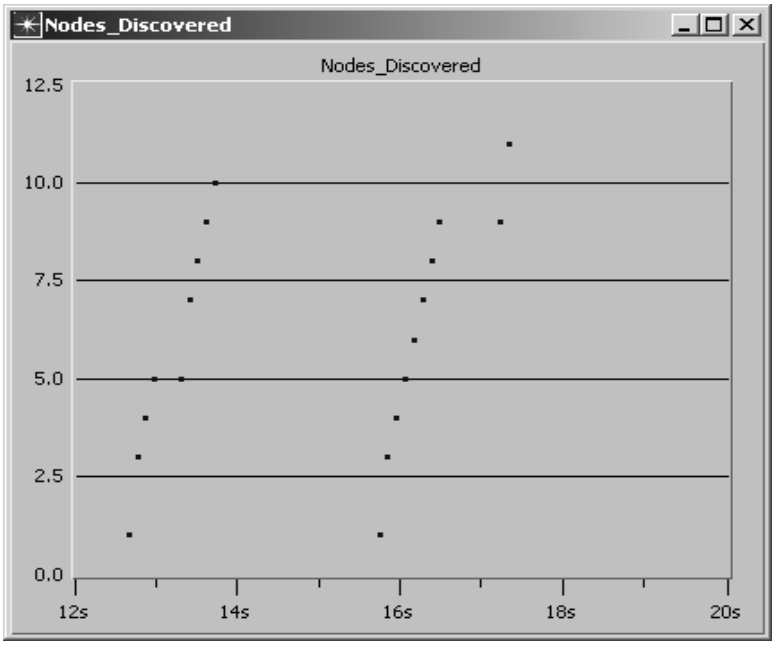


Fig. 3.15: Coverage for the 25 Nodes Non Symmetric with Source at Node 9

A network configured in a grid topology is shown in Figure 3.16. It has 30 nodes with up to four links at each node. The coverage when discovery packets are deployed from the middle of the network at node 14 is shown in Figure 3.17, and the coverage when packets are deployed from one corner at node 1 is shown in Figure 3.18.

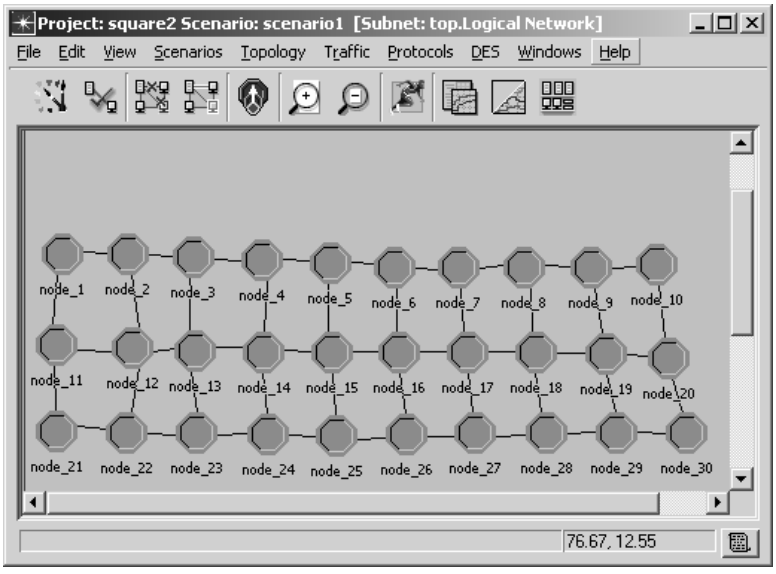


Fig. 3.16: 30-Nodes Grid Topology Network

It should be noted that the discovery time increases for each round of packets when the source is moved away from the center of the network as it did from node 1.

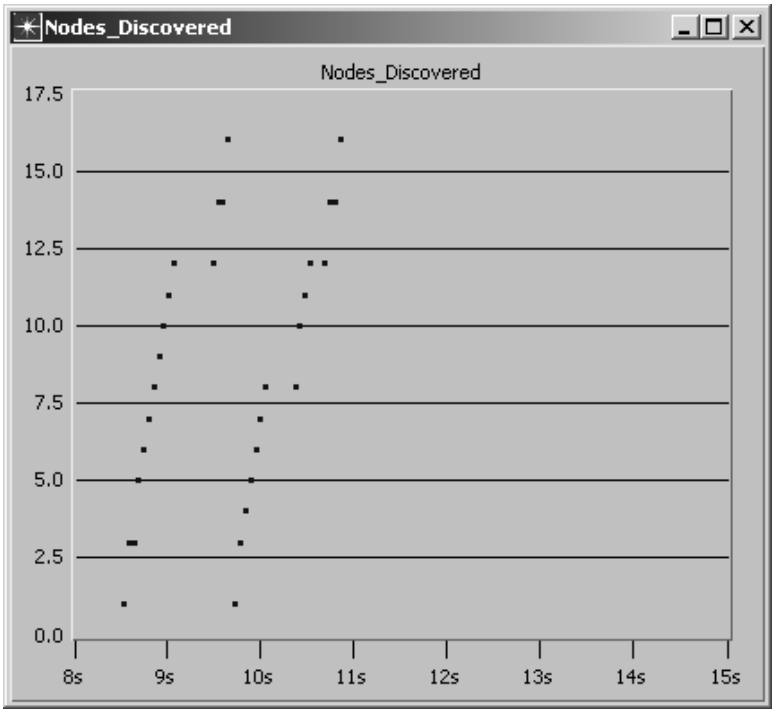


Fig. 3.17: Coverage for the 30 Nodes Grid Topology with the Source at Node 14

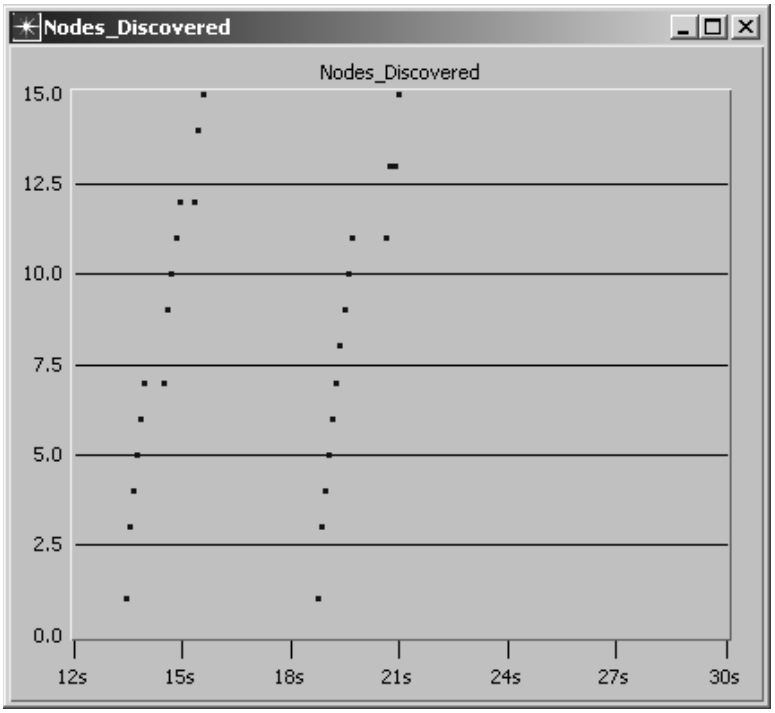


Fig. 3.18: Coverage for the 30 Nodes Grid Topology with Source at Node 1

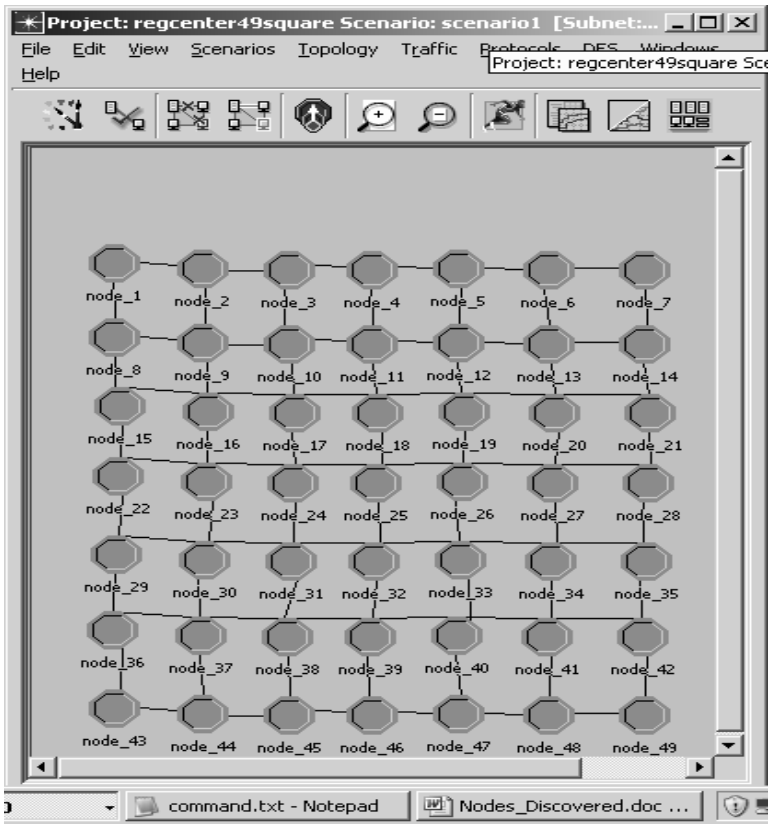


Fig. 3.19: 49 Nodes Grid Topology Network

Figure 3.19 shows a network of 49 nodes connected in a grid topology, and the coverage for this network when discovery packets are deployed from the middle at node 18, from one side at node 22, and from one corner at node 1 are shown in Figure 3.20, Figure 3.21, and Figure 3.22 respectively. The coverage for a non-symmetric network and two grid topology networks is shown together in Figure 3.23.

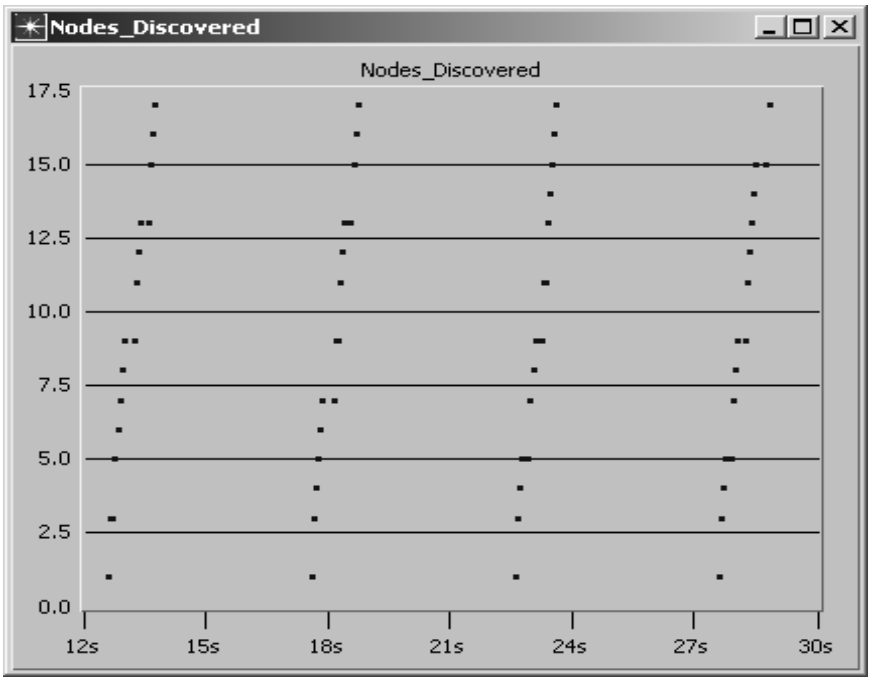


Fig. 3.20: Coverage for 49 Nodes Grid Topology with Source at Node 18

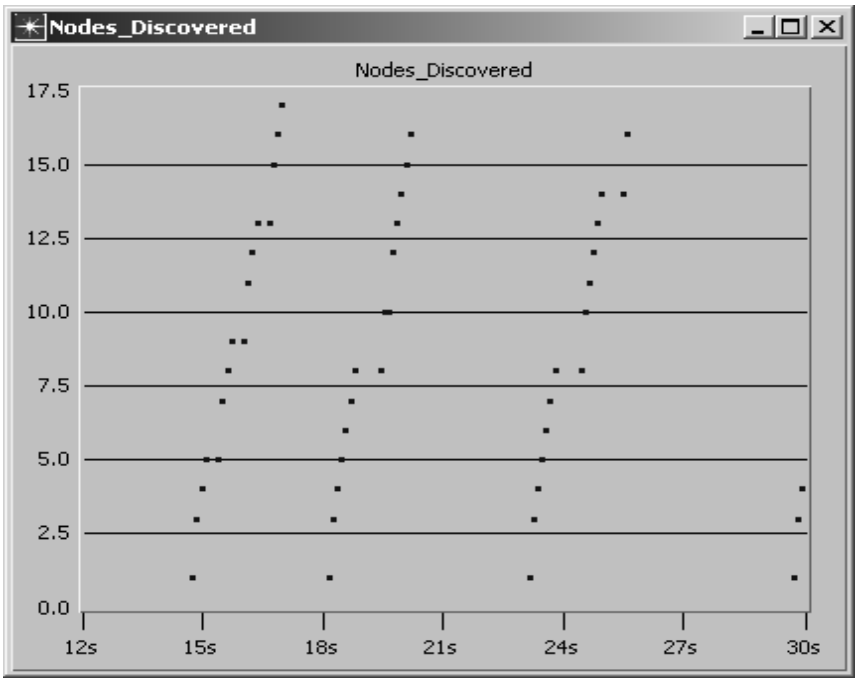


Fig. 3.21: Coverage for 49 Nodes Grid Topology with Source at Node 22

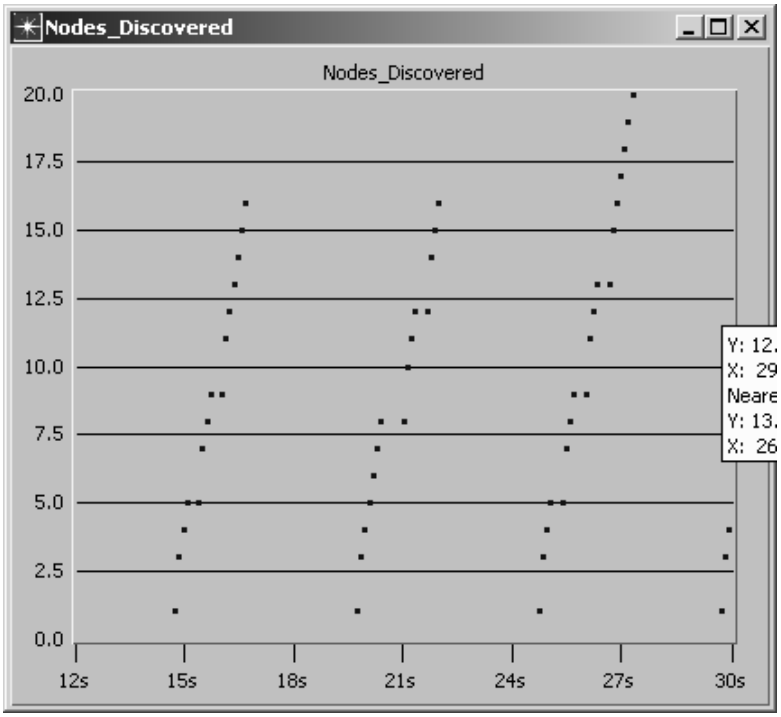


Fig. 3.22: Coverage for 49-Node Grid Topology Network with Source at Node 1

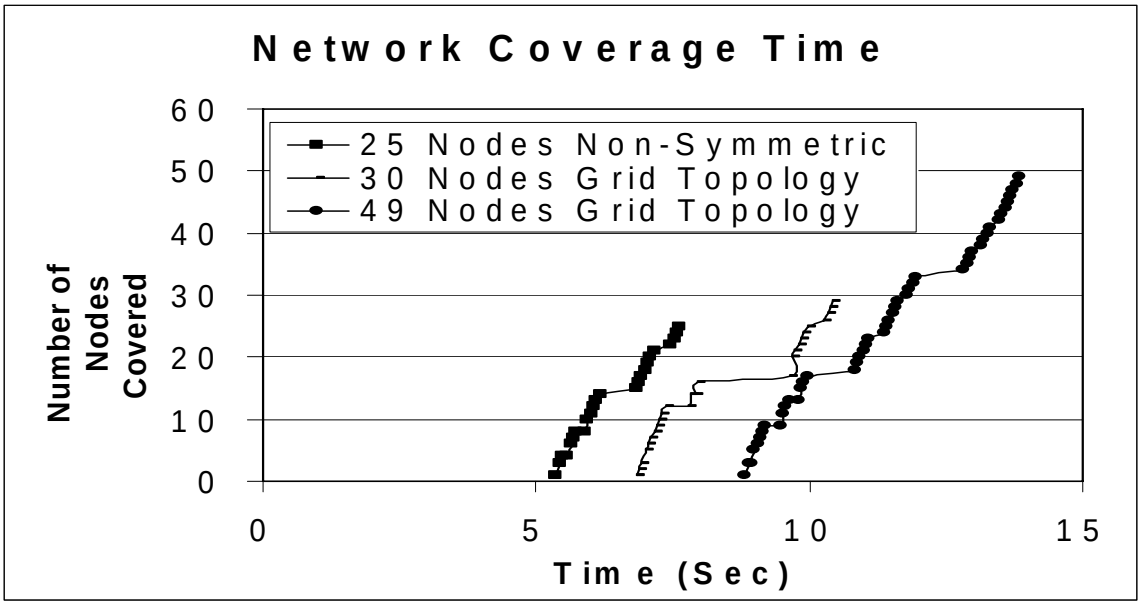


Fig. 3.23: Coverage for three different topologies

4. MODULAR MULTICASTING ALGORITHM

4.1 INTRODUCTION

Multicasting is a communication process where a single source transmits user data to more than one receiver. In a multicast network, sources send data traffic to a group of hosts represented by a multicast group address [44]. A multicast wireless host cannot base its forwarding decision on a destination address; a host in a wireless network must forward multicast data in a broadcasting mode in order to reach all possible receivers.

In order to forward multicast data to appropriate receivers, multicast routing algorithm protocols have to deal with many of the same issues as a unicast routing algorithm, such as learning the inter-network topology, detecting changes in the topology, and computing delivery paths in the topology. Forwarding mechanisms of multicasting are based on the source address, and to prevent loops, packets are discarded if they do not arrive from a designated forwarder host.

The goals of a multicast algorithm are to: a) minimize the number of steps needed to complete the multicast process, and b) minimize the total multicast delivery delay, i.e., the time between the beginning of the reception of a packet by the first member and the end of the reception of the same packet by the last member.

A multicast algorithm may be state based [38]. However, state based algorithms (either hard or soft) are not suitable for mobile networks [39] [40] because these networks are characterized by their random mobility. In some multicast algorithms for wireless networks, states or tables are kept for very short periods of time and therefore, are considered stateless. In order to deliver the multicast data, some protocols, such as the

On-Demand Multicast Routing Protocol (ODMRP) [12], construct a multicast mesh. Others, such as the Multicast Ad-hoc On-Demand Distance Vector protocol (MAODV) [13], construct a Distributed Shared Tree [31]. The Distributed Algorithm for Multicasting proposed herein is of the tree based form.

The majority of multicast protocols that have been written and implemented depend on the ability of a unicast routing protocol to find the best path to a destination. The Multicast Ad-hoc On-Demand Distance Vector protocol (MAODV) [13] is an extension of the AODV [14] unicast routing protocol. The Multicast Dynamic Source Routing Protocol (MDSR) [22] is an extension of the DSR [15] unicast routing protocol. The Lightweight Adaptive Multicast (LAM) protocol [16] depends on the Temporally Ordered Routing Algorithm (TORA) unicast protocol [17]. The Distributed Algorithm for Multicasting proposed herein does not depend on any unicast routing protocol; it is capable of discovering all nodes interested in the communication process and deliver the multicast data to them. The algorithm can be implemented in wired and low mobility wireless networks.

In some algorithms, routers have to compute spanning trees [20] to cover all other routers in the network; this technique lacks scalability to large networks. Neither receiver-based mode [34], nor sender-based mode [25] requires considerable storage capabilities on the routers, but their disadvantage is that sources do not have information on group membership. Another technique uses core-based trees [21] [32] [33] [42] where a single spanning tree is computed per group and a root is selected in the center of the group.

4.2 DISCOVERY PACKETS AND MODULARITY

Discovery packets are tools that can be used to make systems either modular or distributed, and their main function in this algorithm is to discover nodes that can be used as centers of those modular components. In modular systems, this approach allows the overall problem to be partitioned into a number of smaller and simpler components called modules, which are easier to develop and maintain, and which are specialized at solving sub problems. Packets act on behalf of their sources; they can be assigned destinations to involve mobility; they can collect objects inside a module, and they can be assigned a time to live or a maximum distance to travel. Discovery packets can be transmitted from a source and relayed from node to node; they can stop at remote locations to obtain the information sought, and they can then be transmitted back to their original source.

Discovery packets can encapsulate data packets, control packets, and knowledge provided by each node they visit. This information can include neighbors, particularly parents and children, nodes that have been visited, node capabilities in terms of energy and processing, location if available, data requests, etc.

Using discovery packets, and applying modularity, the congestion caused by overhead and the problem referred to as “acknowledgement implosion,” which causes performance bottlenecking at a multicasting source, can be distributed among several modules in the network. Furthermore, the multicast delivery delay can be reduced by distributing the data to all modules at the same time.

Initially, discovery packets are sent out from the multicast source with a double purpose: first, they must advertise a multicast session and second, they must explore the network in order to find appropriate locations for nodes to be used as module centers. After modularity has been established, the source distributes the multicast problem among the modules by using their centers as points for registration to the multicasting session, and as hubs for data distribution. The multicast algorithm can be repeatedly implemented at each module and the source interacts with its centers only.

Multicast communications involve flooding and broadcasting, especially when node mobility needs to be taken into consideration. These processes create contention and collision problems, and the redundancy involved leads to poor scalability. The problem of serious redundancy, creating what is referred to as the Broadcast Storm Problem has been studied [18], and some approaches to reduce such redundancy are presented in [18] and [19].

In the Distributed Algorithm for Multicasting presented here, neither flooding nor broadcasting are used during the discovery, advertisement or registration sessions. Instead, the information originates at a multicasting source and is passed from node to node in a sequential manner.

The goal in this session is to minimize the discovery time, which is the time required to propagate discovery (forward and backward) packets through a high percentage of nodes in the network. In order to achieve this goal, discovery packets will be deployed from a source at a periodic rate. The maximum number of hops such packets have to travel is specified in the packet itself. Initially, this number has a unity value, and then it is adjusted following one the following three expansions:

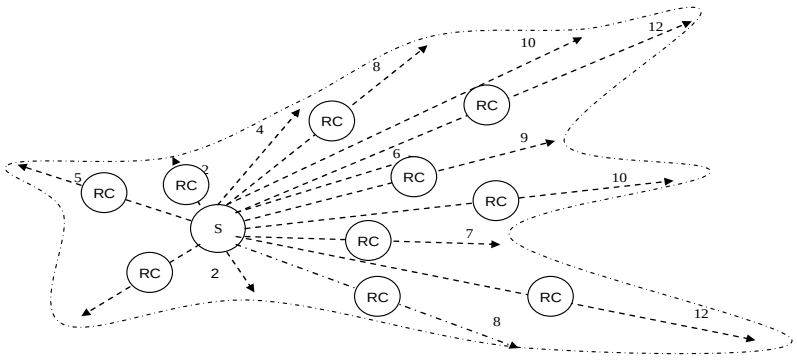
a) Unity Expansion

b) Binary Expansion

c) Square Expansion

A complete description of these expansion techniques was presented above in section 3.4. A modularity graph as established by the algorithm is presented below in Figure 4.1. The propagation of a forward discovery packet can be stopped at any node for one of the following reasons: a node does not have any neighbors, or all neighbors of the node have already processed a forward packet for the same discovery session. Any one of these events will trigger the generation of a backward packet to be sent out to the source.

Modularity



With the information provided by the backward agents, the source selects nodes in the middle of the paths to be used as regional or module centers.

Fig. 4.1: Nodes in the middle of the paths are selected as Registration Centers

The number of forward packets to be emitted at the source depends initially on the number of neighbors, and later on the information being received from backward packets.

The start of the algorithm is determined by the available power at the source. If power is sufficient, a packet will be sent through each neighbor at least once. Alternatively, if the number of neighbors is small, more than one packet will be sent through each neighbor. In the extreme case that the source has only one neighbor, multiple packets will be sent through it.

To increase coverage, the source releases a second round of packets to be propagated through a different path from that of the first round. Even though all neighbors of the source have processed forward packets during a first round of packets, they will be forced to re-process the packet during a second round of packets and send it to their unvisited neighbor nodes. In effect, this process sends those packets away from the source.

Four factors determine the number of rounds used by the algorithm in the discovery process: a) The available power at the source, b) the number of neighbors at the source, c) the number of hops the discovery packets are traveling (this information is provided by backward packets), and d) the coverage or transmission range of the source. It is not expected that discovery packets travel away in a straight line from the source. In fact, the probability that they take a curved path is higher than the probability that they take a straight path. To increase the probability of straight paths, forward packets embed in them the sender's neighbors list. This information is passed to the next node so that new discovery packets do not visit those neighbors. It was mentioned earlier that the source calculates the average number of hops after backward packets return; nodes located at this number of hops from the source will be used as the centers for the modules in this distributed algorithm. With the average number of hops calculated from each of the

backward discovery packets, and the number of nodes in each direction, the source obtains a good indication of the size and shape of the network.

4.3 DISTRIBUTED MULTICAST DELIVERY DELAY

In a network with N nodes on a branch in a multicasting tree, a Non-Distributed Algorithm will spend one unit of time to transmit data from a source to the first node, and N units of time to transmit data to the last node in the branch, therefore the multicast delivery delay which is the time between the beginning of the reception of a packet by the first group member and the end of the reception of the same packet by the last member will be N-1 units of time.

In our Distributed Algorithm, if we select one Registration Center in the middle of the branch, and data is sent to that center, the first node on both sides of the branch will receive data in one unit of time measured from the registration center, while the last node on both sides of the branch will receive data in N/2 units of time, making the multicast delivery delay equal to N/2 – 1. The delay between the source and the registration center is being neglected since we are only interested in the multicast delivery delay.

Figure 4.2 shows the comparison of the delivery delay when data is multicast in sequence from the first to the last node, and when data is sent to a node in the center of the network.

If we further select two Registration Centers on the branch, located in a way that each center covers one half of the network (i.e. first registration center at N/4 and the second one at 3N/4 nodes away from the source) and send data to those centers, the first node at each side of the first center will receive data in one unit of time measured from

the center, while the last node on each side of the first center will receive data at N/4 units of time. Furthermore, the first node on each side of the second center will receive data in N/2 + 1 units of time, while the last node on each side of the second center will receive data in N/2 + N/4 units of time (N/2 is the delay between the two registration centers).

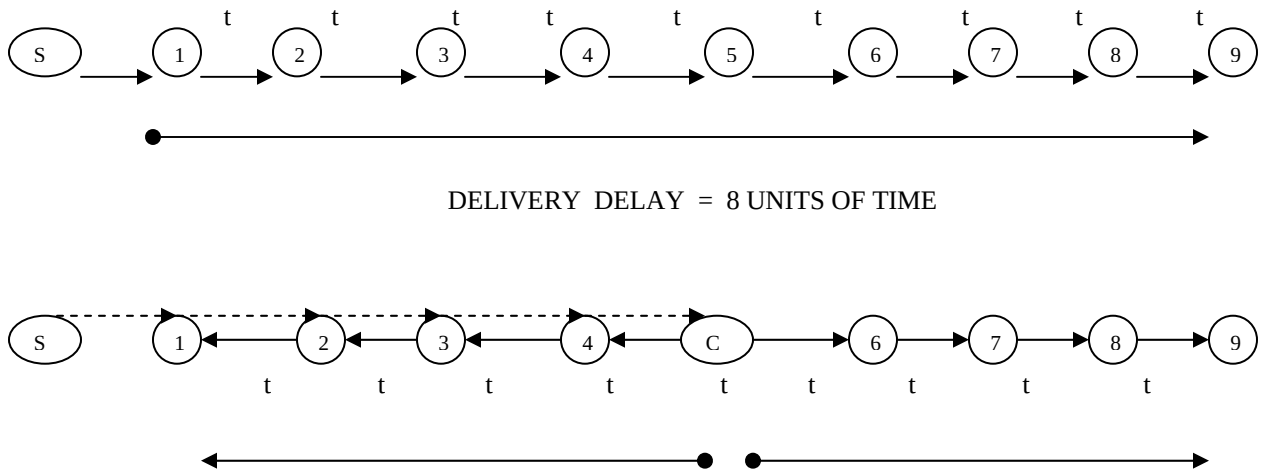


Fig. 4.2: Delay for Non-Distributed and Distributed Multicast Algorithms

The multicast delivery delay with two registration centers which is the difference between the time the last nodes in the second registration center receive data and the time when first nodes in the first registration center receive data will be: N/2 + N/4 -1 units of time.

In general, for a branch with N nodes and R registration centers distributed in a way that each center covers N/R nodes in the network, the time when the first node receives the multicast data from the first registration center is one unit of time measured from that center, while the time when last nodes receive data from the last registration center can be determined from the equation (4.1) shown below:

N(R-1)/R + N/2R (units of time) (4.1)

for R>=1

The first term in equation (4.1) is the delay from the first registration center to the last registration center, and the second term is the delay from the last registration center to the last node. The multicast delivery delay in general will be the difference between those two times and given by equation (4.2) shown below:

N(R-1)/R + N/2R – 1 (units of time) **(4.2)**

for R>=1

Further simplification of this equation will give us a general expression for the multicast delivery delay on a branch with N nodes and R registration centers equally distributed:

Distributed Multicast Delivery Delay = N – 0.5N/R – 1 (units of time) **(4.3)**

for R>=1

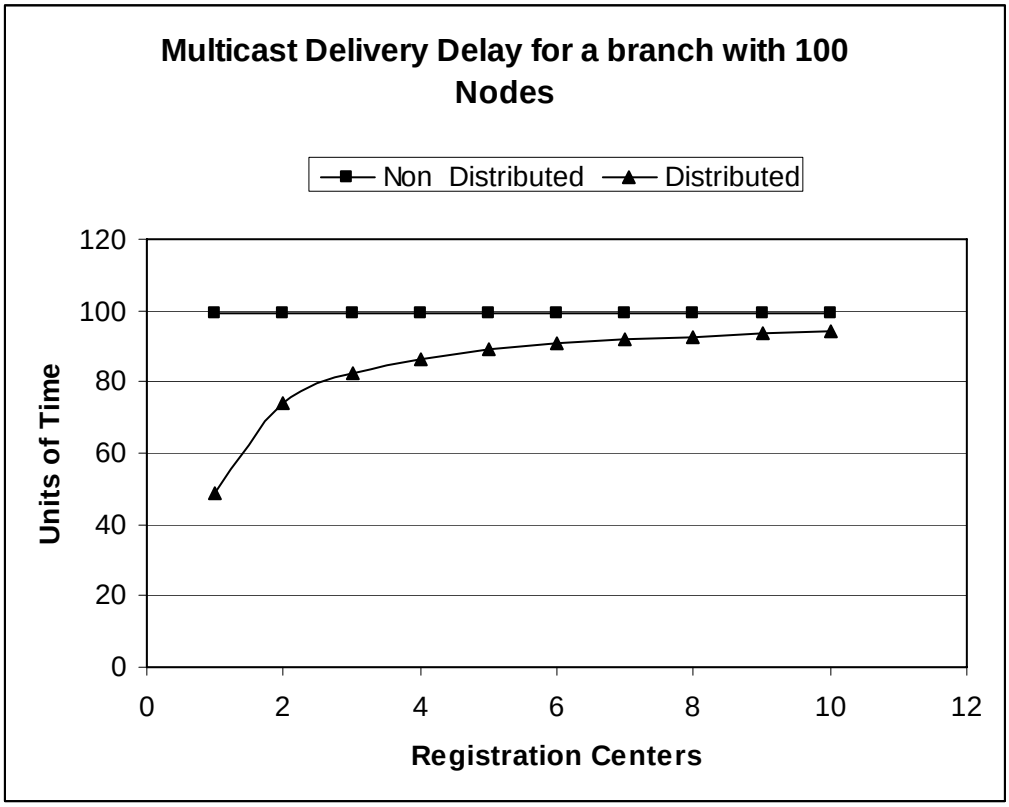


Fig. 4.3: Multicast Delivery Delay for a branch with 100 Nodes

This equation is plotted in Figure 4.3 for a value of N equal to 100 versus the number of registration centers. It should be noted that compared to the multicast delivery delay for a Non-Distributed Multicast Algorithm which is 100 – 1 units of time, this delay is significantly improved in our Distributed Algorithm for small values of registration centers. As the number of registration centers increases, the multicast delivery delay in the Distributed Algorithm increases approaching that in the Non-Distributed Algorithms. This graph shows the performance of the algorithm in only one branch. In large networks with several branches, the same performance is expected provided that the registration centers are equally distributed on each branch.

4.4 THE REGISTRATION SESSION

The nodes to be used as the centers for the modules are selected at the end of the discovery and advertisement session. The registration process for the multicast session involves two major steps. First, the source distributes the multicast information to the registration centers, and second, these centers work as registrations points distributing registration data to the nodes in their region by using registration packets.

Each registration center creates packets carrying the multicast information, and the number of hops to travel which has been determined by the source. These registration packets can be sent in sequence or they can be multicast to all of the center's neighbors. When a node receives the packet, it has the option to register for the multicast session; and if the number of hops is not at the maximum value, and the node has neighbors, the packet is re-sent to one of those neighbors. Each node records the path to the center of the

module as the packet passes by. This path information can be used later for data recovery or tree reconstruction.

The packet carrying the registration information travels until either the number of hops has reached the maximum, or the packet reaches a node whose neighbors are the same as those of the sender. When either of these two situations occurs, a backward registration packet is generated and sent back to the module's center, following the reverse path of the forward packet. Nodes in this path are informed of members immediately down the branch that have registered for the multicast session, establishing a parent-child relationship. Even if a node is not participating in the session, it will still be recorded to serve as a forwarder so that it can send registration packets down the branch. This is precisely how nodes are added to the distribution tree. As the registration process continues, additional branches are added. It is the arrival of a backward packet to the center of the module that results in the formation of a new branch for the multicasting tree. If a node is not interested in participating in the session but one of its children nodes is a member or a forwarder, this node is included in the branch and its role becomes that of a forwarder during the multicast data session. When all neighbors of the registration center have processed backward packets, the center is provided with information about members and non-members for the session on all branches.

After the distribution tree has been formed, the registration centers contact the multicast source to initiate sending the multicast data to be distributed in each module. Registration packets will still be emitted at each module center, but at a lower rate. Figure 4.4 illustrates the formation of a branch of the multicast tree during a registration session. Figure 4.5 shows a flow graph of registration packets.

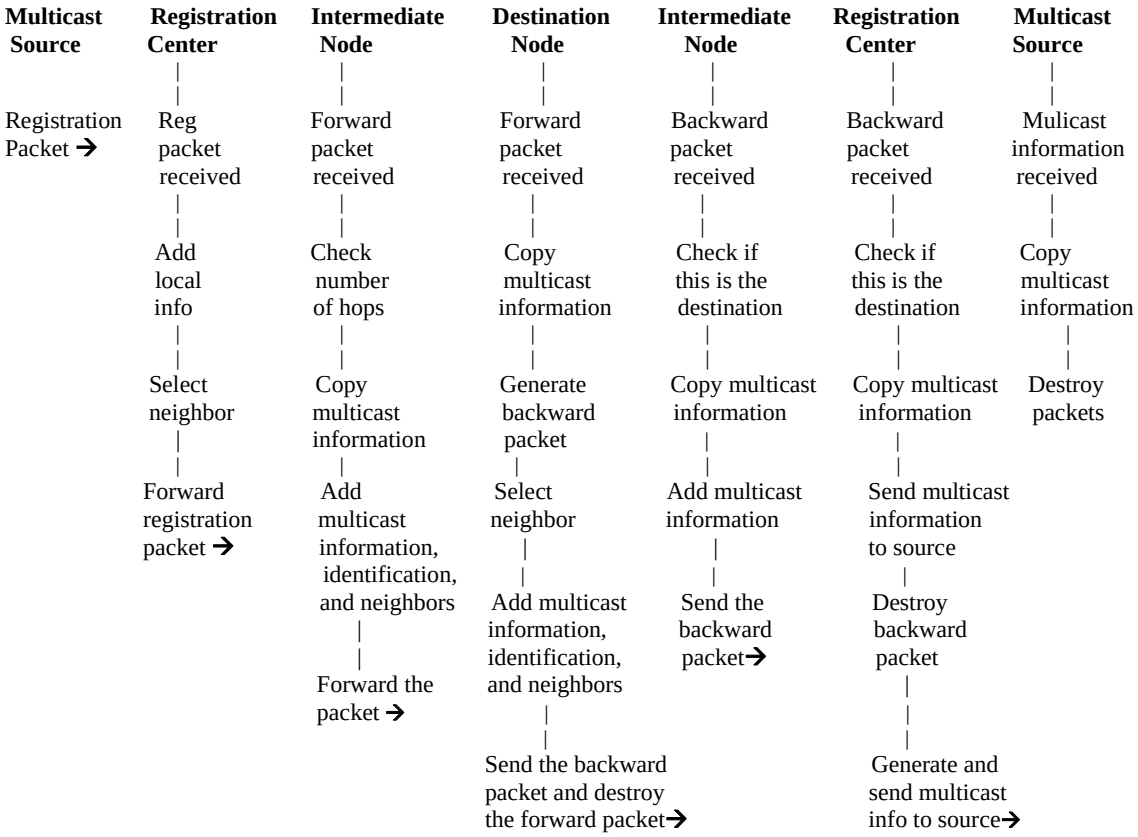


Fig. 4.4: Processes at the nodes during the Registration Session

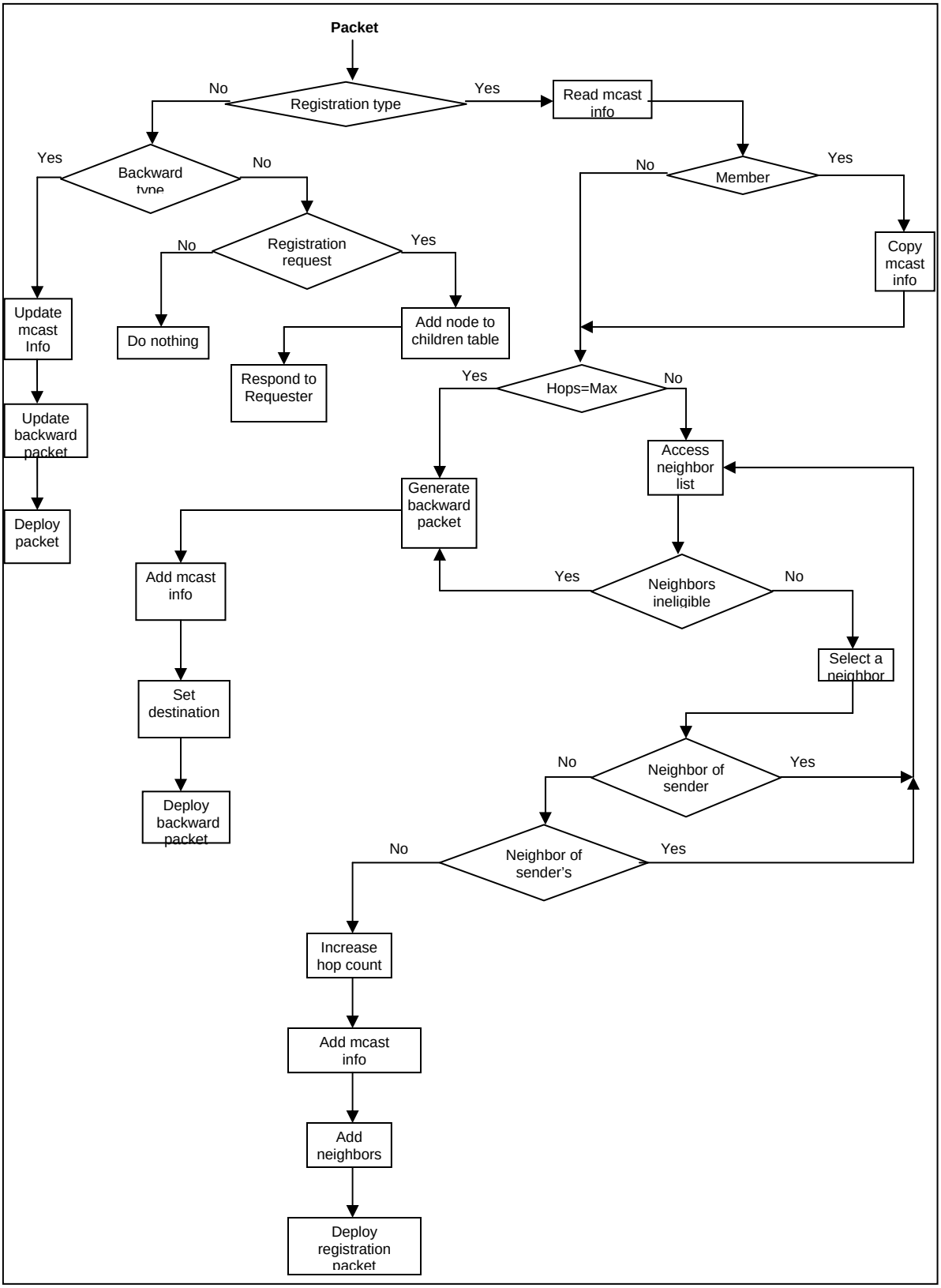


Fig. 4.5: Registration Session Flow Chart

4.5 THE DISTRIBUTED MULTICASTING PROCESS

The complete multicast process depicted in Figure 4.6, is as follows: A multicast source delivers a packet to travel up to the last node in the perimeter of the network; the propagation of this packet is controlled in the Network Discovery and Multicast Advertisement Forward Packets (fwd) module.

The last node creates and delivers a backward packet destined to the multicast source; the propagation of this packet is controlled in the Backward Discovery Process (bkwd) module. The multicast source creates and delivers a packet destined to a Registration Center; this packet contains registration information and is controlled in the Registration Packets from the Multicast Source to a Registration Center (reg_center) module. The Registration Center creates and delivers registration packets destined to all nodes around it; the propagation of these packets is controlled in the Registration Advertisement Packets (reg_center_adv) module. When all nodes around a registration center are visited by registration packets, backward packets containing registration for the multicast session are generated and delivered back to the registration center; the propagation of these packets is controlled in the Backward Registration Packets (reg_center_bkwd) module. Once a Registration Center obtains registration confirmation from nodes around it, it creates and delivers a packet destined to the multicast source. This packet contains information about registered nodes and its propagation is controlled in the Registration Information from Registration Center to the Multicast Source (reg_to_mc) module. When the multicast source obtains verification of registration for a multicast group it creates and delivers data packets destined to the registration center; the

propagation of these packets is controlled in the Multicast Source to Registration Center (mc_to_reg) module.

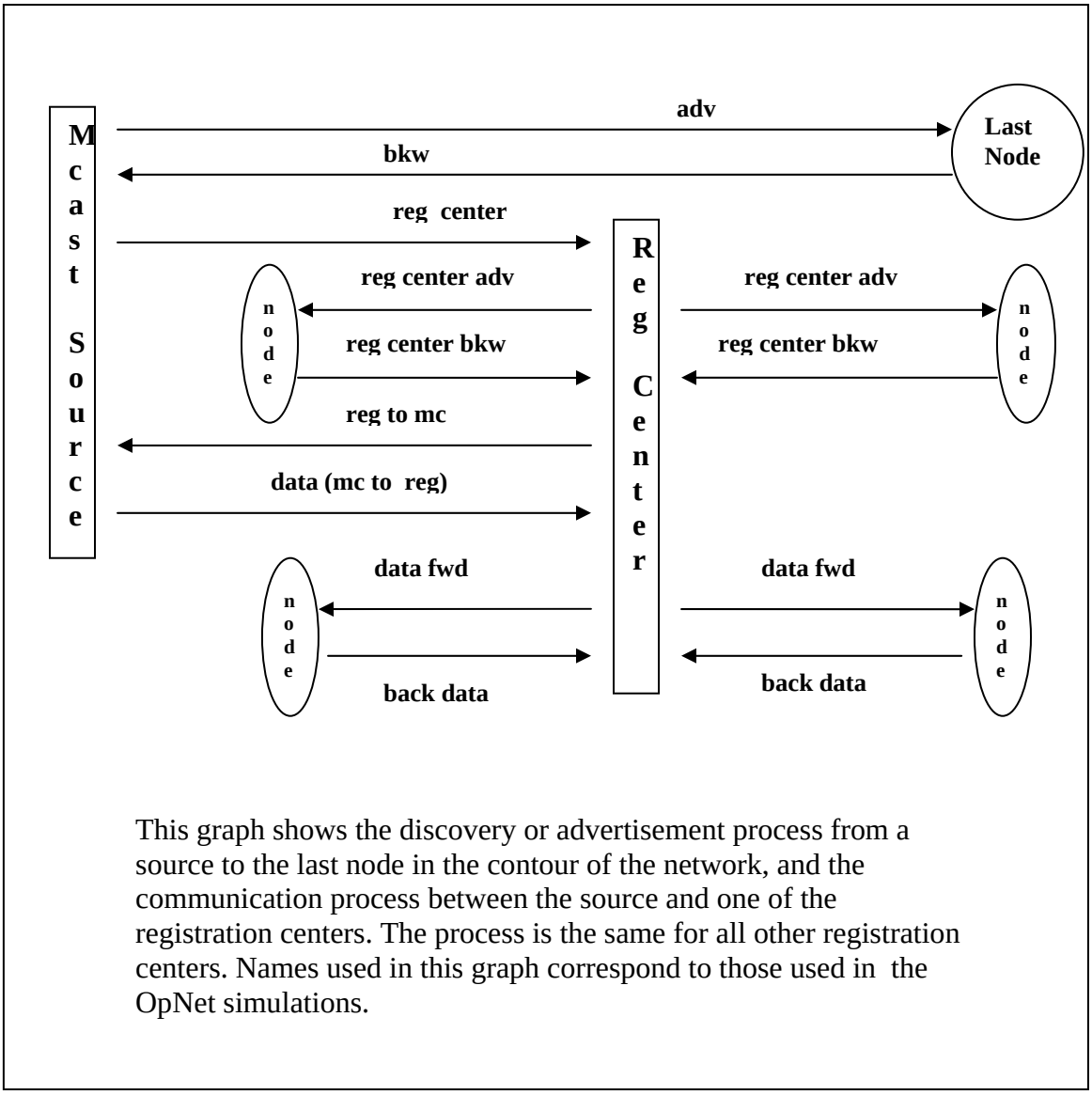


Fig. 4.6: The complete Distributed Multicasting Process

Once data packets arrive at a registration center, this center proceeds to deliver those packets to all registered nodes around it; the propagation of these packets is controlled in the Data Distribution (fwd_data) module. Once a data packet has been delivered to the last node in the neighborhood of a registration center, a backward packet is created and delivered back to the registration center; the propagation of these packets is controlled in

the Backward Data Packets (back_data) module. Figure 4.6 shows the flow of all these packets and the correlation among all modules mentioned above. The following subsections in this section describe each of these processes.

4.5.1 NETWORK DISCOVERY AND PROPAGATION OF MULTICASTING FORWARD ADVERTISEMENT PACKETS

This module makes network discovery possible and at the same time enables the advertisement of a multicast session to be advertised as nodes are visited by discovery packets.

A discovery or discovery-advertisement packet can arrive at this module across two different places: either from the local packet generator source in the node itself or from another node. If the packet is emitted from the local source, the node declares itself as the multicast source and proceeds to set the following parameters in an advertisement or discovery packet: source id, multicast source id, multicast session number, destination id, packet number, id of all its neighbors, the maximum number of hops the packet can travel, as well as the *nodes_visited[0]* element of the visited nodes array. The visited nodes count and the log for the number of packets sent are initialized to one. At this point, the packet is sent to all its neighbors in sequence.

If the discovery or advertisement packet arrives from a different node, the current node retrieves the following information from the incoming packet: multicast session number, source id number, number of hops the packet has traveled, the next return for the packet; it then identifies the neighbors of the sender and stores them in an array (*sender_nbr[i]*, where i varies from 0 to the number of neighbors), and also identifies the visited nodes

and stores them in an array *(nodes_visited[i]*, where i varies from 0 to the maximum number of nodes to be visited).

The current hop count is then tested against the maximum number of hops specified in the packet. If the hop count is equal to the maximum number of hops, the node sets the number of traveled hops as the hop count. It then decreases the hop count by one, updates the back sender (*bk_sender[i],* where i varies from 0 to the number of different returning packets) array, and sets the packet type number as the number that identifies a backward packet, and returns the packet to the stream from which the discovery packet emanated. The *bk_sender* array in a node stores the nodes which have sent backward packets to it.

If the hop count at the current node is not equal to the maximum number of hops, the current node performs the following operations: it counts the number of neighbors, counts the number of back senders, and selects a receiver or receivers for the packet. For a node to be selected as a receiver it cannot be a neighbor of the sender which are stored in an array called *sender_nbr*, and it cannot be a neighbor of the sender's neighbors which are stored in an array called *my_nbr_nbr*. The current node sets all elements of the *my_nbr_nbr* arrays containing its own id to -1. The sentinel value of -1 is required to eliminate the current node from those arrays, since obviously this node is a neighbor of all its neighbors. If this action was not taken, no neighbor would be eligible for forwarding, since the outward propagation mechanism requires that neighbors of sender's neighbors can not be eligible as possible receivers.

Identifications of eligible receivers are stored in an array *receivers[i]* (with 0<=i<=3) and the number of receivers is stored in the <*receivers_count*> variable.

A receiver_count equal to zero indicates that there are no eligible receivers for the packet and a backward packet is generated. On the other hand, if the value of *receiver_count* is different than zero, the current node sets the source id as its own id, updates its neighbors into the packet, increases the number of hops by one, and proceeds to select a receiver for the packet from the list of eligible receivers. In order to select a receiver from the eligible receivers, the current node checks the *back_sender* and receivers arrays. The back sender's array stores nodes sending backward packets towards the multicast source and the receiver's array stores nodes that were determined as eligible to receive the forward packet. A value of zero in the *back_sender's* count indicates that the packet is moving forward for the first time. A value of one indicates that the current node has been visited by only one backward packet, and a value of two indicates that the node has been visited by two backward packets coming from different nodes (i.e. two branches).

If the value of back senders is equal to zero, and the receivers' list is not empty, the current node selects a receiver from this list to which it forwards the packet. If the back sender value is one, and there is more than one receiver eligible to receive the packet, the current node selects as the receiver a node that does not show its id in the back senders' array. If the value of back senders is higher than one, the current node selects as the receiver for the packet that node which has not received the packet, (a node that shows its id in the receiver array and, in turn, is not present in the sender's array). In other words, the back sender's array maintains a list of nodes which have already seen the packet. Once the receiver has been determined, the forward discovery or advertisement packet is sent to the stream connecting to that receiver. The pseudo-code for this module is appended in appendix C.

Discovery packets are illustrated in Figure 4.7. These packets are used for two purposes: Firstly they are used to advertise the multicast session, and secondly they are used for discovering the size and contour of the network. These packets with different packet types are used for both forward and backward packets. They have the following fields:

Type: This field identifies the packet. Different numbers identify forward and backward packets.

Packet_no: This field holds the number of packets being used.

Source_id: This field identifies the node generating these packets.

Forwarder: This field identifies the node forwarding these packets.

Dest: This field is used by backward packets to identify the final destination for the packet.

Mc_session: This field identifies the multicast session taking place.

Mc_source: Identifies the node generating multicast data.

Hops: This field keeps the number of hops traveled by the packet.

Nbr_1 through nbr_4: These fields contain the neighbors of a node forwarding this packet.

My_nbr_nbr_1_1 through my_nbr_nbr_4_4: These fields contain the sender's neighbors' neighbors.

Node_visited_1 through node_visited_10: These fields contain the identification of the nodes visited by the forward packet.

Reverse_path_hops: This field is used to keep the count of hops traveled by backward packets.

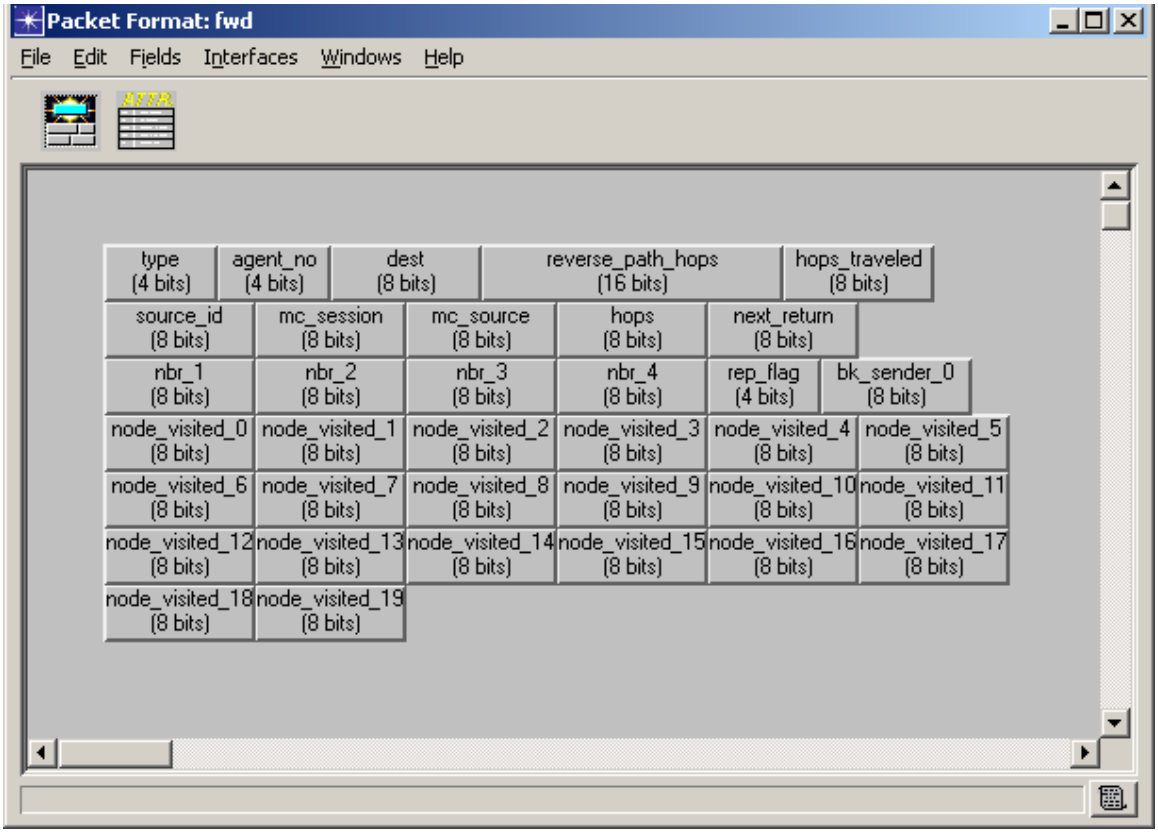


Fig. 4.7: The Discovery Packet

4.5.2 PROPAGATION OF BACKWARD DISCOVERY PACKETS

When a forward discovery packet has traveled the maximum number of hops specified in the packet, or when the packet reaches a node that has no neighbors, or all its neighbors have already seen the packet, the current node generates a backward packet to be sent back to the source.

When a backward packet arrives at a node, the control of the algorithm is transitioned to this backward discovery process module. Once in this module, the node examines the final destination for the packet. If the destination is not the current node, the number of

reverse hops is obtained and decreased by one. The current node then gets the reverse path and determines the next node in that path to which to relay the backward packet.

If the final destination for the backward packet is the current node, the current node gets the id of the visited nodes, gets the hop count, gets the next return, and also gets the number of hops the packet has traveled. The node now proceeds to check the hops traveled against the next return. The value of the next return parameter depends on the mechanism used to control the flow of forward packets. Expansion mechanisms used in this work include unity, binary, and square increments as well as a go up to last node technique.

If the hops traveled is equal to the next return or to the maximum number of hops, the packet has been forced to return and the current node acts as follows: it resets the *nodes_visited* array, sets itself as the first node visited, sets itself as the multicast source and source id, and while also setting itself as the destination for the next backward packet, embeds its neighbors into the packet, and sets the packet type number as the number that identifies a forward packet. Then, the packet is returned to propagate in the forward direction.

If the destination id for the backward packet is equal to the current node's id but the number of hops traveled is equal neither to the next return nor to the maximum number of hops, this situation indicates that the last node visited did not have any neighbors to which to forward the packet. In this case, the current node, which is the multicast source, gets and counts the number of nodes visited to select a registration center for the multicast session. This registration center will be located at the mean value of the number of nodes visited. From the array holding the visited nodes, the source determines the path

to the registration center, and, before it forwards the packet, includes the path to be used by other nodes into the packet. The source then proceeds to set the destination for returning packets as well as the source id. The source also sets the packet type number as the number that identifies a multicast to registration center packet; the packet is then sent to the stream connecting the first node in the path to the designated registration center. The Pseudo-Code for this module is appended in appendix D.

4.5.3 PROPAGATION OF REGISTRATION PACKETS FROM THE MULTICAST SOURCE TO A REGISTRATION CENTER

After a backward discovery packet arrives at the multicast source and this source has determined the visited nodes and the registration center, a packet is then sent, with the destination set to that registration center. The mission of this packet is to inform the registration center about its new designation. This packet has the maximum number of hops that registration packets will travel, which is the mean value of the number of nodes visited by the original discovery packet. When a node receives this type of packet, it examines the destination node. If the destination is different from the node's id, the current node retrieves the path to the registration center from the packet to determine the next node in the path. Once the next node is determined, the packet is forwarded to the stream connecting that node. Equality between the destination node and the current node's id indicates that the packet has arrived at the designated registration center. From the nodes visited by this packet, the registration center determines the reverse path to the multicast source and stores it in the *path_to_mc[i]* array (where i varies from 0 to one half of the number of nodes visited). This path will be useful whenever the registration center needs to contact the multicast source. The control is then passed to the registration

center advertisement **(***reg_center_adv)* module. This module distributes registration packets as described in the upcoming section, entitled Registration Advertisement Packets.

4.5.4 PROPAGATION OF REGISTRATION ADVERTISEMENT PACKETS

Registration centers are charged with offering registration for the multicast session to all branches of nodes connected to its neighbors. When a registration packet deployed at the multicast source arrives at a node in the path to a registration center, the node examines the destination node for the packet.

If the destination is, in fact, the current node, this is an indication that the current node is the registration center and the node proceeds to do the following: sets destination for returning packets; sets source id; resets the hop count to zero; initializes the packet number to one; resets the visited nodes array to its initial value; resets the registered nodes array to its initial value; and sets its neighbors into the packet. The packet type number is set as the number that identifies a registration advertisement packet. The packet is then sent to all neighbors of the registration center, to be distributed throughout all its neighboring branches.

When a registration packet arrives at a node in the current module, this node gets the multicast session number, gets the source id, gets and increases the hop count, gets the sender's neighbors, gets and updates visited nodes, and then it checks if the node is already registered for the multicast session. If the aforementioned check returns a positive value, the process is sent to a control point in this module called: already registered. On the other hand, if the check returns a negative value, the node is not registered and a

decision concerned with registration must be made. A random number generator is used at this point to decide registration. If the number generated is higher than a threshold value, the node will be registered; otherwise the node won't register and the program will transition to the already registered control point in this module. To register, a node will include itself in the *registered_nodes* array. Once the registration process is completed, the hop count is checked against the maximum number of hops.

If the hop count is equal to the maximum number of hops for registration, the current node performs the following operations: it sets hops traveled, decreases the hop count, and sets the reverse path. The packet type number is set as the number that identifies a backward registration advertisement packet, and the packet is then sent to the stream connecting the first node in the reverse path.

If the hop count is not equal to the maximum number of hops for registration, the current node does the following: sets all elements of the *my_nbr_nbr* arrays containing its own id to -1. These arrays hold the neighbors of the current node's neighbors. The sentinel value of -1 is required to eliminate the current node from those arrays, since obviously this node is a neighbor of all its neighbors. If this action was not taken, no neighbor would be eligible for forwarding, since the outward propagation mechanism requires that neighbors of sender's neighbors can not be eligible as possible receivers. After an eligible node is determined for forwarding the packet, the current node sends the packet to the stream connecting to that node.

In the event that none of the nodes is eligible to receive the packet, or all neighbors of the current node have received the registration packet, a backward packet is generated and sent back to the registration center. Figure 4.8 shows the formation of a branch as a result

of the propagation of forward and backward registration packets. The pseudo-code for the registration process is appended in appendix E.

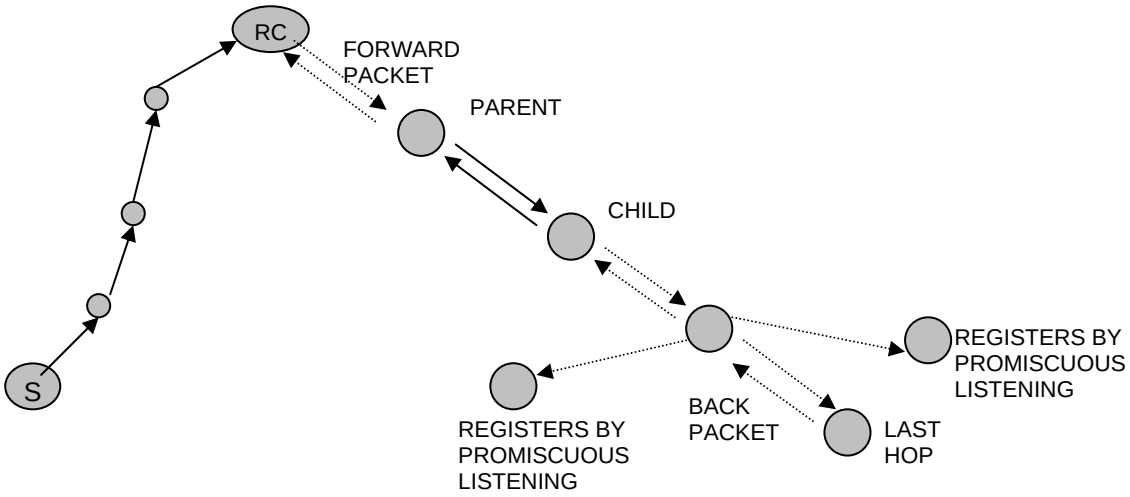


Fig. 4.8: Formation of a Registration Branch

Forward Registration packets are illustrated in Figure 4.9. These packets are used to offer registration to visited nodes and register those nodes which decide to participate in the multicast session. These packets with different packet types are used for both forward and backward packets. They have the following fields:

Type: This field identifies the packet.

Packet_no: This field holds the number of packets being used.

Reg_center_id: Identification of the registration center node offering registration

Reverse_path_hops: This field is used to keep the count of hops traveled by backward packets.

Source_id: This field identifies the node generating these packets.

Forwarder: This field identifies the node forwarding these packets.

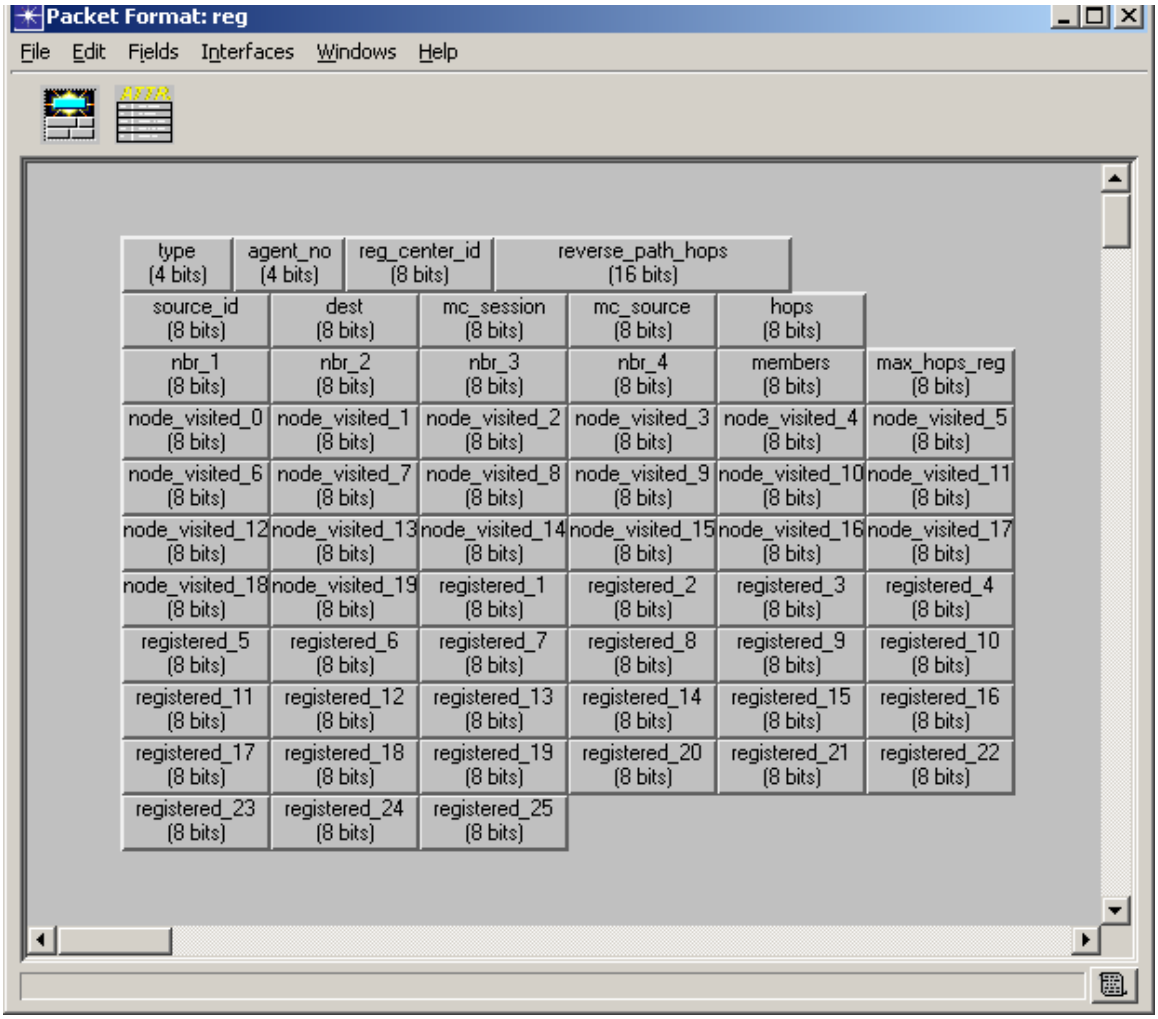


Fig. 4.9: Forward Registration Packets

Dest: This field is used by backward packets to identify the final destination for the packet.

Mc_session: This field identifies the multicast session taking place.

Mc_source: Identifies the node generating multicast data.

Hops: This field keeps the number of hops traveled by the packet.

Nbr_1 through nbr_4: These fields contain the neighbors of a node forwarding this packet.

Members: Number of nodes that have registered for the multicast session.

Max_hops_reg: Maximum number of hops a registration packet can travel.

Node_visited_0 through node_visited_19: These fields contain the identification of the visited nodes by the forward registration packet.

Registered_1 through Registered_25: These fields contain the identification of nodes registered for the multicast session.

4.5.5 PROPAGATION OF BACKWARD REGISTRATION PACKETS

When a forward registration advertisement packet has traveled the maximum number of hops specified in the packet, or when the packet reaches a node that has no neighbors, or all its neighbors have already seen the registration packet, the current node generates a backward registration packet to be sent back to the registration center.

When a node receives a registration backward packet destined to a registration center, it first gets and examines the destination node from the packet. If the destination is not equal to the current node's id, this node does the following: it gets the reverse path; decreases the number of reverse hops; sets the new reverse path; determines the next node in the reverse path, and sends the packet to the stream connecting that node.

If the final destination is the current node's id, the current node which is the registration center gets and stores nodes registered. It then counts and logs new members (registered nodes) to the multicast group, and also counts the number of neighbors and the number of deployed packets.

If the number of neighbors is equal to the number of deployed packets, this indicates that registration data has been sent throughout all branches originating at the registration center node. This being the case, the current node proceeds to access the path to the multicast source. This path was recorded when the multicast source first contacted the

registration center. Once the path is determined, the registration center sets the following: the reverse path to the multicast source; the number of reverse hops; nodes registered, source id, and destination; the packet type is set as a number that identifies a registration center-multicast source packet. The registration center then sends the packet to the first node in the reverse path to the multicast source.

If the number of neighbors is not equal to the number of deployed packets, this indicates that not all branches from the registration center have received the registration packets. Accordingly, the control is sent to the registration center advertisement (*reg_center_adv*) module to continue the task of distributing registration packets to other branches. The pseudo-code for the propagation of the backward registration packets is appended in appendix F.

4.5.6 REGISTRATION INFORMATION FROM REGISTRATION CENTERS TO THE MULTICAST SOURCE

When a registration center has sent registration data through all its neighbors or branches and acknowledgements have been received, it then prepares a packet to be sent to the multicast source.

When a packet from a registration center destined for the multicast source arrives at a node, this node first obtains and examines the destination node from the packet.

If the destination id is not equal to the current node's id, the number of reverse hops is obtained and decreased by one. The reverse path is then obtained to determine the next node in that path. Once the next node is determined, the packet will be relayed to the stream connecting to that node.

If the destination in the packet is the current node's id, this indicates that the current node is the multicast source. The source first obtains and logs the path to the registration center. It then collects the number of members for the multicast session, updates the multicast group, creates a data packet, sets the source id, sets the packet type that identifies a data packet, and sends the packet to the stream connecting the first node in the multicast source registration center path. The control of this data packet is described in the following section. The pseudo-code for the propagation of packets from registration centers to the multicast source is appended in appendix G.

4.5.7 DATA FROM THE MULTICAST SOURCE TO REGISTRATION CENTERS

When the multicast source receives a packet from a registration center, it gets the destination node id from the packet (*reg_center_dest*), and then gets the path to the registration center (*path_to_rc[i]* where i can range from 0 to one half the number of nodes visited). Then it gets the registered nodes from the packet to determine if multicast data has to be sent to the registration center. If data needs to be sent, the source creates a data packet, sets the path to the center, sets the destination for the packet, and sends the packet to the stream connecting the first node in the path to the registration center.

When a node receives the data packet destined to a registration center, it compares the destination id for the packet against its own id. If the destination id is equal to the node's id, meaning that the current node is the actual registration center, the control of the program is passed to the forward data *(fwd_data*) module for data distribution. If the destination id is not equal to the current node's id, the current node retrieves the path to

the registration center from the packet to determine the next node in the path. Once this node is determined, the packet is sent to the stream connected to it.

Packets from the multicast source to a registration center including data are illustrated in Figure 4.10. These packets will be used by registration centers to distribute user data to those nodes which are registered for the multicast session. These packets have the following fields:

Type: This field identifies the packet.

Reg_center_id: Identification of the registration center distributing data

Source_id: This field identifies the node generating these packets.

Dest: This field is used to identify destination for returning packets.

Mc_session: This field identifies the multicast session taking place.

Mc_source: Identifies the node generating multicast data.

Hops: This field keeps the number of hops traveled by the packet.

Nbr_1 through nbr_4: These fields contain the neighbors of a node forwarding this packet.

Max_hops_reg: Maximum number of hops a registration packet can travel.

p_mc_reg_0 through p_mc_reg_19: These fields contain the identification of nodes in the path from the multicast source to the registration center.

mc_data_0 through mc_data_3: These fields contain data to be delivered to members.

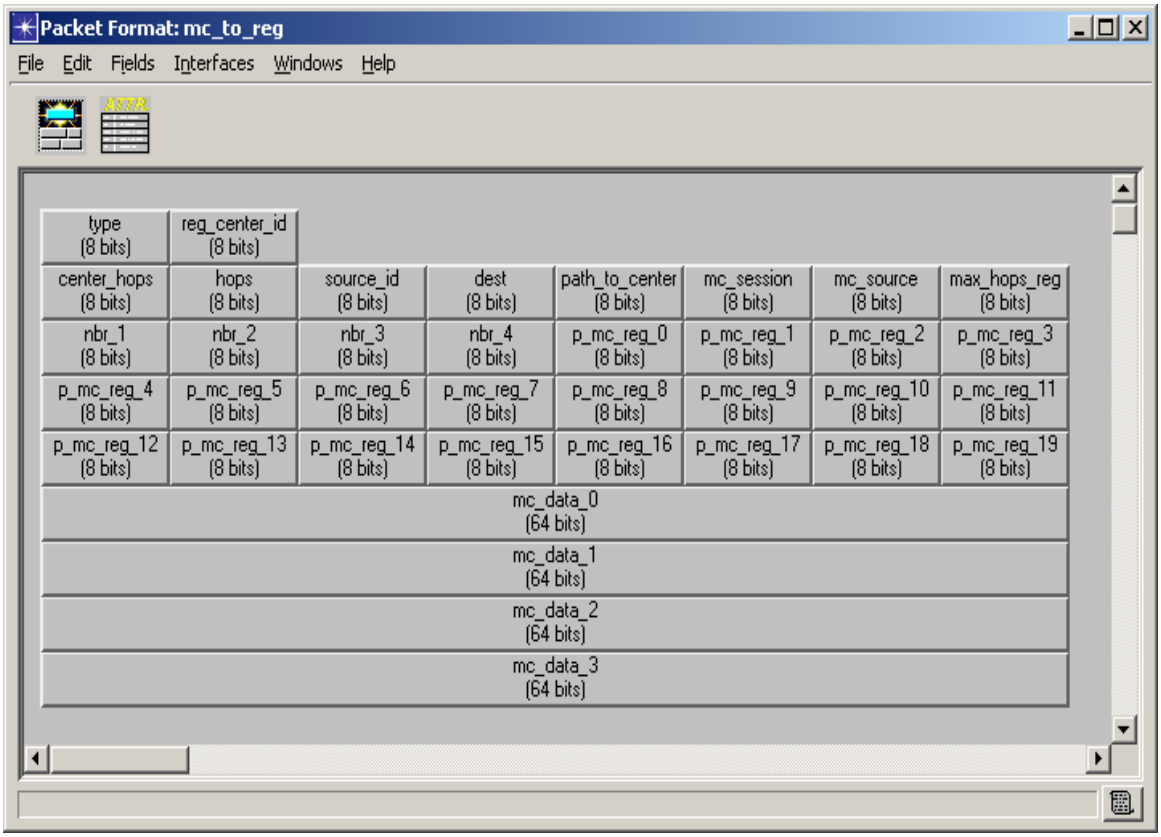


Fig. 4.10: Packet including data from the multicast source to Registration Centers

4.5.8 DATA DISTRIBUTION

When a data packet arrives at a registration center from the multicast source, this center proceeds to distribute that packet to all nodes that registered for the multicast session. The data distribution process is as follows: the registration center node sets its *node_id* as the destination for returning packets. The neighbors of this registration center node are then included into the packet. The packet type number is set as the number that identifies a data packet, the number of hops is set to zero and the number of nodes visited is also set to zero. The maximum number of nodes to be visited by a data packet has been pre-determined by the multicast source. After setting the aforementioned parameters, the

packet is multicast to all neighbors of the registration center to be distributed throughout all branches.

When a data packet arrives at a node, this node checks an array called registered[i]. The presence of its *node_id* as an element of the array is an indication that this node is registered for the multicast session and the node proceeds to extract the data from the packet. For the purpose of relaying the packet, the recipient node logs the neighbors of the sender, increases the hop count by one and then checks for the maximum number of hops the data packet has to travel. If the hop count has reached the maximum, the number of hops traveled is set as the hop count, the hop count is decreased by one, the packet type number is set as the number that identifies a backward data packet, and the packet is sent through the same stream from where the packet was received. On the other hand, if the hop count is not at its maximum, the recipient node has to find a neighbor to which it can forward the packet. In order to comply with the outward propagation, the selected node cannot be the sender, and it cannot be a neighbor of the sender. In order to achieve this task, arrays that contain the neighbors of the sender (*sender_nbr[i]* where i varies from 0 up to the maximum number of neighbors) are compared against arrays that contain the neighbors of the current node neighbors *(my_nbr_nbr_i[j]* where i and j vary from 0 up to the maximum number of neighbors). For a neighbor node to be selected as a receiver for the data packet, none of its neighbors can be neighbors of the sender. Once the arrays are compared and a node is found as an eligible recipient for the packet, the current node sets its neighbors, sets the hop count, and sets the packet type number that identifies a data packet. The packet is then sent to the stream connecting the selected node.

A modification to this module has also been implemented as follows: the registration center has determined previously in the Registration Advertisement Packets section the last node in a branch which registered for the multicast session. When a node receives the data packet from the registration center, the node checks the last node's id against its own id. If the ids are different, the node retrieves the multicast session number from the packet and then accesses the node's registered array to determine if it is registered for the session. If the node is registered, it retrieves the data from the packet, and then from the nodes visited array determines the next node in the path to the last node. The packet is then sent to the stream connecting the next node in the path.

If the id of the current node's id is the id of the last node, the node then determines if it is registered for the multicast session. If it is registered, it proceeds to retrieve the data from the packet and the data packet is then sent to the sink.

This variation of the present module does not provide the return of any confirmation for packet reception (acknowledgement) to the registration center as the first part described above in this module does, therefore there is no source registration. The propagation of acknowledgement messages is described in the next section. The pseudo-code to control the propagation of data distribution packets is appended in appendix H.

Data packets are illustrated in Figure 4.11. These packets are used to distribute data to registered nodes which have decided to participate in the multicast session. They have the following fields:

Type: This field identifies the packet.

Reg_center_id: Identification of the registration center offering registration

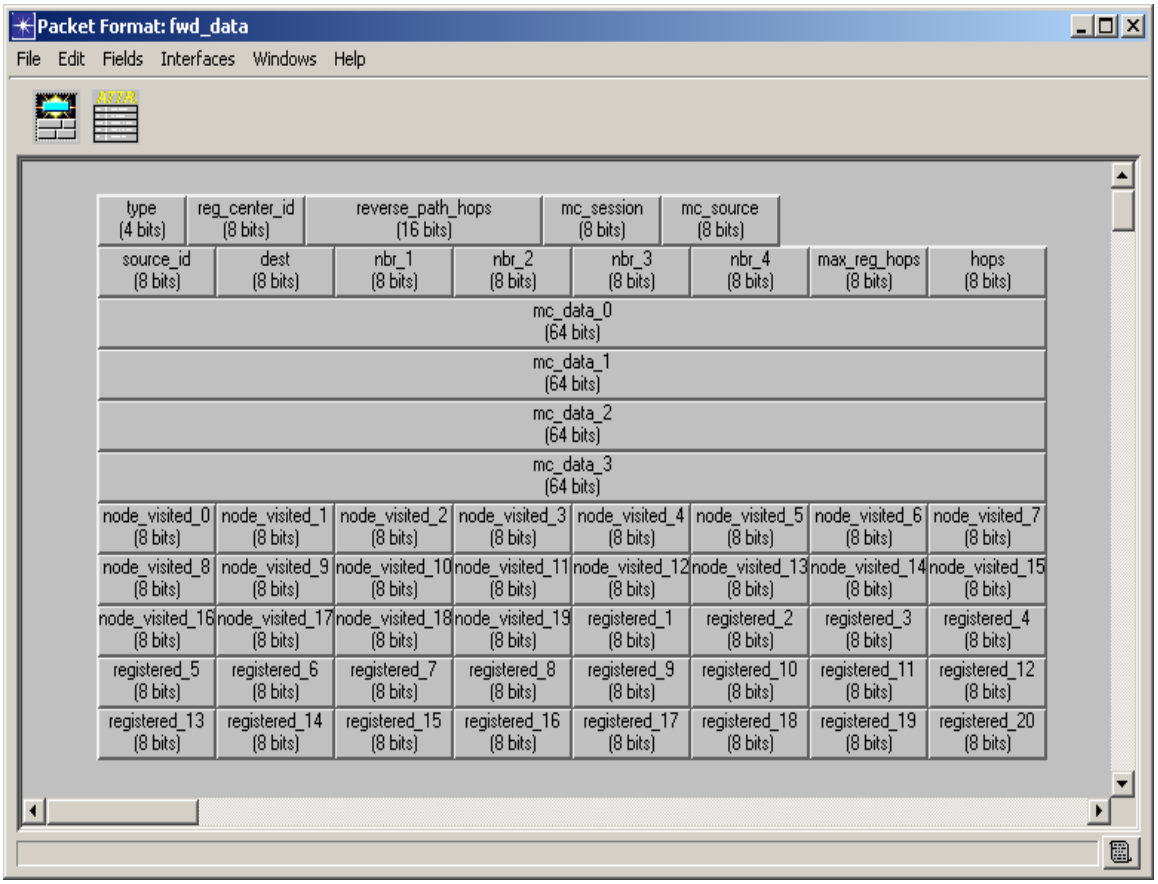


Fig. 4.11: Data Packet

Reverse_path_hops: This field is used to keep the count of hops traveled by backward packets.

Mc_session: This field identifies the multicast session taking place.

Mc_source: Identifies the node generating multicast data.

Source_id: This field identifies the node generating these packets.

Dest: This field is used by backward packets to identify the final destination for the packet.

Nbr_1 through nbr_4: These fields contain the neighbors of a node forwarding this packet.

Mc_data_0 through mc_data_3: user data

Node_visited_0 through node_visited_19: These fields contain the identification of

nodes visited by forward registration packets.

Registered_1 through Registered_20: These fields contain the identification of nodes registered for the multicast session.

The process to distribute data around a Registration Center shown below in Figure 4.12 is the same for all branches.

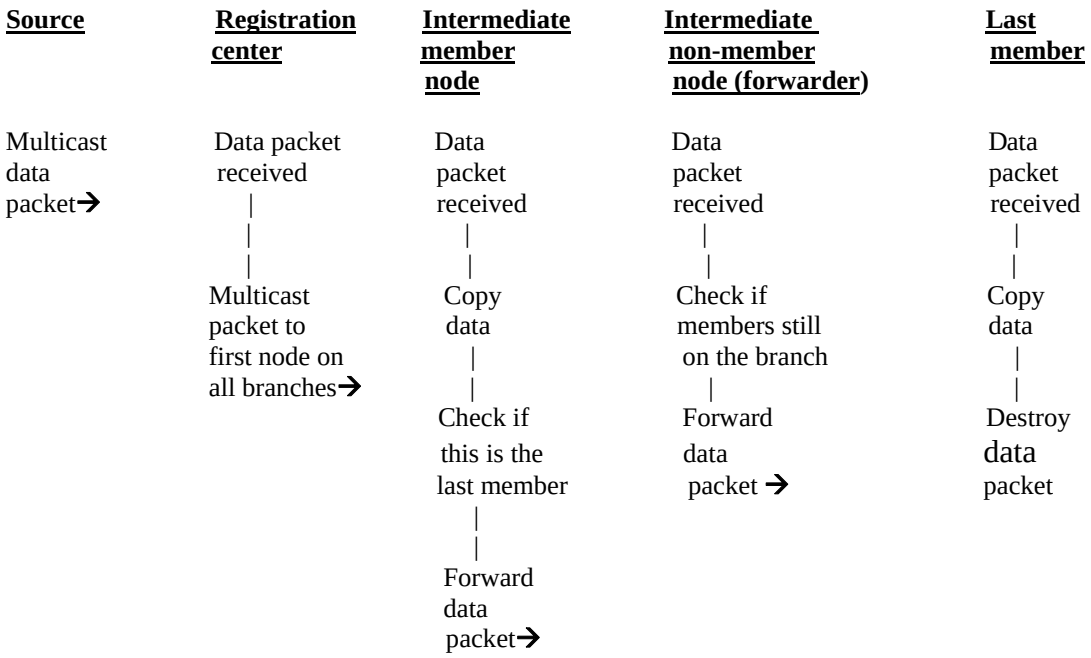


Fig. 4.12: Multicasting Data on a Tree Branch

4.5.9 DATA RECEPTION ACKNOWLEDGEMENT

When a forward data distribution packet arrives at the last node registered for the multicast session, this node generates a backward packet to be sent back to the registration center. This packet follows the path to the registration center.

When a node receives a *back_data* packet, it gets the destination (registration center) node from the packet. If the node's id and the destination id are different, the current

node retrieves the path to the destination from the packet and proceeds to identify the next node in the path. Once this node is determined, the packet is relayed to the stream connecting that node. On the other hand, if the destination node id and current node's id are equal, the current node is the registration center, and the number of neighbors is calculated and compared to the number of packets that has been delivered. The equality of these two numbers indicates that the packet has been sent to all branches and the packet can be sent to the sink. An inequality in the number of neighbors and the number of packets (forward data packets) indicates that some branches still need to receive the data, and therefore the control has to be sent to the *fwd_data* module. This module distributes the data to the registered nodes as delineated in the Data Distribution (fwd data) section above. Figure 4.13 shows a data acknowledgement packet.

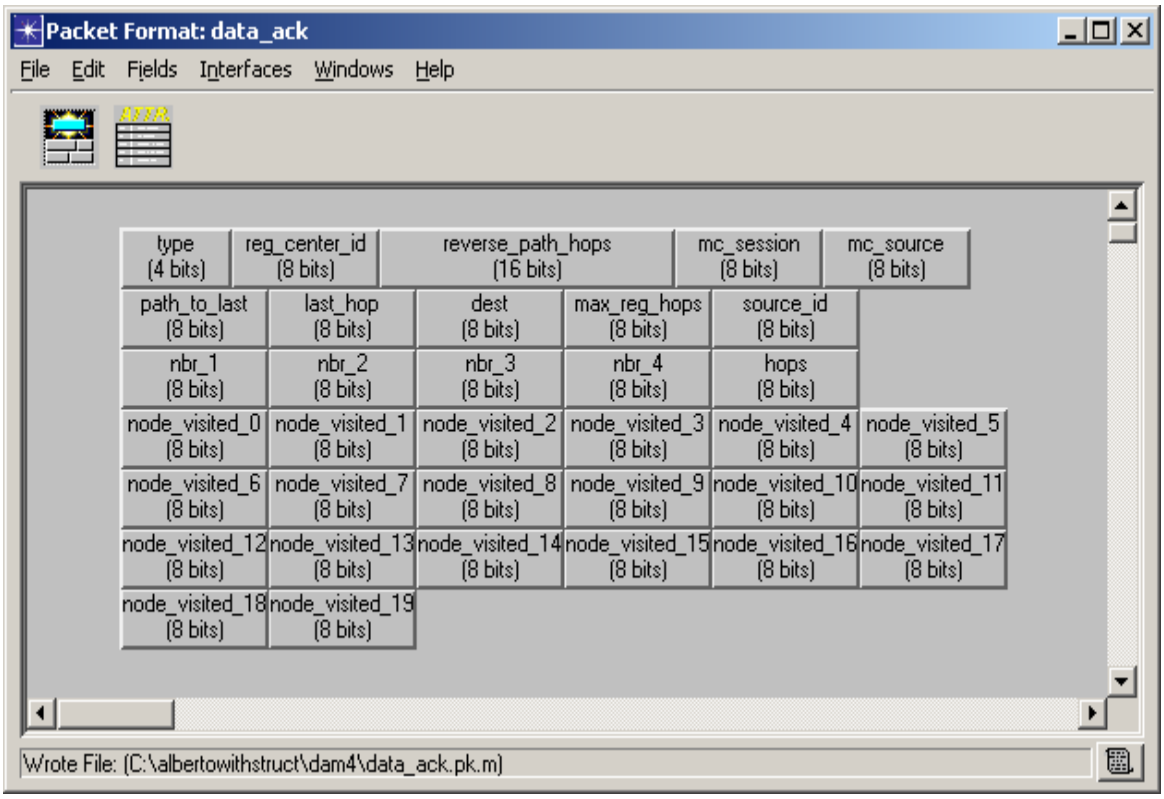


Fig. 4.13: Data acknowledgement packet

4.6 SIMULATION OBSERVATIONS AND RESULTS

We have selected OPNET 11.0 as the platform to implement our simulations and to study the performance and accuracy of the algorithm. The following metrics have been evaluated: End to End Delay, Throughput, and Multicast Delivery Delay.

The maximum size of a network that can be simulated in our projects has a diameter of 40 hops provided that the multicast source is located in the middle of the network, otherwise the diameter must be decreased down to a value of 20 hops depending on the location of the source. Since the Multicast Delivery Delay is our primary concern, we have compared this delay by running both the non-distributed and distributed algorithms on symmetric and non-symmetric networks.

4.6.1 SYMMETRIC PROJECTS

On a symmetric network, the multicast source is located right in the middle and the topology as well as the number of nodes is identical on each side of the source. Figure 4.14 shows a symmetrical network with 39 nodes using node 20 as the multicasting source. This network has been used to study the multicast delivery delay on typical non-distributed and on our distributed multicast algorithm. Typical multicast algorithms distribute data from node 20 to nodes 19 and 21; these nodes relay the data which goes hoping from node to node until it reaches the last nodes 1, and 39. Our distributed algorithm, using discovery and registration packets creates a multicast distribution tree which is shown in Figure 4.15. In the middle of each branch of the distribution tree a

node has been selected as a registration and data distribution center and data is distributed on each branch from these centers.

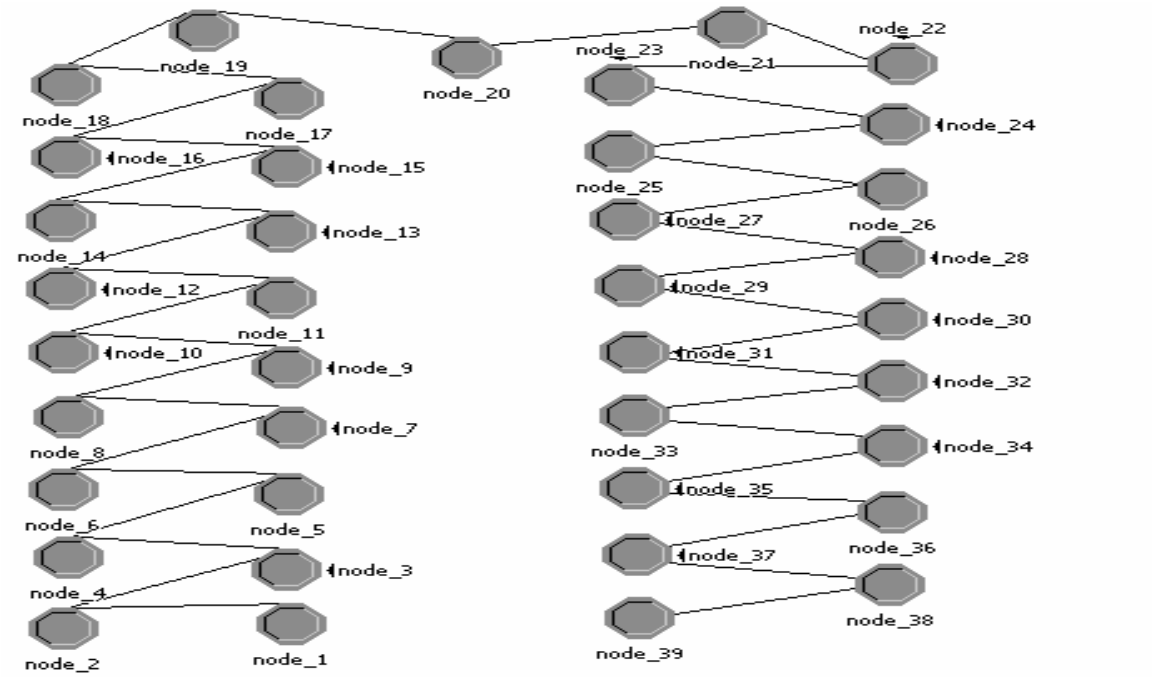


Fig. 4.14: Symmetric network of 39 nodes with the source located at node 20

Clearly, if data distribution centers are located exactly in the middle of each branch, and the network is symmetric with respect to the multicast source, the multicast delivery delay time in a non-distributed algorithm will be double compared to that in the distributed multicast algorithm as shown in Figure 4.16.

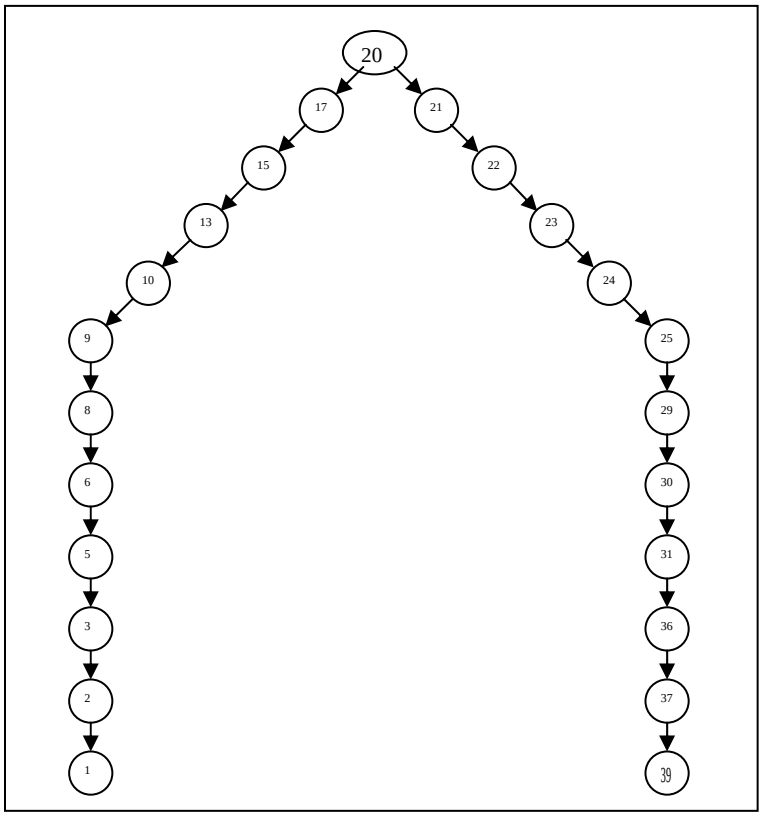


Fig. 4.15: Formed tree of 22 nodes from a symmetric network with 39 nodes

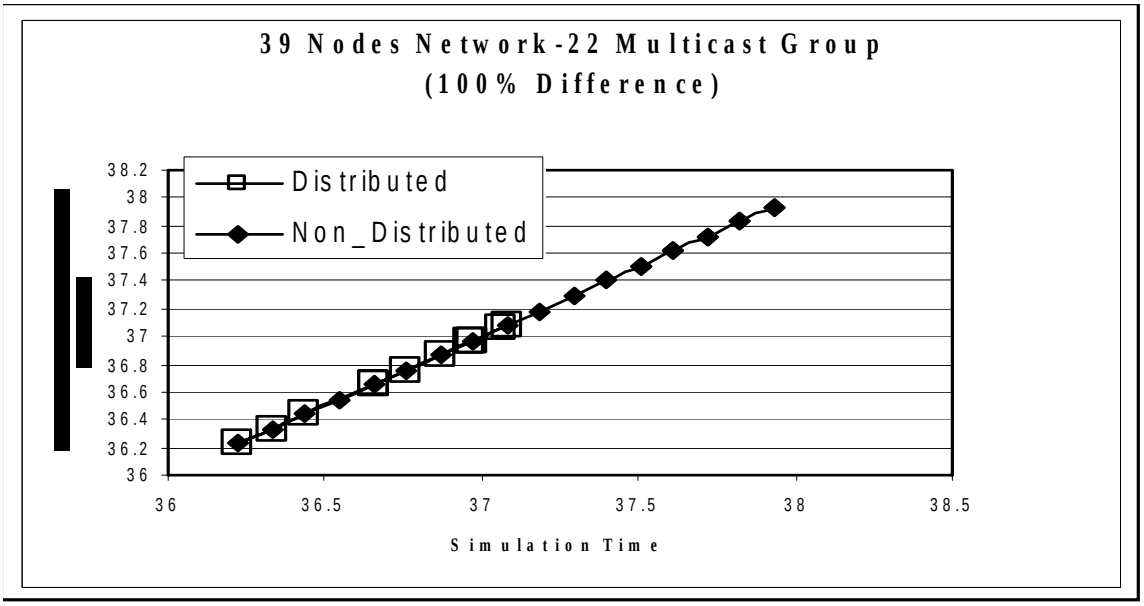


Fig. 4.16: Multicast Delivery Delay for Distributed and Non-Distributed Algorithms in Symmetrical Networks.

4.6.2 NON-SYMMETRIC PROJECTS

On a non-symmetric network, the multicast source is located anywhere in the network and the branches around the source are all different in topology as well as in the number of nodes. Figure 4.17 shows a non-symmetric network of 25 nodes with the multicast source located at node 3.

This network has been used to study the multicast delivery delay on typical non-distributed and on our distributed multicast algorithm. Typical multicast algorithms distribute data from node 3 to nodes 13 and 6; node 13 relays the data to node 7 and node 6 multicasts data to nodes 4, 8, and 9. This process of relaying and multicasting go on until nodes 18, 11, 25, 23, 19, 17, 21, and 14 obtained the data.

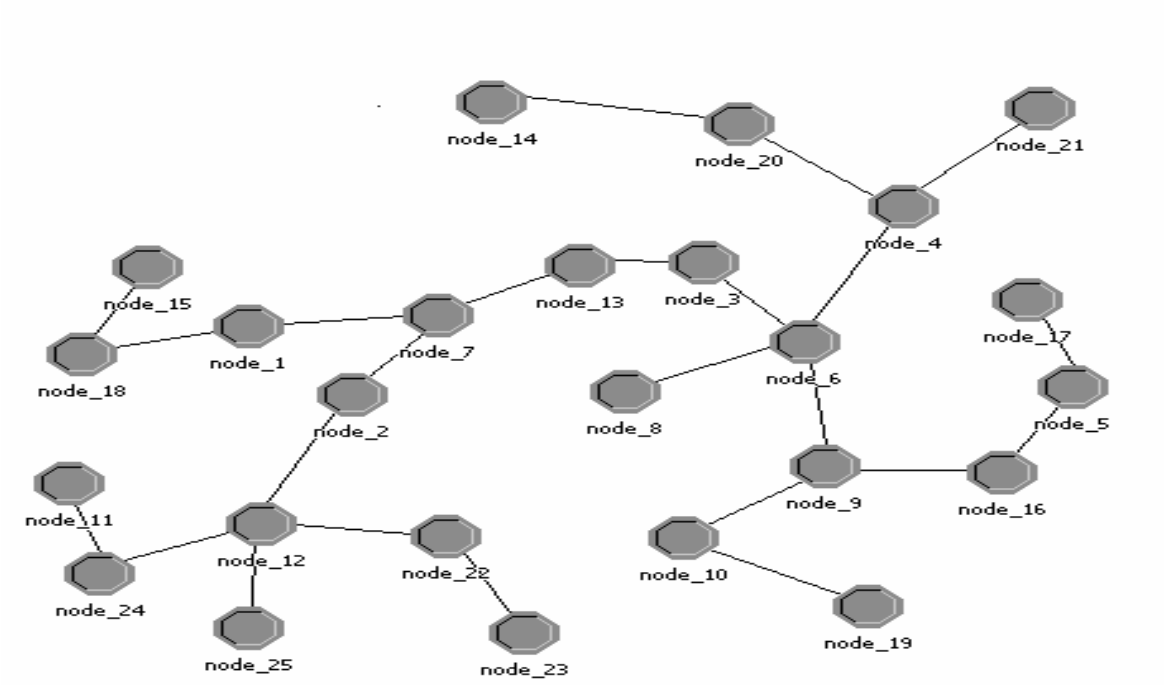


Fig. 4.17: Non-Symmetric network of 25 nodes with the source located at node 3

Our distributed algorithm, using discovery and registration packets creates a multicast distribution tree of 11 nodes which is shown in Figure 4.18. In the middle of each branch

of the distribution tree a node has been selected as a registration and data distribution center and data is distributed on each branch from these centers.

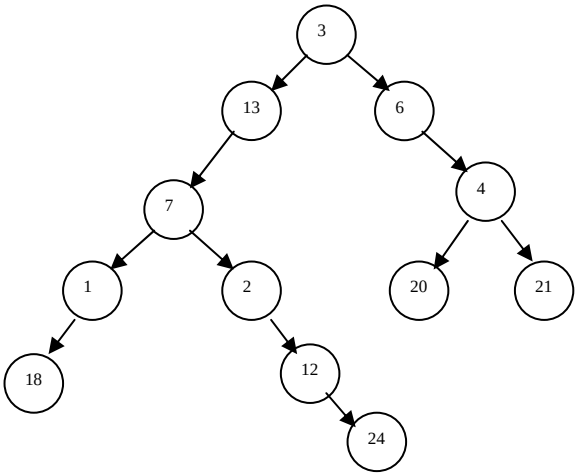


Fig. 4.18: Formed tree of 11 nodes from a non-symmetric network with 25 nodes

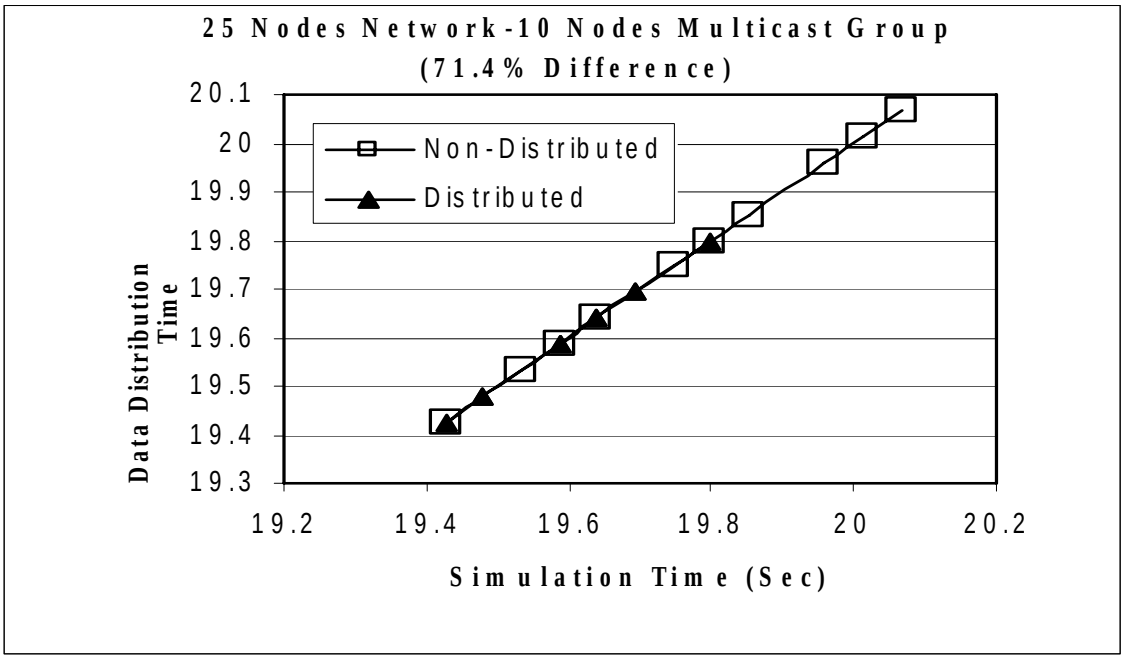


Fig. 4.19: Multicast Delivery Delay for Distributed and Non-Distributed Algorithms in Non-Symmetrical Networks.

Figure 4.19 shows the multicast delivery delay for both algorithms when run on a non-symmetric network of 25 nodes with the multicast source located at node 3. As expected,

the delay in non-distributed is no longer 100% higher than that in the distributed network, but still is considerably higher at 71.4%.

We have tested the distributed multicast delivery delay in a non-symmetric network with multiple branches and multicast source at different locations. The obtained results are presented in the sequence of graphs from Figure 4.20 to Figure 4.23, and a comprehensive graph to compare the distributed multicast delivery delay against that in the non-distributed in that type of networks is presented in Figure 4.24. Results in Figure 4.21 are slightly different from those presented in Figure 4.3. The difference is due to the fact that these registration centers are not in locations equally distributed in the network, and as a result delays among registrations centers are not equal as we claim in the development of equation (4.3).

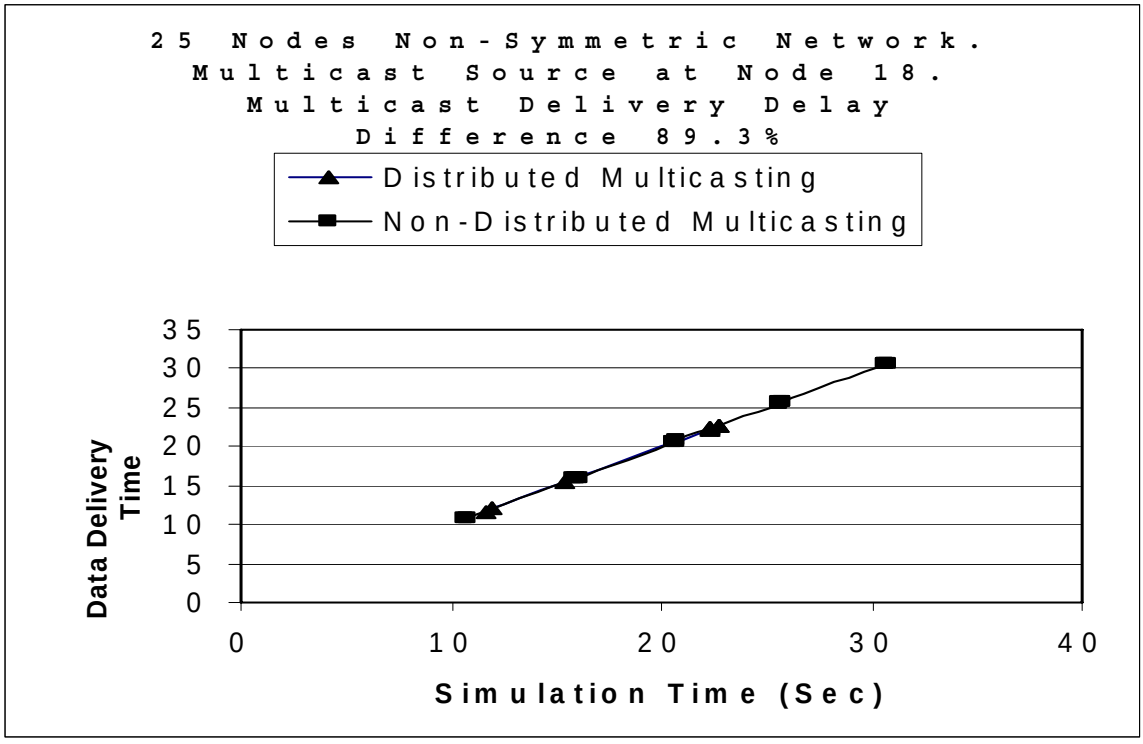


Fig. 4.20: Multicast Delivery Delay comparison with source at Node 18.

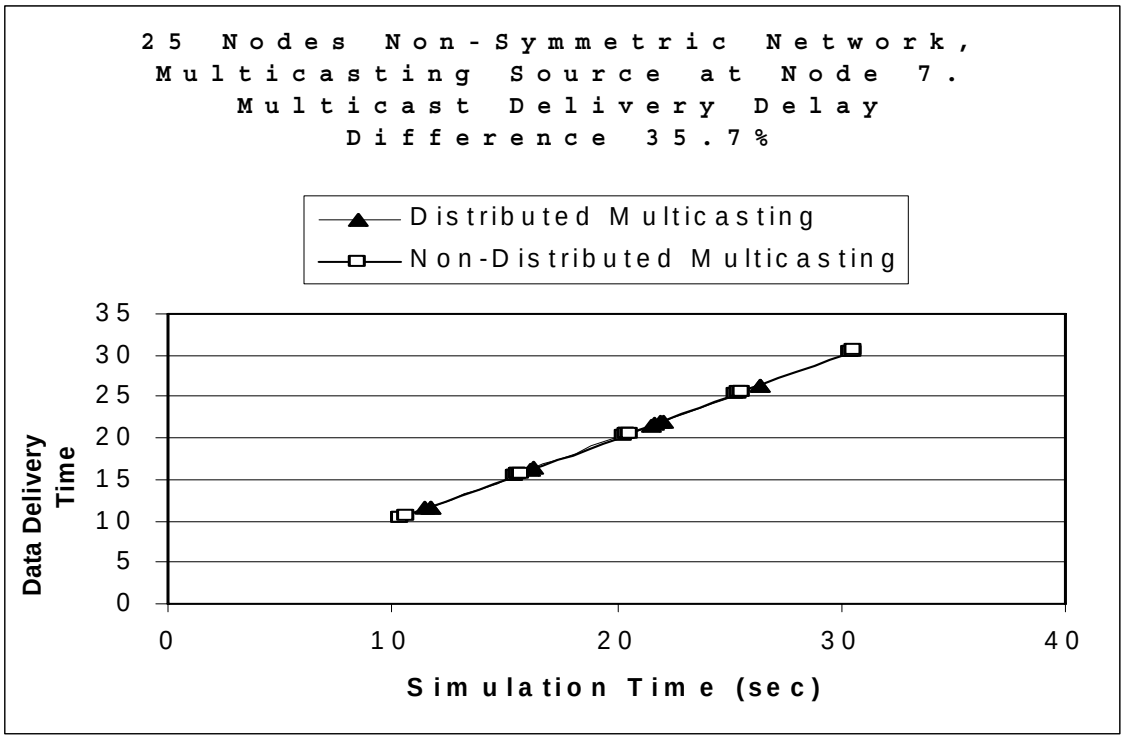


Fig. 4.21: Multicast Delivery Delay comparison with source at Node 7.

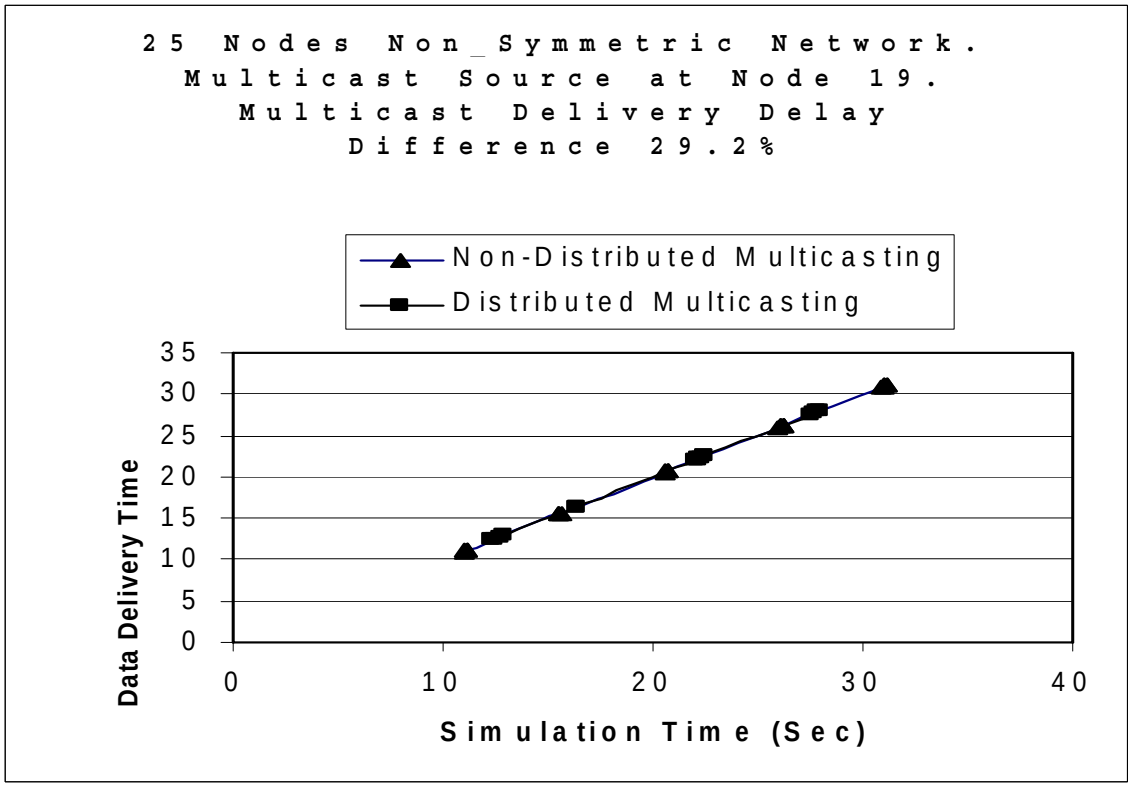


Fig. 4.22: Multicast Delivery Delay comparison with source at Node 19.

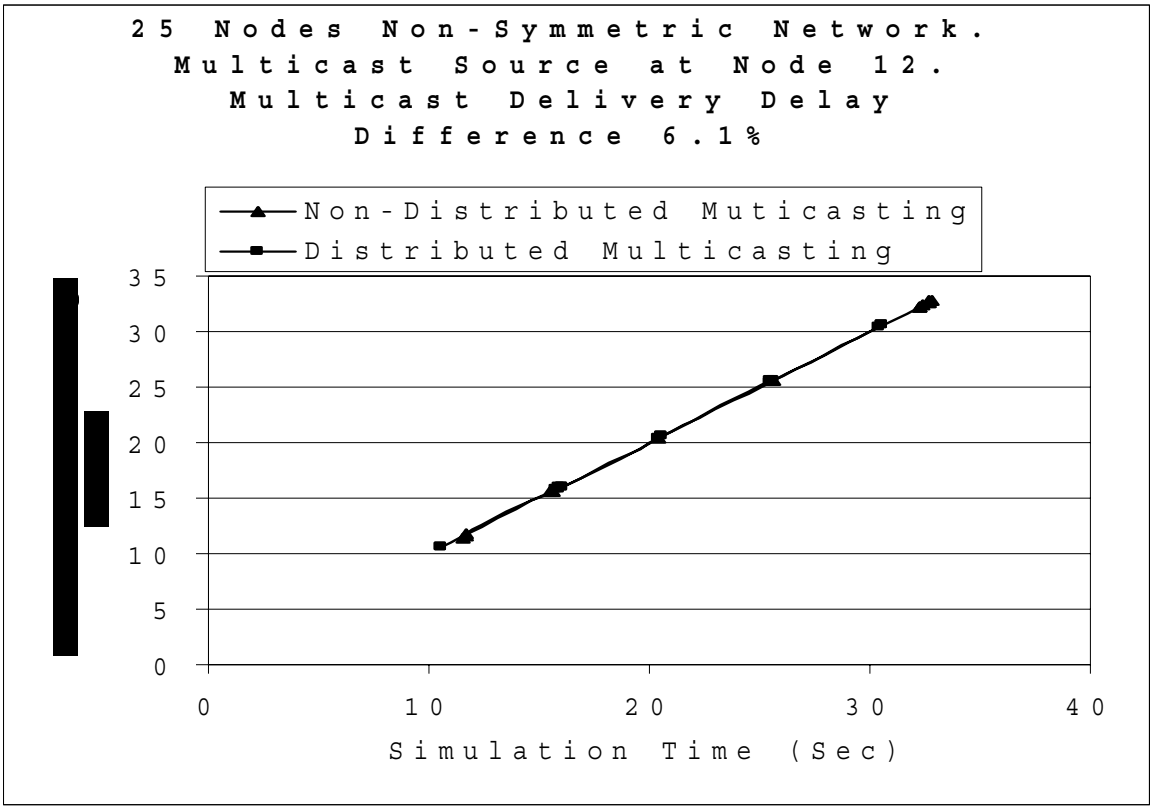


Fig. 4.23: Multicast Delivery Delay comparison with source at Node 12.

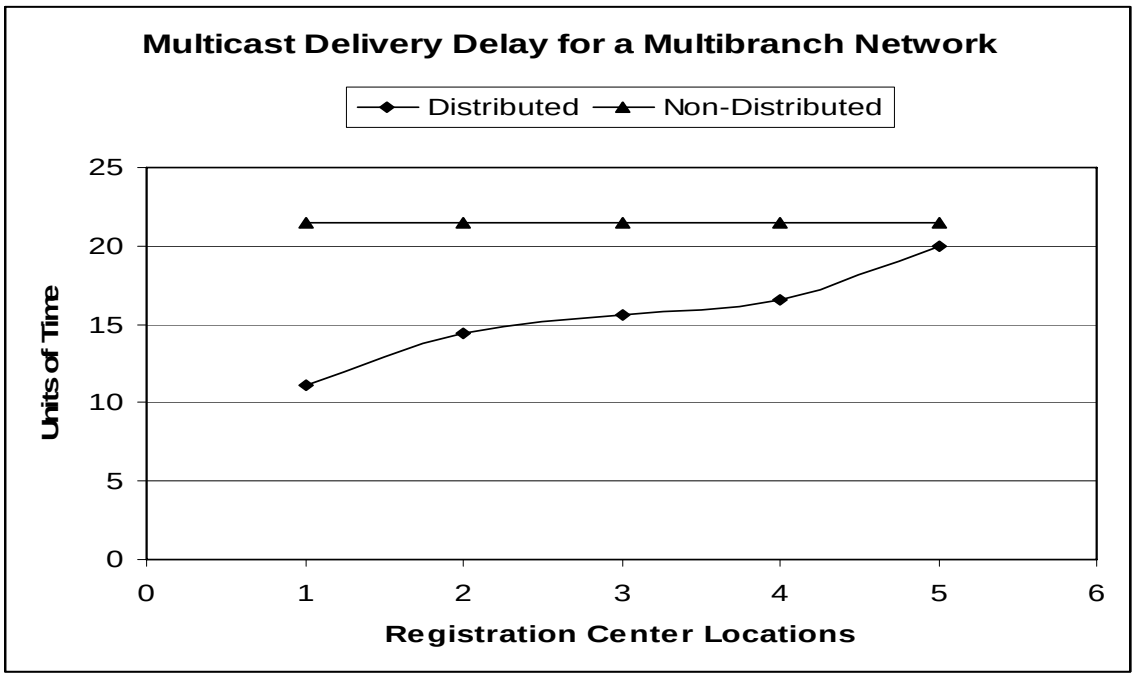


Fig. 4.24: Multicast Delivery Delay for a Non-Symmetric Multi-branch Network

A model has been implemented for testing the reliability of the distributed multicasting algorithm and the non-symmetric network shown in Figure 4.16 has been used to test its functionality. Simulations on that network have been run for 20 seconds and the obtained results are presented in Figure 4.25 through Figure 4.29. The sequence of graphs have been obtained from running simulations as a random number that generates group members is changed from 10 to 1. These graphs display the number of visited nodes, the number of multicast members, the time when data packets are received, and the numbers of registration centers. This model also calculates the number of data packets that should be received based on the number of registration centers and the number of nodes registered for the multicast session at each registration center. It then calculates the Packet Delivery Ratio which is the ratio of the number of data packets actually delivered to the group members versus the number of data packets supposed to be received. Even though we are presenting the results from a non-symmetric network with only 25 nodes to test the algorithm, it has also been tested on non-symmetric networks of 45 nodes, and grid networks with 96 nodes. The sequence from Figure 4.25 through Figure 4.29 shows the Packet Delivery Ratio as the “thruput”. It should be noted that this ratio remains constant for all values of multicast members.

The number of bits per second to be delivered to all multicast members has been varied with the objective of obtaining a maximum packet size. Simulations show that a good packet size is around 512 bits for all members of the multicast session to receive data

packets. This packet delivery ratio has been tested in a network with 30 nodes and up to 20 multicast members. Experimental results are shown in Figure 4.30.

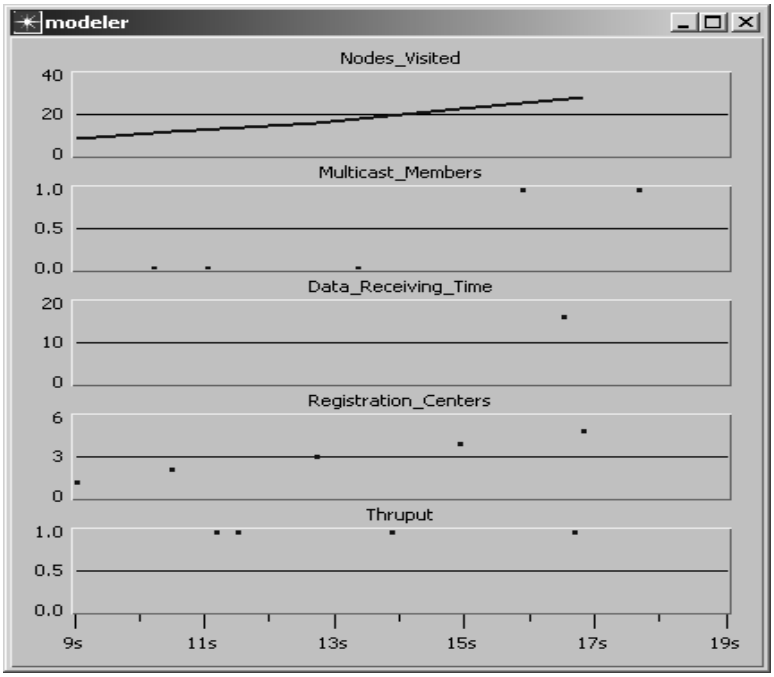


Fig. 4.25: 30 Nodes Non-Symmetric with one multicast member

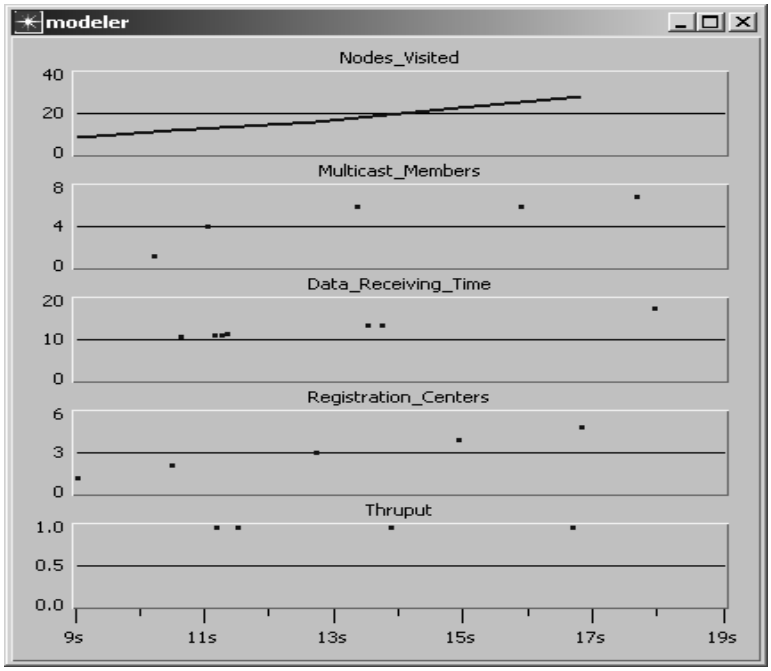


Fig. 4.26: 30 Nodes Non-Symmetric with seven multicast members

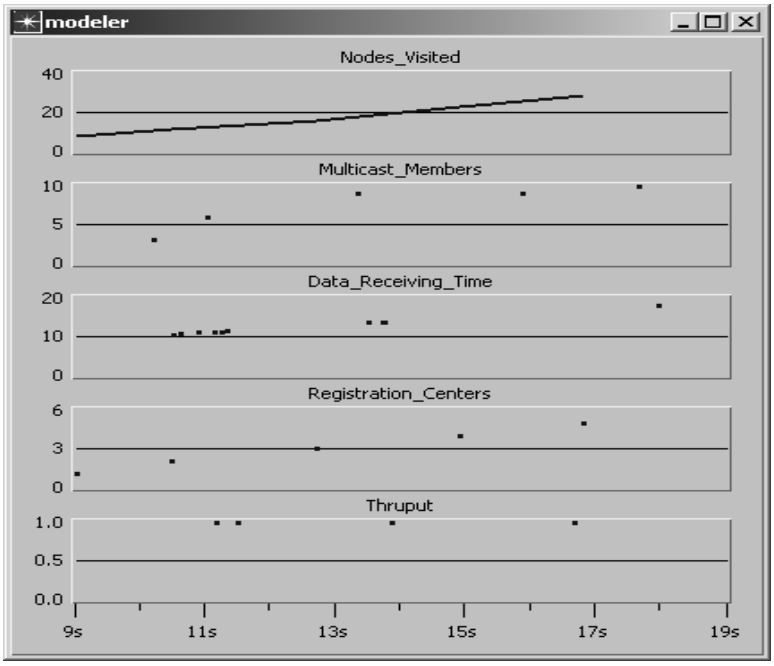


Fig. 4.27: 30 Nodes Non-Symmetric with ten multicast members

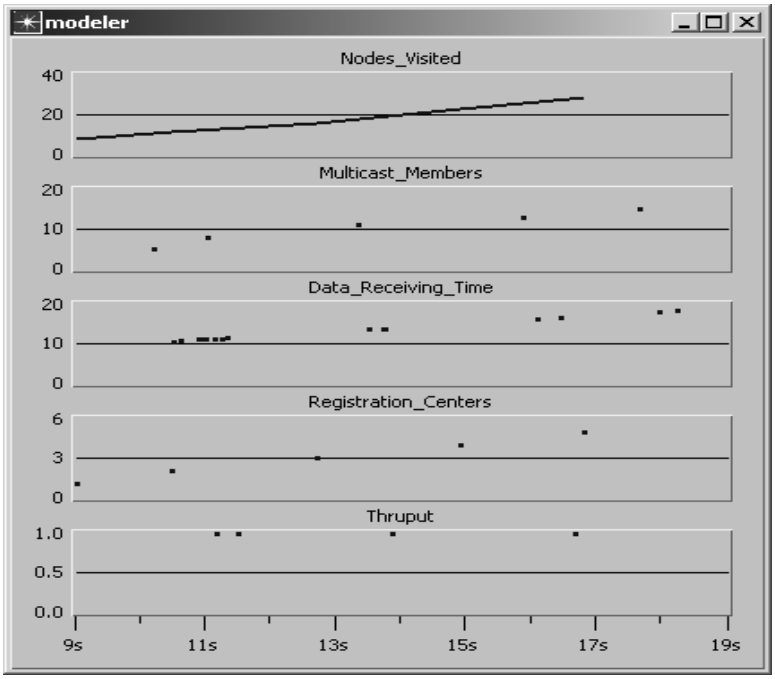


Fig. 4.28: 30 Nodes Non-Symmetric with 14 multicast members

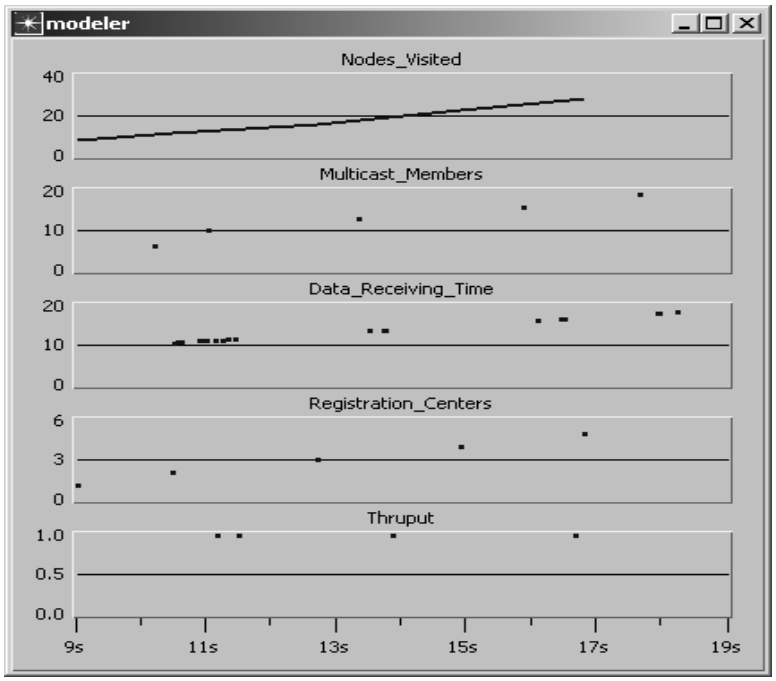


Fig. 4.29: 30 Nodes Non-Symmetric with 19 multicast members

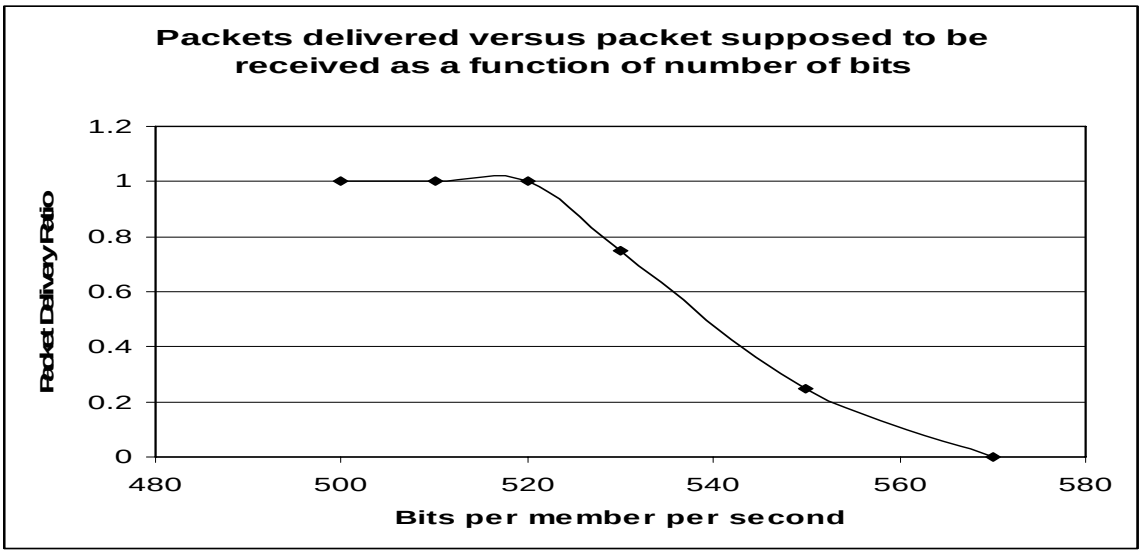


Fig. 4.30: Packet Delivery Ratio as a function of bits per member per second

5. MAINTENANCE MECHANISMS

In order to maintain the performance of the algorithm throughout the multicast session, some control and performance mechanisms are implemented at different stages of the process. In general, these mechanisms do the following: avoid loops, improve network coverage, maintain connectivity, recover from lost connections, and recover missing data packets. The remainder of this section discusses such mechanisms in further detail.

5.1 OUTWARD PROPAGATION

Outward propagation is another important maintenance mechanism. It is utilized in order to minimize looping. In the current algorithm, outward propagation will occur as all nodes update all discovery packets with information about their neighbors. This information will be used by the visited node in the following manner: if a sender's neighbor is also a receiver's neighbor, it is assumed that that node also received the packet; therefore, that node will be discarded from the list of nodes to be used as prospective destinations. By providing neighbor information to the receivers, discovery packets will move away from the source until they travel for the specified number of hops, or until they reach a node with no neighbors. Furthermore, neighbor nodes exchange information about their neighbors. If this information (i.e., neighbors of neighbors) is included in the discovery packets, their trajectory will be more directive since no packet can be forwarded to a neighbor of the sender's neighbors. Under this technique, it is expected that loops may not occur.

The process for outward propagation has been completely explained in section 5.

5.2 DISTRIBUTION TREE RECOVERY

The distribution tree is maintained by message exchange between parent and child on each branch. If those nodes are communicating through registration packets or data packets there will be no need for tree maintenance messages. The algorithm is provided with a timer to monitor the connection time. If a Parent or Child node disconnects from a branch, the timer will expire and seeking packets will be released from the child node to reconnect the tree.

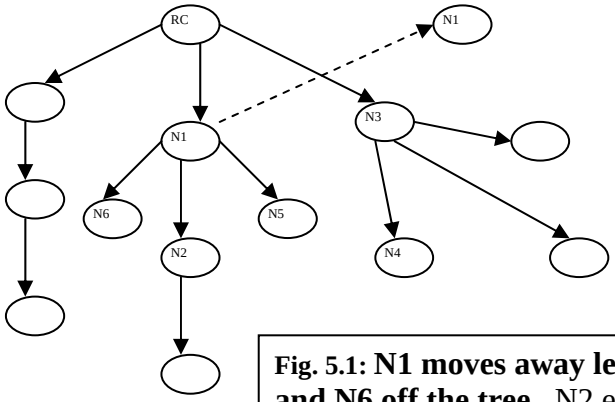


Fig. 5.1: N1 moves away leaving N2 and its neighbors N5, and N6 off the tree. N2 emits seeking packets searching for an RC. Neighbors of N1 are used to start the search up the tree. N3 has registration information and responds forming a new branch. N1 emits seeking agents too searching for a reconnection.

Figure 5.1 shows the process that takes place when a node is disconnected from the tree. Node 1 has moved away from the original location, causing Node 2 to release seeking packets through the neighbors of Node 1. The proximity between Node 5 on the

broken branch and Node 4 on the well connected branch allows the reconnection of the tree shown in Figure 5.2.

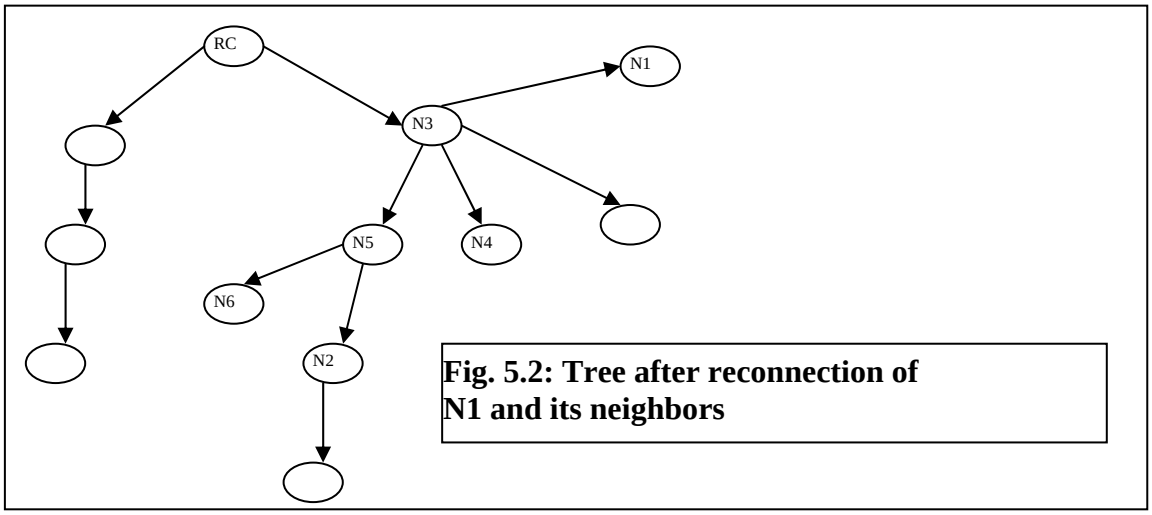


Fig. 5.2: Tree after reconnection of N1 and its neighbors

5.3 DATA RECOVERY

All nodes in a branch of the tree need to know if the node above them is a member of the multicast session or if it is a forwarder. This information is provided by the forward packets as they travel down from registration centers. The importance of this information lies in the fact that data can only be recovered from members of the session since they process and store data packets while forwarders do not.

A node will detect missing data from sequence numbers specified in the data packets. When a node detects a missing packet, a seeking packet will be released. The immediate destination for this new packet will be any of the parent's neighbors, and the final destination will be the registration center of that particular branch. The first contacted member that has the missing data will respond. In the event that a registration center disconnects, the immediate child will seek another center to reconnect to. From information provided by the source, registration centers are able to maintain current information about each other. In this way, they can share that information with the

immediate child on each branch. Registration centers' information is also exchanged with the neighbor exchange packets. Figure 5.3 shows a node starting the process to recover a packet.

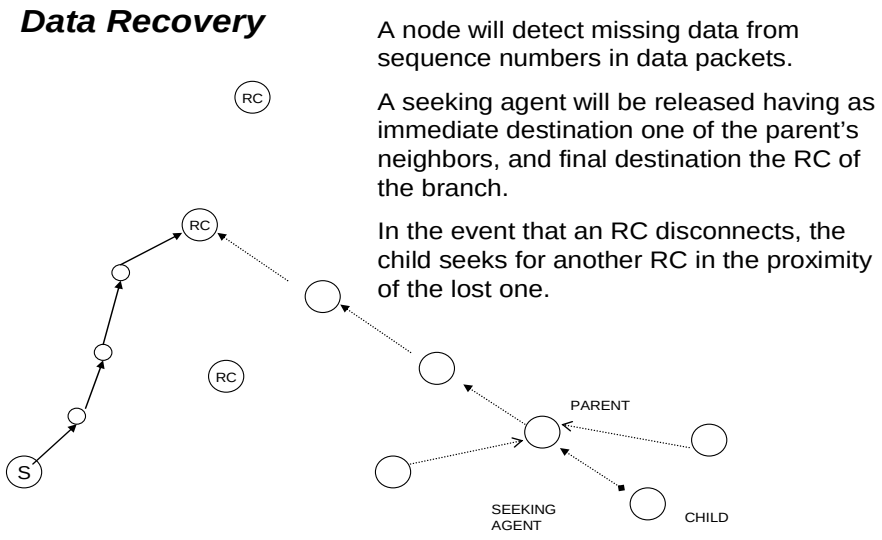


Fig. 5.3: Nodes can recover missing packets from nodes in the path to a registration center

5.4 THE PRUNNING PROCESS

The pruning process, i.e., removing a node from the distribution tree when the node is no longer interested in receiving data from a multicast session, is implemented in the following manner. A node wishing to leave the multicast session sends a pruning packet to its forwarder in the distribution tree. If the node receiving the pruning packet is in the distribution tree, but is not a member of the multicast session, it will send the pruning packet to its parent node in the tree. This process continues until a member of the session is reached. When that happens, all nodes in the path from the node leaving are removed from the distribution tree. If the node receiving the pruning packet is a member

of the multicast session, it will stop sending data to the leaving node, and a message will be sent to the registration center of the module. The center, in turn, will inform the multicasting source. Figure 5.4 shows node I leaving the multicast group.

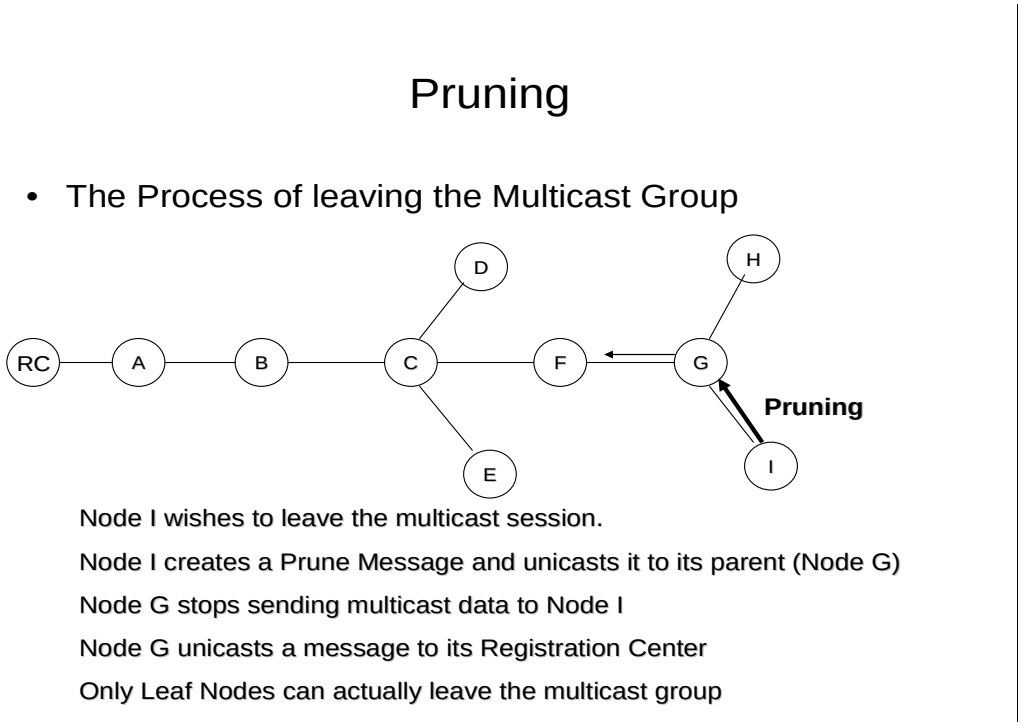


Fig. 5.4: Nodes can leave the multicast session at any time

6. PROCESSES AND PROJECTS DESCRIPTION

We have implemented our simulations to study the performance and accuracy of the algorithms in OPNET version 11.0. The nodes, links, processes, projects, and simulations will be described in this section.

6.1 THE NODE

The node presented in Figure 6.1 has the following components: a router with the process embedded into it, four transmitters to send packets up to four neighbors connected to it, four receivers to receive packets from up to four neighbors connected to it, a source to generate packets for neighbors' discovery, a source to generate packets for neighbors' exchange, a source to generate discovery and advertising packets, a source to generate data packets, and a sink to destroy unnecessary packets.

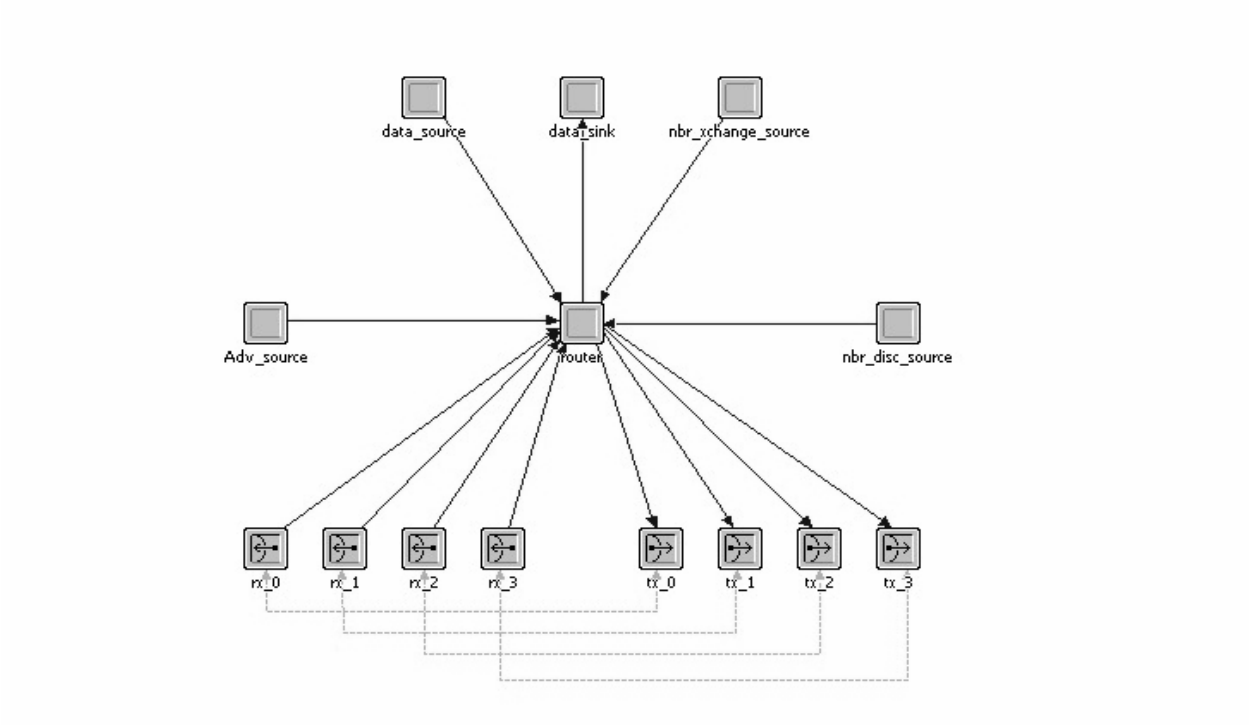


Fig. 6.1: Node model used in the projects

Every node used in the simulations has the following components:

Four Transmitters (RX_0, RX_1, RX_2, RX_3)

Four Receivers (TX_0, TX_1, TX_2, TX_3)

A Processor (router)

A source to generate packets used for neighbor discovery (nbr_disc_source)

A source to generate packets for neighbor exchange information (nbr_xchange_source)

A source to generate advertisement packets (Adv_source)

A source to generate packets that communicate the source with the center's modules (mcast_source)

A source to generate registration packets (reg_source)

A source to generate seeking packets

A source to generate data packets

A sink to collect destroyed packets

6.2 PROCESSES

We have implemented several processes for testing several stages of the algorithms. We present here only three of those processes: 1) neighbor discovery and exchange, and network contour discovery (advertisement), 2) neighbor discovery and exchange, and network contour discovery plus the registration process, and 3) neighbor discovery and exchange, network contour discovery, registration, plus the data distribution process.

6.2.1 NEIGHBOR DISCOVERY, NEIGHBOR EXCHANGE, AND NETWORK CONTOUR DISCOVERY PROCESSES

As its name indicates, this process is used by every node to discover nodes in its neighborhood. After a convergence time when all nodes have created a list of neighbors, they proceed to exchange that list among all neighbors. Neighbor discovery and exchange are necessary to maintain packets propagating in an outward direction during the network contour discovery process.

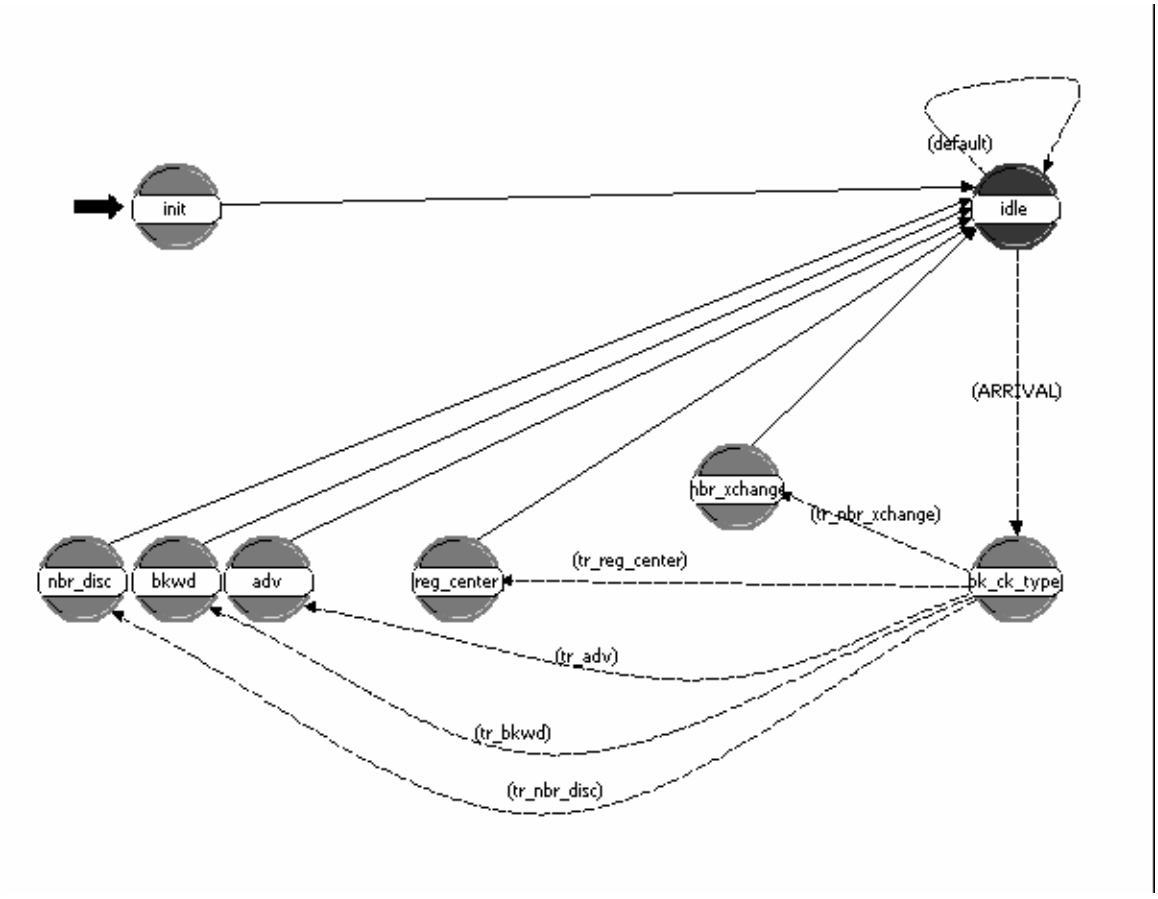


Fig. 6.2: Neighbor Discovery, Neighbor Exchange, and Network Contour Discovery Process

The Process attached to a node as shown in Figure 6.2 contains the following states:

Pk_ck_type: This state checks the packet type of all arriving packets and determines the next transition for the system.

Nbr_disc: This state has the instructions to request and collect neighborhood information.

Nbr_xchange: This state has the instructions to exchange and collect neighborhood information.

Adv: This state has the instructions to control the flow of forward advertisements packets.

Bkwd: This state has the instructions to control the flow of backward advertisements packets.

Mcast: This state has the instructions to control the communication between the multicast source and the centers of the modules.

Reg: This state has the instructions to control the flow of registration packets on each branch of the tree

Once the exchange process have converged (all nodes have exchanged their neighbors), the discovery of the contour of the network can be initialized by transferring control of the process to the adv (discovery packets propagation) module. When discovery packets have traveled the maximum number of hops specified in the packet, or when they are instructed to return by the control mechanism being used (unity, binary, or square), or when there are no neighbors to forward the discover packet to, the control of the process is transferred to the bkwd (backward discovery packets propagation) module. Four different packet types are used in this part of the process: neighbor discovery packets (type 1), neighbor exchange packets (type 2), network contour forward discovery packets (type 3), and network contour discovery backward packets (type 4). The module

chk_pk_type (check packet type) checks the packet type and transfers control of the process to the appropriate module. Once a module reaches the end (last instruction in the embedded code), the control is transferred to the idle module to wait for the transition from the chk_pk_type module. A complete description of neighbor discovery appears in section 5.4.1, a complete description of neighbor exchange is presented in section 5.4.2, a complete description of the forward network contour discovery module is presented in section 6.4.1, and a complete description of the backward network contour discovery module is presented in section 6.4.2.

6.2.2 NEIGHBOR DISCOVERY, NEIGHBOR EXCHANGE, NETWORK CONTOUR DISCOVERY, AND REGISTRATION PROCESSES

The Neighbor Discovery, Neighbor Exchange, and Network Contour Discovery processes appear in the previous section. This section adds the registration process as shown in Figure 6.3. When a backward discovery packet returns to the source, this source calculates the location of a well-located node to be used as a registration center. Once this node is selected, the control of the process is transferred to the reg_center (registration center) module. This module controls the propagation of the packets until they reach the reg center. When the registration center is reached, the control of the process is transferred to the reg_center_adv (registration center advertisement).

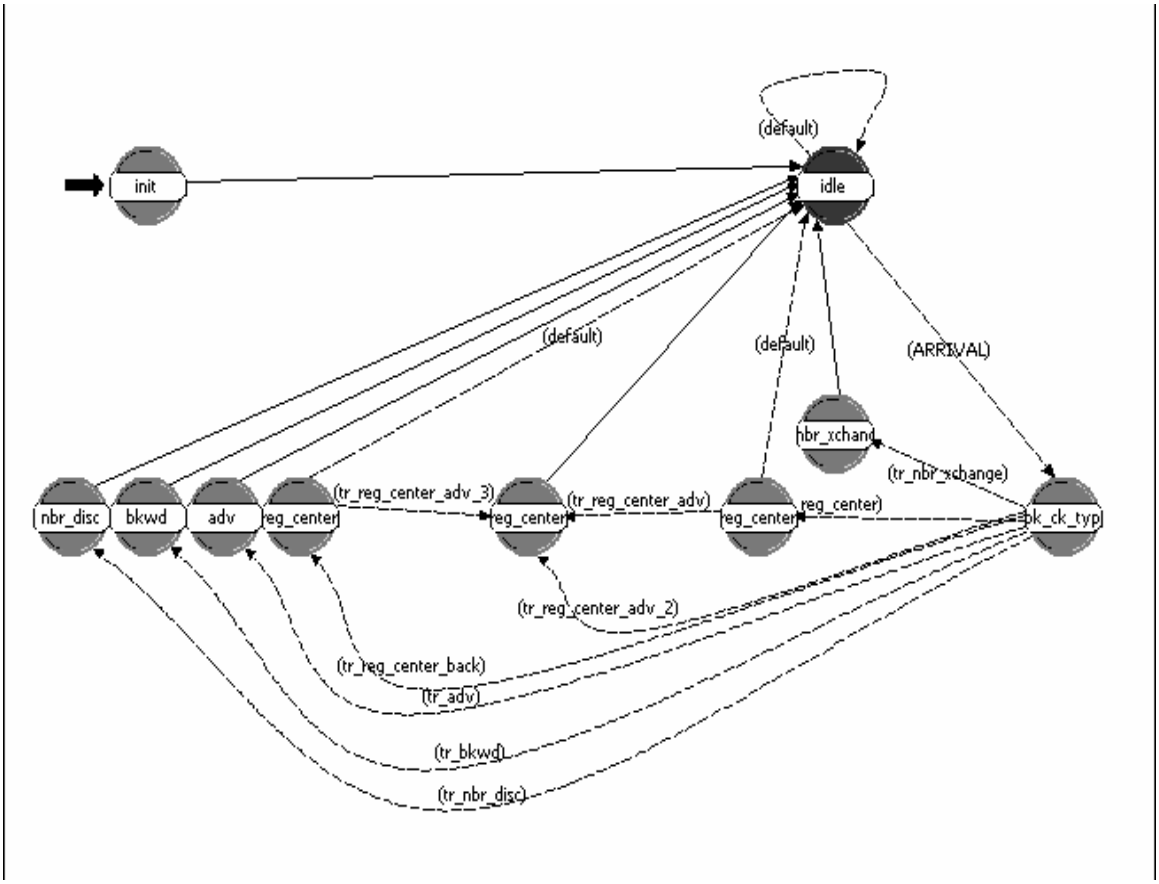


Fig. 6.3: Neighbor Discovery, Neighbor Exchange, Network Contour Discovery, and Registration Process

This module controls the forward propagation of registration packets until these packets reach a node with no neighbors, or until a packet travels the maximum number of hops specified in the packet. When either of these two situations check positive, the control of the process is transferred to the reg_center_bkwd (registration center backward) module. This module controls the propagation of the backward registration packets until they reach the registration center. When these backward registration packets reach the registration center, this node collects registration information from them, and then creates a packet to be sent to the multicast source. The control of the process is then transferred to the reg_mc (registration center to multicast source) module. This module controls the

propagation of the packets containing registration for the multicast session until they reach the multicasting source. A complete description of the reg_center (registration center) module appears in section 6.4.3, a complete description of the propagation of registration advertisement packets module is presented in section 6.4.4, a complete description of the propagation of backward registration packets module appears in section 6.4.5, and a complete description of the registration center to multicast source module is presented in section 6.4.6. Four different packet types are added in this part of the process: reg center packets (type 5), forward registration advertisement (type 6), backward registration advertisement packets (type 7), and registration to multicast source packets (type 8).

As described in the previous section, the module chk_pk_type (check packet type) checks the packet type and transfers control of the process to the appropriate module according to the packet type arriving into the process.

6.2.3 NEIGHBOR DISCOVERY, NEIGHBOR EXCHANGE, NETWORK CONTOUR DISCOVERY, REGISTRATION, AND DATA DISTRIBUTION PROCESSES

The Neighbor Discovery, Neighbor Exchange, and Network Contour Discovery processes appear in section 8.2.1, and the Registration process is presented in the previous section. This section adds the data distribution process as shown in Figure 6.4. When a packet from a registration center arrives at the multicast source, this source reads the information about nodes registered for the multicast session. If there are nodes registered, the source inserts user data into a packet and the control of the process is transferred to the mc_reg (multicast to registration center) module.

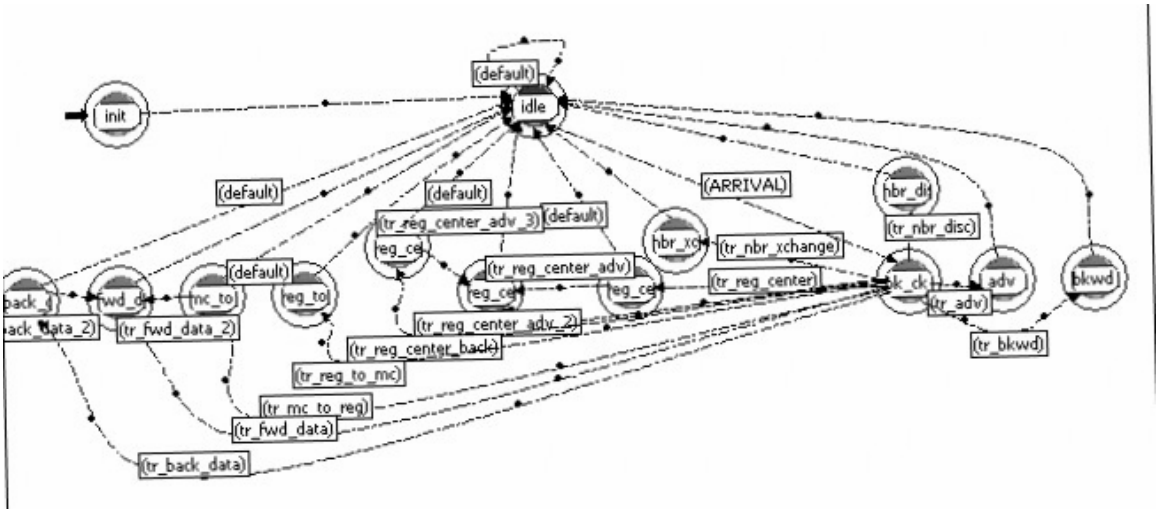


Fig. 6.4: Neighbor Discovery, Neighbor Exchange, Network Contour Discovery, Registration, and Data Distribution Process

This module controls the propagation of this packet until it reaches the registration center. When the packet reaches the registration center, the control of the process is transferred to the fwd_data (forward data) module. This module controls the distribution of the user data to all nodes that have registered for the multicast session. When the packet reaches the last registered node in any direction, the control of the process is transferred to the bkwd_data (backward data) module. This module controls the propagation of a backward packet until it reaches the registration center which originated the propagation of the data packet. The process is complete when all branches connected to the registration center acknowledge receipt of all data user packets. Detailed descriptions of various processes appear as follows: the mc_reg (multicast source to reg center) module is described in section 6.4.7; the fwd_data (forward data) module is presented in section 6.4.8, and the bkwd_data (backward data) module is described in section 6.4.9. Three different packet types are added in this part of the process: multicast to reg center packets (type 9), forward data distribution (type 10), and backward data distribution packets (type 11).

Once more, as cited in the previous two sections, the module chk_pk_type (check packet type) checks the packet type and transfers control of the process to the appropriate module according to the packet type arriving into the process.

6.3 PROJECTS

6.3.1 STRING TOPOLOGY PROJECTS

Using string topology, the algorithm performs as expected, forward packets effectively propagate outward from the source. Then, when the maximum number of hops is reached, the current node generates a backward packet and sends it back to the source. The backward packet propagates along the reverse path of the forward packet, with its destination set as the original source. The source detects the arrival of a backward packet and selects a node in the middle of the path as the new center for a module. Figure 6.5 shows a network with 10 nodes connected in a string topology, and Figure 6.6 shows a network with 19 nodes. The maximum number of nodes used in our simulations was 100 nodes for this topology type.

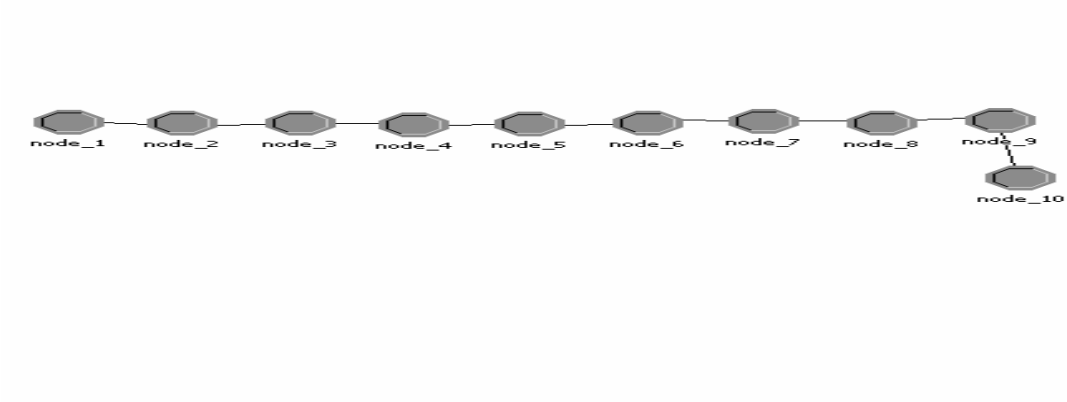


Fig. 6.5: Network with 10 nodes connected in string topology

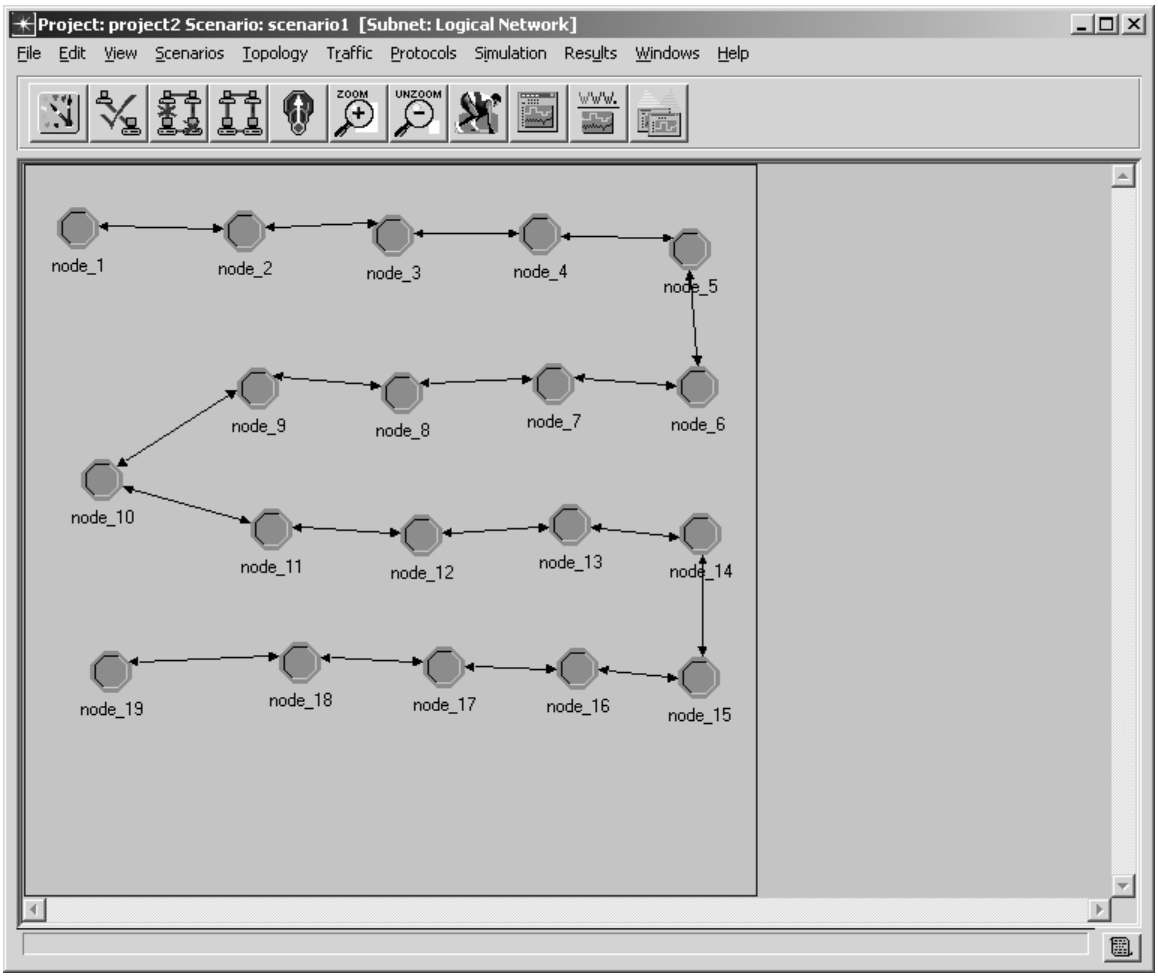


Fig. 6.6: Network with 19 nodes connected in string topology

These topologies have been used to observe the following:

1. Neighbor Discovery
2. Neighbor Information Exchange
3. Creation and Propagation of Forward Advertisement Packets
4. Creation and Propagation of Backward Advertisement Packets.
5. Time and number of packets to advertise all nodes in the network.

6.3.2 STAR TOPOLOGY PROJECTS

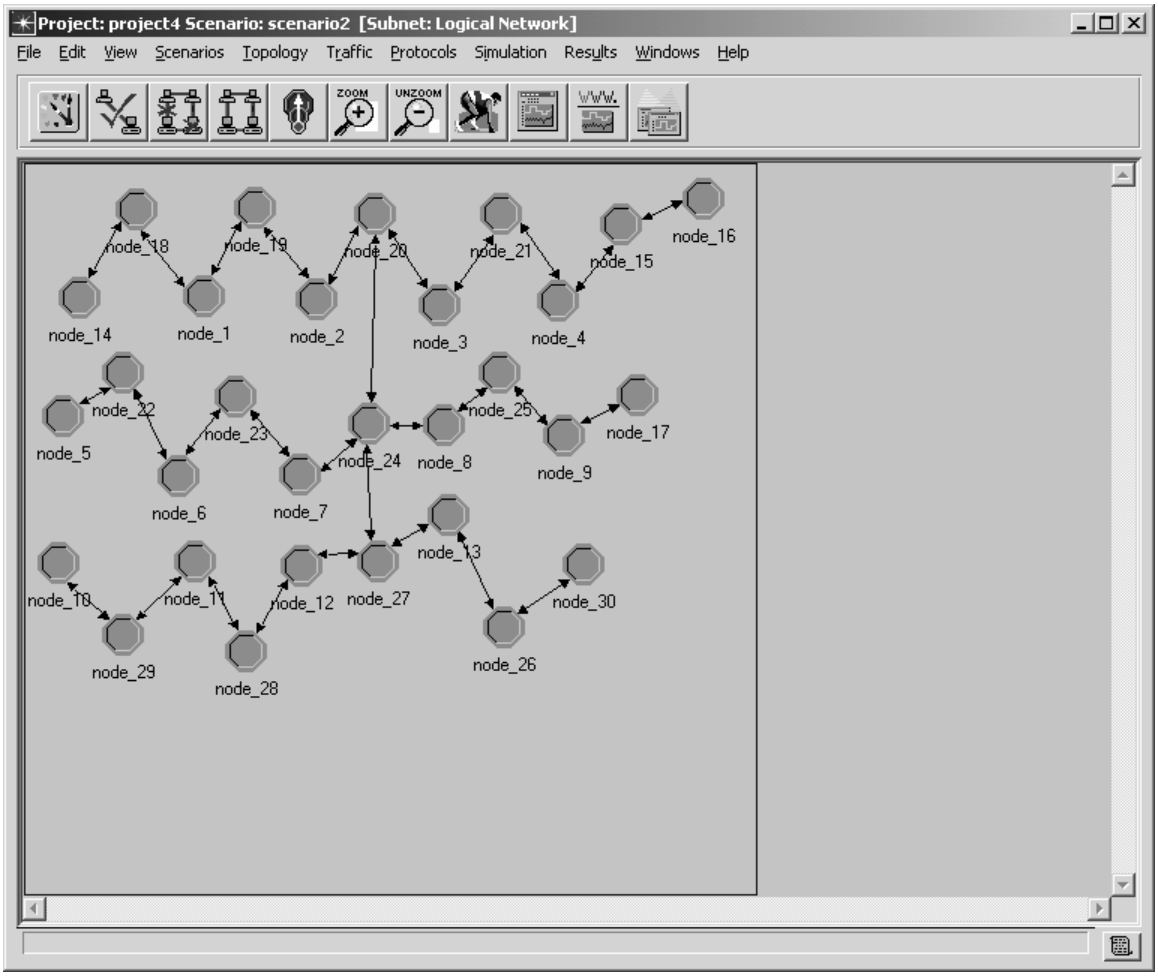


Fig. 6.7: Network with 30 nodes connected in a star topology

Using the star topology, the algorithm performs as expected. Forward packets are deployed sequentially through every source's neighbor, and these packets travel outward from the source. When the number of hops reaches its maximum or when the current node's neighbors are the same as the sender's neighbor, a backward packet is generated and sent back to the source. When the backward packet reaches the source, an indication

of the length in terms of number of nodes of that branch has been obtained. When all backward packets return to the source, the source obtains an indication about the size and shape of the network. Figure 6.7 shows a network with 30 nodes connected in a symmetric star topology, and Figure 6.8 shows a network with 25 nodes connected in a non-symmetric star topology. The maximum number of nodes used in our simulations was 45 nodes for this topology type.

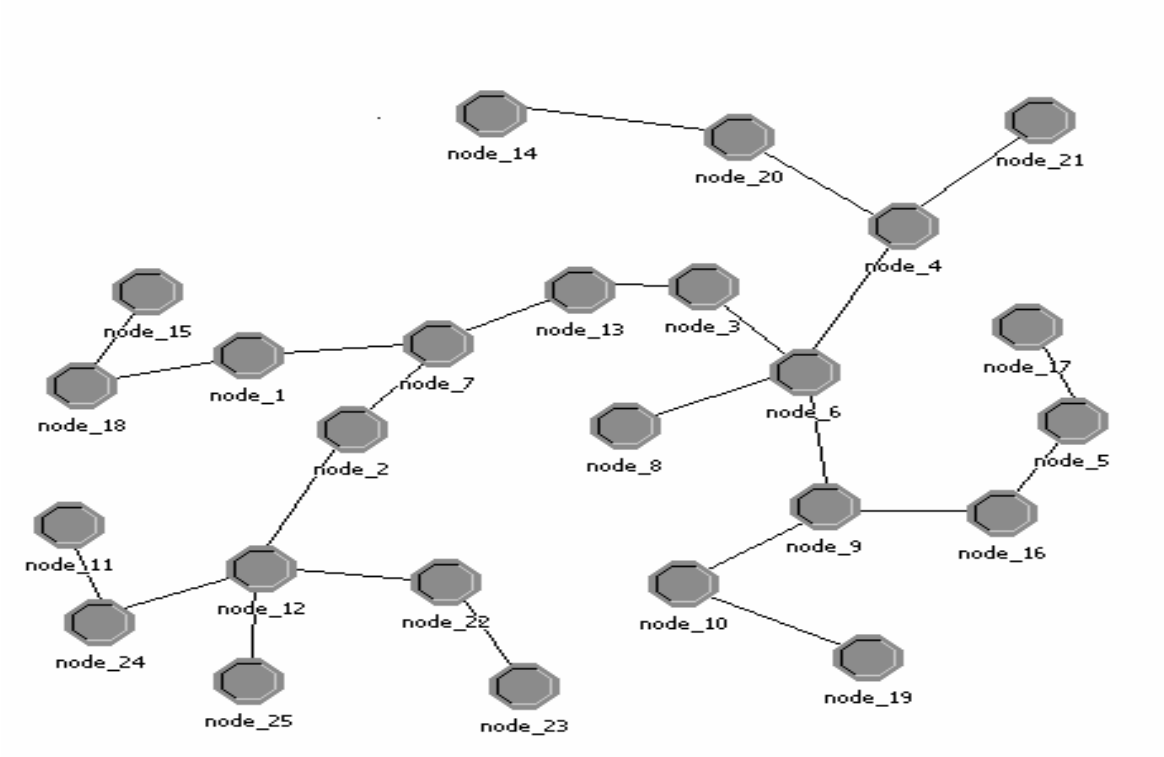


Fig. 6.8: Node with 25 nodes connected in a star topology

These topologies have been used to observe the following:

1. Time required by all nodes to discover their neighbors.
2. Time required by all nodes to exchange neighbor information.
3. Propagation of forward and backward packets.
4. Number of nodes visited by each forward packet.
5. Redundancy of nodes visited.

6. Number of forward packets required to visit all nodes in the network.

7. Multicast delivery delay.

6.3.3 GRID TOPOLOGY PROJECTS

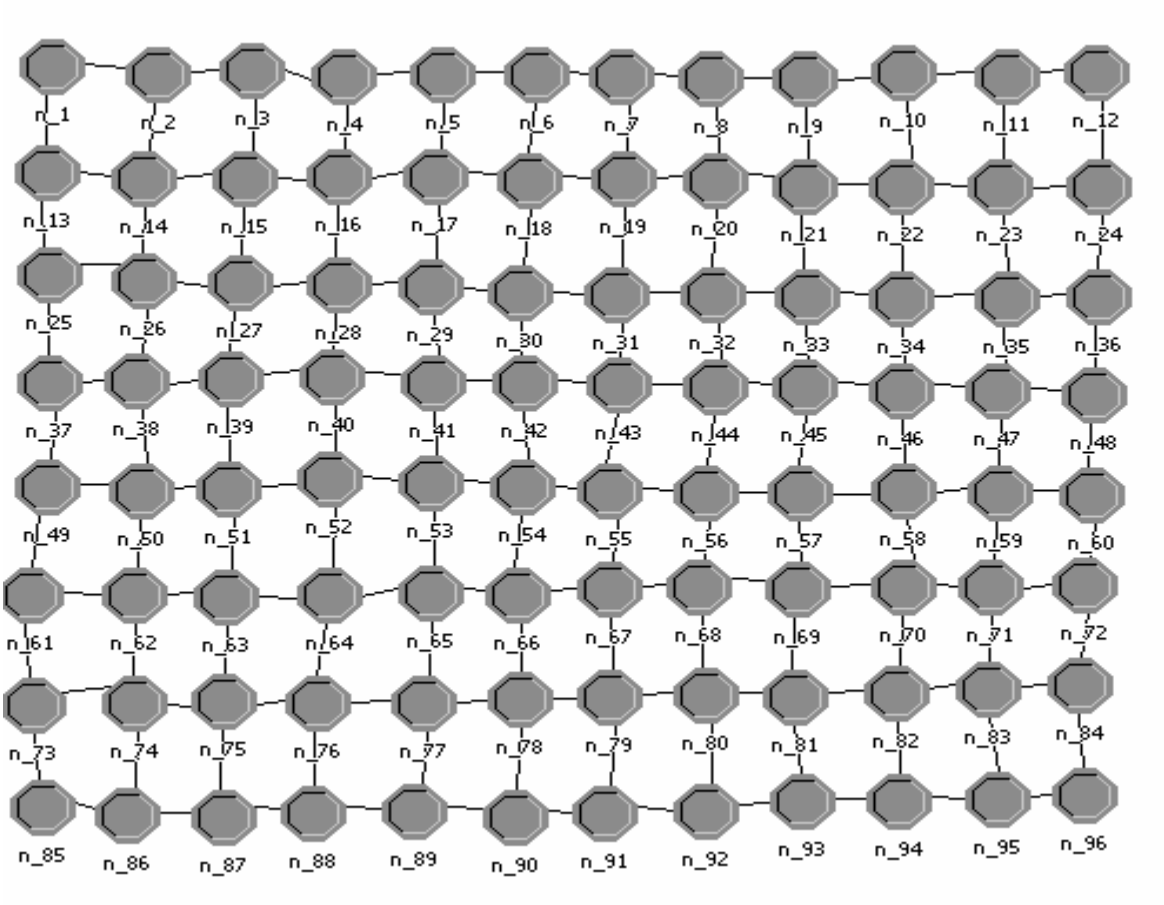


Fig. 6.9: Network with 96 nodes connected in a grid topology

Using the grid topology, the algorithm performs as expected. Loops are reduced when the sender's neighbors are provided to the forward packets; loops are minimized when the sender's neighbors' neighbors are provided to the forward packets.

The coverage of this network shows no linearity with time. This is caused by redundancy as some nodes are visited more than once by forward packets. This

redundancy is necessary so that nodes that are located far away from the source are visited.

This topology as shown in Figure 6.9 has been used to observe the following:

1. Time required by all nodes to discover their neighbors.
2. Time required by all nodes to exchange neighbor information.
3. Outward Propagation of forward packets.
4. Creation and propagation of backward packets.
5. Number of visited nodes by each forward packet.
6. Number of forward packets required to visit all nodes in the network.
7. Redundancy of visited nodes
8. Multicast delivery delay

7. SUMMARY AND FUTURE WORK

We have created an algorithm that guarantees outward propagation of packets when discovering the contour of a network. This type of propagation is required to minimize the possibility of creating inward paths or even worse, the creation of loops.

The size and contour of the network is discovered by a very well controlled deployment of forward and backward packets. The connectivity between any two nodes is maintained by a periodic exchange of neighbor messages.

Complete neighbor information is exchanged periodically. This information helps to keep the path of those packets in an outward direction from their sources; neither sender's neighbors nor neighbors of the sender's neighbors are selected as destinations at forwarder nodes.

The network distribution discovered by our algorithm can be effectively used to create modularity for the distributed multicasting algorithm.

We are also proposing an algorithm that has all the components required for multicasting. These components are: Network Topology Discovery and Advertisement, Registration, Data Distribution, as well as required maintenance mechanisms.

The network is divided into regions to apply modularity mainly for the following two reasons: first to avoid the acknowledgement implosion at the source, and second to minimize the multicast delivery delay, that is, the time between the beginning of the reception of a packet by the first group member and the end of the reception of the same packet by the last member.

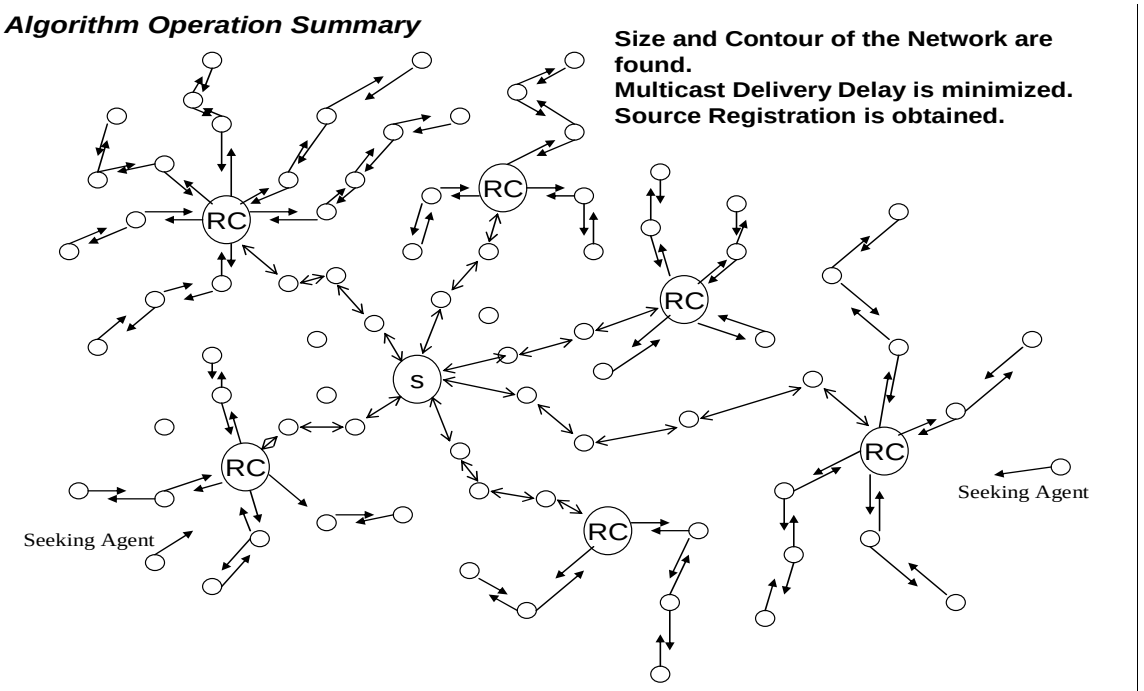


Fig. 7.1: A source and several registration centers showing the modularity in the network

Figure 7.1 shows a network where modularity has been implemented. A multicast source (S) and several registration centers (RC) are shown.

Future work will include the research to improve the following: (1) data rate while maintaining the high performance on the packet delivery ratio, (2) size of the multicasting group without compromising the packet delivery ratio. The algorithms proposed in this work have most of the require elements for wireless mobile networks, therefore we will study its applicability in these type of networks.

9. APPENDIXES

APPENDIX A: NEIGHBOR DISCOVERY MODULE

```
op_sim_message("enter_nbr_disc","");
                                op_pk_nfd_get(pk_ptr,"rep_flag",&pk_f);
                                if (op_intrpt_strm()==nbr_source)
                                        {
                                        if(pk_f==0)
                                                {
                                                op_pk_nfd_set(pk_ptr,"source_id",node_id);

        op_pk_nfd_get(pk_ptr,"source_id",&pk_source_id);
                                                for (i=0;i<=3;i++)
                                                        {
                                                        pk_ptr_cp=op_pk_copy(pk_ptr);
                                                        op_pk_send(pk_ptr_cp,i);
                                                        }
                                                op_pk_destroy(pk_ptr);
        }
                                        else
                                                {
op_sim_end("error","neighbor disc from the same source ","and flag not zero","");
                                                }
                                        }
                                else
                                        switch (pk_f)
                                        {
                                        case 0:
                                        op_pk_nfd_get(pk_ptr,"source_id",&pk_source_id);
                                        op_pk_nfd_set(pk_ptr,"source_id",node_id);
                                        op_pk_nfd_set(pk_ptr,"nbr_id",node_id);
                                        op_pk_nfd_get(pk_ptr,"nbr_id",&pk_node_id);
                                        op_pk_nfd_set(pk_ptr,"rep_flag",1);
                                        op_pk_send(pk_ptr,op_intrpt_strm());
                                        break;
                                        case 1:
                                                        switch (op_intrpt_strm())
                                                                {
                                                                case 0:
        op_pk_nfd_get(pk_ptr,"nbr_id",&my_nbr_id[0]);
        nbr_strm[0]=op_intrpt_strm();
                                                                break;
                                                                case 1:
        op_pk_nfd_get(pk_ptr,"nbr_id",&my_nbr_id[1]);
```

```
nbr_strm[1]=op_intrpt_strm();
                                                    break;
                                                    case 2:
op_pk_nfd_get(pk_ptr,"nbr_id",&my_nbr_id[2]);
nbr_strm[2]=op_intrpt_strm();
                                                    break;
                                                    case 3:
op_pk_nfd_get(pk_ptr,"nbr_id",&my_nbr_id[3]);
nbr_strm[3]=op_intrpt_strm();
                                                    break;
default:op_sim_end("error","","","");
                                                    }
                                        }
                                op_sim_message("exit_nbr_disc","");
```

APPENDIX B: NEIGHBORING EXCHANGE MODULE

```
op_sim_message("entering nbr_exchange", "");
                                        op_pk_nfd_get(pk_ptr,"rep_flag",&pk_f);

                                    if (op_intrpt_strm()==nbr_xchange_source)
                                                {
                                                 if(pk_f==0)
                                                                {

op_pk_nfd_set(pk_ptr,"nbr_sender",node_id);
op_pk_nfd_set(pk_ptr,"my_nbr_1",my_nbr_id[0]);
op_pk_nfd_set(pk_ptr,"my_nbr_2",my_nbr_id[1]);
op_pk_nfd_set(pk_ptr,"my_nbr_3",my_nbr_id[2]);
op_pk_nfd_set(pk_ptr,"my_nbr_4",my_nbr_id[3]);
                                                                 for (i=0;i<=3;i++)
                                                                  {
pk_ptr_cp=op_pk_copy(pk_ptr);
op_pk_send(pk_ptr_cp,i);
                                                                  }
                                                                op_pk_destroy(pk_ptr);
                                                                }
                                                                else
                                                                        {
        op_sim_end("error","nbr_xchange_packet from the same source ","and flag not
zero","");
                                                                        }
                                                                }
                                                        else switch (op_intrpt_strm())
                                                                {
                                                                case 0:
                                                                nbr_strm[0]=op_intrpt_strm();
                                                op_pk_nfd_get(pk_ptr,"nbr_sender", &pk_nbr_sender_0);

op_pk_nfd_get(pk_ptr,"my_nbr_1",&my_nbr_nbr_id_1[0]);
op_pk_nfd_get(pk_ptr,"my_nbr_2",&my_nbr_nbr_id_1[1]);
op_pk_nfd_get(pk_ptr,"my_nbr_3",&my_nbr_nbr_id_1[2]);
op_pk_nfd_get(pk_ptr,"my_nbr_4",&my_nbr_nbr_id_1[3]);
                                                                break;
                                                                        case 1:

        nbr_strm[1]=op_intrpt_strm();
        op_pk_nfd_get(pk_ptr,"nbr_sender", &pk_nbr_sender_1);

op_pk_nfd_get(pk_ptr,"my_nbr_1",&my_nbr_nbr_id_2[0]);
op_pk_nfd_get(pk_ptr,"my_nbr_2",&my_nbr_nbr_id_2[1]);
```

```
op_pk_nfd_get(pk_ptr,"my_nbr_3",&my_nbr_nbr_id_2[2]);
op_pk_nfd_get(pk_ptr,"my_nbr_4",&my_nbr_nbr_id_2[3]);

                                                        break;
                                                       case 2:
        nbr_strm[2]=op_intrpt_strm();
        op_pk_nfd_get(pk_ptr,"nbr_sender", &pk_nbr_sender_2);
op_pk_nfd_get(pk_ptr,"my_nbr_1",&my_nbr_nbr_id_3[0]);
op_pk_nfd_get(pk_ptr,"my_nbr_2",&my_nbr_nbr_id_3[1]);
op_pk_nfd_get(pk_ptr,"my_nbr_3",&my_nbr_nbr_id_3[2]);
op_pk_nfd_get(pk_ptr,"my_nbr_4",&my_nbr_nbr_id_3[3]);
                                                              break;
                                                        case 3:
nbr_strm[3]=op_intrpt_strm();
op_pk_nfd_get(pk_ptr,"nbr_sender", &pk_nbr_sender_3);
        op_pk_nfd_get(pk_ptr,"my_nbr_1",&my_nbr_nbr_id_4[0]);
op_pk_nfd_get(pk_ptr,"my_nbr_2",&my_nbr_nbr_id_4[1]);
op_pk_nfd_get(pk_ptr,"my_nbr_3",&my_nbr_nbr_id_4[2]);
op_pk_nfd_get(pk_ptr,"my_nbr_4",&my_nbr_nbr_id_4[3]);
                            break;
                                                        default:op_sim_end("error","","","");
                                              op_sim_message("exiting nbr_exchange", "");
```

APPENDIX C: FORWARD DISCOVERY MODULE

sprintf(str1, "entering the advertisement state for node: %d",node_id);

If(packet is coming from its own source)
{
set source id into packet
set node id as multicast source
set multicast session number
set own id in destination for returning packets
set number of hops into the packet
set the next return as one
copy your neighbors into the advertisement packet
set own id as node_visited_0 in the node visited array
send packet to next stream available
}
else
{
if (discovery packet is coming from another node)
{
get multicast information(source, and session)
get sender's id
get number of hops from the packet
get the next return
increment the hop count
get sender neighbors' id
get nodes visited from the packet
insert own id into node_visited[i] array
}
if (number of hops is equal to maximum number of hops as specified in the packet)
{
set number of hops traveled as the hop count
decrease the hop count
set the number of hops for reverse path
change the packet type to the number that identifies a backward packet
wherever the current node's id appears in the list of neighbors of neighbor one, replace it by -1
wherever the current node's id appears in the list of neighbors of neighbor two, replace it by -1
wherever the current node's id appears in the list of neighbors of neighbor three, replace it by -1
wherever the current node's id appears in the list of neighbors of neighbor four, replace it by -1
*select a receiver for the packet. /*This receiver cannot be a neighbor of the sender and it cannot be a neighbor of the sender's neighbor*/*

}

if (anyone of the neighbors of my neighbor one does not have a value of -1)

{

select neighbor one as a possible receiver for the packet

go and check neighbor two as a possible receiver

set the count of neighbors to zero

}

/ If my_nbr_nbr list has a sender's neighbor in the list is because my nbr is an nbr of the sender's nbr*/*

*/*if any neighbor of my neighbor one is in the list of the sender's neighbors is because my neighbor one is a neighbor of one of the sender's neighbors, and therefore node one is not eligible to receive the packet */*

go and check neighbor two as a possible receiver

/ if any neighbor of my neighbor one is the sender of the packet, node one is not eligible to receive the packet */*

go and check neighbor two as a possible receiver

set the source id

copy your neighbors into the packet

set the hop count

send the packet to the stream connected to the selected node

go to the end of the module

/ start the process to check the second neighbor */*

increment the count of neighbors

*if(the count of neighbors is equal to the maximum number of neighbors) /*generate a backward packet*/*

goto gen_back

/ the process repeats for the rest of the neighbors until a neighbor is selected or a backward packet is generated */*

sprintf(str1, "exiting the adv state for node: %d",node_id);

APPENDIX D: BACKWARD DISCOVERY MODULE

sprintf(str1, "entering the bkwd state for node: %d",node_id);

if (a backward packet has arrived to the multicast node)
{
if(the forward packet traveled the number of hops specified in the packet as next return)
{
get and store the next return
calculate the next return
set the next return into the packet
}

else
{
count the number of visited nodes
set the registration center in the middle of the number of visited nodes

find and set the location of the registration center
send the packet back through the arrival stream
}
}

sprintf(str1, "Exiting the bkwd state for node: %d",node_id);

APPENDIX E: REGISTRATION CENTER ADVERTISEMENT

sprintf(str1, "entering the reg_center_adv state for node: %d",node_id);

```
get destination for the packet
if (destination is equal to the node's id)
        {
        /*this is the registration center*/
        set destination for the returning packets
        set source's id
        reset the number of hops to zero
        initialize the number of packets to zero
        reset number of visited nodes to zero
        set neighbors into the packet
        reset elements of the registered nodes array
        set packet type
        multicast the packet to neighbors
        }
        if (destination is not equal to node's id)
                {
                get multicast session number
                get source's id
                get hop count
                increase hop count
                get sender's neighbors
                get visited nodes
                update visited nodes
                }
                if (node is already registered)
                        switch control to already registered check point
                if (node is not registered)
                generate a random number
                if (random number is above a threshold number)
                        {
                        register the node for the multicast session
                        get the multicast session
                        update registered node array
                        check hop count
                        }
                if (hop count is equal to max_hop_reg specified in the packet)
                        {
                        set number of hops traveled
                        decrease the hop count
                        set the reverse path
```

set packet type
send the packet back
}
if (hop count is not equal to max_hop_reg specified in the packet)
{
set all elements with own id in the neighbors of neighbors array to -1
check all eligible neighbors
}
if (there are no eligible neighbors or all neighbors received the packet)
send the packet back
if(there are eligible neighbors)
select a neighbor and send the packet to it

if(the node is already registered for the mc session)
{
goto alreadyregistered;
}

initialize the random variable used to decide registration in the multicast session
generate a random number
if (the random number is less than a threshold number)
don't register
else if (random number is greater than or equal to a threshold number)
{
register for the multicast session
include node id into the node_registered array
get nodes registered from the packet and store them in the node_registered[i] array
set new node registered into the packet
}

*alreadyregistered: /*nodes already registered or nodes which do not register jump to this point where the process continues to select a neighbor as a receiver for the packet, as presented in section 6.4.1*/*

sprintf(str1, "exiting the reg_center_adv state for node: %d",node_id);

APPENDIX F: REGISTRATION CENTER BACKWARD ADVERTISEMENT

sprintf(str1, "entering the reg_center_back state for node: %d",node_id);

```
if(final destination is not equal to node's id)
{
get reverse path
decrease number of reverse hops
set new reverse path
find the next node in reverse path
send the packet to first node in reverse path
}
else if(final destination is equal to node's id)
        {
        get nodes registered for the multicast session
        count and add new registered nodes
        count number of neighbors
        count number of packets delivered
        if (number of neighbors is equal to number of packets)
                {
                /*all branches have received the data*/
                get the path to the multicast source
                /*this path was found when the multicast source first contacted the
                registration center*/
                set reverse path to the multicast source
                set initial reverse number of hops
                set number of nodes registered
                set source's id
                set destination for the packet
                set the packet's type
                send the packet to the first node in the reverse path
                }
                else if (number of neighbors is not equal to number of packets)
                        send control to the reg_center_adv module
        }
```

sprintf(str1, "Exiting the reg_center_back state for node: %d",node_id);

APPENDIX G: REGISTRATION CENTER TO MULTICAST SOURCE MODULE

sprintf(str1, "entering the reg_to_mc state for node: %d",node_id);

get destination from the packet

if (destination is not equal to node's id)
{
get and decrease reverse number of hops
get reverse path
determine and send to next node in the reverse path
}
if(destination is equal to node's id)
{
get path to registration center
get members for the multicast session
log members for the multicast session
set the path to the registration center
set the packet type
embed data into packet
send packet to first node in the multicast to registration center path
}

APPENDIX H: FORWARD DATA MODULE

```
sprintf(str1, "entering the fwd_data state for node: %d",node_id);
if (  the current node is the registration center)
        {
        set node id as source id
        set node id as destination for returning packets
        set initial number of hops into the packet
        insert neighbors of current node into the packet
        set current node as node_visited_0
        initialize node_visited_i array to  0
        set the packet type as a number that identifies a forward data packet
        copy data into packet
        send the packet to a stream connected to an available receiver
        increment the number of packets
        count the number of streams with nodes connected to them
         increment the receiver number
if ( the number of receivers equals the number of stream)
 reset the receiver to zero
        else if ( the current node is not the registration center)
                {
        Read multicast information
        if (current node is registered for the multicast session)
        get data and store it in the data_user variable
        Read sender's id
        Read the number of hops from the packet
        increment the hop count
        read neighbors of the sender
        read nodes visited by the forward data packet
        include current node's id as node visited i
        update nodes visited into the packet
        if (hop count equals  maximum number of hops from the registration center)
        {
        set hops traveled equal to hop count
        decrease the hop count
        set the reverse path number of hops as the hop count
        set the packet type as a number that identifies a backward data packet

        send the packet to stream that carried the incoming packet
         }
                else
                        {
                        Execute the process of selecting an eligible node to receive the
packet as described in section 6.4.1
                        }
sprintf(str1, "exiting the fwd_data state for node: %d",node_id);
```

9. REFERENCES

[1] Vu Anh Pham and Ahmed Karmouch, “Mobile Software Agents: An overview”, IEEE Communications Magazine, July 1998.

[2] Osama H. Hussein, Tarek N. Saadawi and Myung Jong Lee, " Probability Routing Algorithm for Mobile Ad-Hoc Network's Resources Management ", IEEE Journal on Selected Areas on Communications (JSAC) vol. 23, No.12, December 2005.

[3] Y. Breitbart, M. Garofalakis, R. Rastogi, S. Seshadri, A. Silberschatz, C. Martin, “Topology Discovery in Heterogeneous IP Networks,” Bell Labs, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974.

[4] W. Fenner and S. Casner, “A ‘traceroute’ facility for IP multicast,” Internet Engineering Task Force (IETF), draft-ietf-idmr-traceroute-ipm-*.txt, July 2000.

[5] K. Sarac, and K. Almeroth “Tracetree: A Scalable Mechanism to Discover Multicast Tree Topologies in the Internet.” IEEE/ACm Transactions on Networking, Vol. 12, No. 5, October 2004.

[6] N. Minar, K. H. Kramer, and P. Maes. Cooperating Mobile Agents for Dynamic Network Routing . In Alex Hayzeldon, editor, Software Agents for Future Communication Systems, chapter 12.Springer-Verlag, 1999. http://www.media.mit.edu/~nelson/research/routes-bookchapter/

[7] R. RoyChoudhury, S. Bandyopadhyay, and K. Paul, “ A Distributed Mechanism for Topology Discovery in Ad Hoc Wireless Networks using Mobile Agents.” 2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing, 11 August 2000, Boston, Massachussets, USA.

[8] R. Shoonderwoerd, et al. (1997) “Ant-based load balancing in telecommunication Networks.” Adaptive Behavior, 5(2): 169207, 1997

[9] Swarm Intelligence From Natural to Artificial Systems, by Eric Bonabeu, Marco Dorigo, and Guy Theraulaz. Chapter 2, pp. 25-147. Oxford 1999.

[10] Computer Networks, by Andrew S. Tanenbaum,
Third Edition, Chapter 5, pp. 358-359, Prentice Hall, 1996.

[11] Algorithms in C, by Sedgewick Robert
Addison Wesley Pub. Co, c1990

[12] S.-J. Lee, M. Gerla, and C.-C. Chiang, “On-Demand Multicast Routing Protocol,” IEEE WCNC’99, New Orleans, LA, September 1999, pp.1298-1304.

[13] C. E. Perkins and E. M. Royer, “Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol,” Proceedings of the ACM/IEEE MOBICOM’99, Seattle, WA, Aug. 1999, pp. 207-218.

[14] C. E. Perkins and E. M. Royer, “Ad-hoc On-Demand Distance Vector Routing,” Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90-100, New Orleans, LA, February 1999.

[15] Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pp. 139-172, Addison-Wesley, 2001.

[16] L. Ji and M.S. Corson, “A lightweight Adaptive Multicast Algorithm,” Proceedings of the IEEE GLOBECOM’98, Sidney, Australia, Nov. 1998, pp. 1036-1042.

[17] Vincent D. Park and M .Scott Corson, “A highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks,” Proceedings of IEEE INFOCOM’97, Kobe, Japan April 1997.

[18] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Shen, “The Broadcast Storm Problem in a Mobile Ad-hoc Network,” Mobicom’99, August 1999.

[19] Wei Peng, and Xi-Cheng Lu, “On The Reduction of Broadcast Redundancy in Mobile Ad-hoc Networks,” Proceedings of the 2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing MobiHoc’2000, Boston, MA, USA, Aug 2000, pp. 129-130.

[20] Computer Networks, 3rd ed. By Andrew S. Tanenbaum, Chapter 5, pp. 370-374, Prentice Hall.

[21] Ballardie, T., Francis, P., and Crowcroft, J., “Core Based Trees (CBT),” Proceedings of the SIGCOMM’93 Conference, ACM, pp. 85-95, 1993.

[22] Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pp. 139-158, Addison-Wesley, 2001.

[23] Chandong Liu, Myung J. Lee, Tarek N. Saadawi, “Core-Manager Based Scalable Multicast Routing.”

[24] Danyang Zhang, Sibabrata Ray, Rajgopal Kannam, S. Sitharama Iyengar, “A New Recovery Mechanism for Reliable Multicasting.”

[25] Don Towsley, Jim Kurose, Sridhar Pingali, “A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicasting Protocols.”

[26] Sanjoy Paul, Krishan K. Sabnami, John C. H. Lin, Supratik Bhattacharyya, "Reliable Multicast Transport Protocol."

[27] Katia Obraczka, Gene Tsudik, "Multicast Routing Issues in Ad Hoc Networks."

[28] "Reliable Broadcast Protocols in Unreliable Networks" Networks Vol. 16 pp. 381-390, 1985.

[29] Ali M. Roumani, Hossam Hassanein, "A Scheme for QoS-Based Dynamic Multicast Routing."

[30] Kwan-Wu Chin, Mohan Kumar, "AMTree: An Active Approach to Multicasting in Mobile Networks."

[31] Ching-Chuan Chiang, Mario Gerla, Lixia Zhang, "Shared Tree Wireless Network Multicast."

[32] Eric Fleury, Yih Huang, Philip K. McKinley, "On the Performance and Feasibility of Multicast Core Slection Heuristics."

[33] Subir Kumar Das, B. S. Manoj, C. Siva Ram Murthy, "A Dynamic Core Based Multicast Routing Protocol for Ad Hoc Wireless Networks."

[34] Barbara D. Gannod, Abdol H. Esfahanian, Eric Torng, "Source-Limited Inclusive Routing: A New Paradigm for Multicast Communication."

[35] Vachaspathi P. Kompella, Joseph C. Pasquale, George C. Polyzos, "Multicast Routing for Multimedia Communication."

[36] D. W. Wall, "Mechanisms for broadcast and selective broadcast," PhD thesis, Stanford Univ., June 1980.

[37] R. Gray, D. Kotz, S. Nog and G. Cybenko, "Mobile Agents for Mobile Computing." Technical Report PCS-TR96-285, Department of Computer Science, Darmouth College, Hanover, NH 03755, May 1996.

[38] Sneha K. Kasera, Jim Kurose, and Don Towsley, "A comparison of server-based and receiver-based local recovery approaches for scalable reliable multicast ," in Proceedings of IEEE INFOCOM '98, San Francisco, CA, USA, March 1998.

[39] C-K Toh, A novel distributed routing protocol to support ad-hoc mobile computing, IEEE International Phoenix Conference on Computer & Communications (IPCCC'96).

[40] K. Paul, S. Bandyopadhyay, D. Saha and A. Mukherjee, Communication-Aware Mobile Hosts in Ad-Hoc Wireless Network, Proc. of the IEEE International Conference on Personal Wireless Communication, Jaipur, India, Feb. 1999.

[41] R. Braudes and S. Zabele, “Requirements for Multicast Protocols,” RFC 1458, May 1993.

[42] A. Ballardie, P. Francis and J. Crowcroft, Core Based Trees (CBT): An Architecture for scalable inter-domain multicast routing, in: SIG-COMM’93, San Francisco, CA (1993) pp. 85-95.

[43] C. Papadopoulos and G. Parulkar, “Implosion control in multipoint transport protocols,” in Proc. IEEE Comp. Comm. Workshop, Sept. 1995.

[44] R. Chalmers and K. Almeroh, “On the topology of multicast trees,” IEEE/ACM Trans. Networking, vol. 11, pp. 153-165, Feb. 2003