A Method for Assessing the Performance of Multicast Algorithms in Wireless Networks

by

Amina Radyastuti

A Thesis
Submitted to the Faculty of Graduate Studies at University of Manitoba
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

Department of Mechanical and Manufacturing Engineering
Faculty of Graduate Studies
University of Manitoba

A Method for Assessing the Performance of Multicast Algorithms in Wireless Networks

BY

Amina Radyastuti

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Abstract

In a mobile wireless environment, mobile nodes often form an arbitrary and dynamically changing network topology. In some mobile networks, such as ad hoc, multicast is often used to support data distribution to many receivers by multihop infrastructureless communication. There have been many algorithms and protocols developed for multicast routing in general but so far there is no simple standard approach for deciding which multicast algorithm is the best for a network with changing topologies. It is difficult to establish this standard since these algorithms work in different ways. For this reason, we focus on the main feature that multicast algorithms have in common, which is the capability to deal with a dynamic network.

A dynamic network can be characterized by its changing topology. In order to study the effect of changing topology on the ability to maintain communication between mobile nodes, we develop a method to assess properly any multicast algorithms based on two new parameters. The two proposed metrics are the average number of arcs per node and the radius of the network topology. Each arc represents direct reachability between any pair of nodes. The radius of the network topology is important to measure the movement of nodes within time unit. We present a new approach for using these two parameters to represent network complexity in a simple way. These parameters are used as direct variables input in assessing multicast algorithms.

What we are investigating is how to use our proposed metrics to measure and see if the arrangement of the nodes in the network affects the algorithm performance. We set up an experiment that can be used to measure the quality of the existing, and even the future, multicast routing algorithms. The experiment involves the movement of the nodes so that any algorithm can be tested to analyze its robustness towards different topologies. Our contribution is a new approach for assessing the multicast performance in a network by exploring the characteristics of the network itself. The result is a tool that is very useful to decide whether any multicast algorithm is good enough to be applied in mobile wireless networks.

Acknowledgements

First, I would like to express my sincere gratitude to my advisor, Dr. Attahiru S. Alfa, for his generous guidance, support, and knowledge sharing throughout my academic life. Dr. Alfa is more like a mentor to all of us, his students, and I will always appreciate that.

I would like to thank TRLabs Winnipeg for providing financial support and a great research environment with helpful people.

I would also like to thank my thesis committee, Dr. Attahiru S. Alfa, Dr. Tarek ElMekkawy, and Dr. Kenneth Snelgrove, for taking time and evaluating this thesis.

I also thank my friends, Shirley, Irene, Daglenia, Budi, Yuya, Sufia, Robert, and Xiaolan, for trusting and cheering me no matter where they are.

Special thanks go to my best friend Deni for supporting me continuously. I thank him for motivating, trusting, and always being there for me. And especially for not letting me starve during the writing of this thesis.

I dedicate this thesis to my whole family, especially to my dearest Mom, Erna Santoso, and my sister, Ardina Rasti, for their enormous encouragement, support, and boundless love.

Finally, I am forever grateful to God for blessing me with great love and strength.

Contents

List of Tables

List of Figures

Acronyms

AMRIS	Ad hoc Multicast Routing protocol utilizing Increasing id-numberS
AMRoute	Ad hoc Multicast Routing
NUMofARCS	Average number of arcs per node
ODMRP	On-Demand Multicast Routing Protocol
topoRADIUS	Radius of the network topology
WSNs	Wireless Sensor Networks

Chapter 1

Introduction

1.1 Description

In mobile wireless networks, mobile nodes interact with each other. The mobile nodes may represent users, servers, and routers. These mobile nodes form an arbitrary and dynamically changing topology of a network because it cannot be predicted when and where the nodes are going to move, how they will move, or why they move. Alternatively, they might not move but simply stay dormant. The delivery of information between those nodes has to adjust to ever changing conditions. They must maintain interaction and communication during each session to minimize information loss. Since high mobility is a characteristic of these nodes, how to maintain communication between these nodes can be a problem.

Multicast is a method of sending a message from a single source to multiple destinations in one operation. There are three primary methods of implementing multicast that differ in the processes happening at the source. The first method is to broadcast the message to all nodes in the networks even when some are not intended destinations (Figure 1.1(a)); hence, multicast is a subset of broadcast. The second method involves the source duplicating the message in as many copies as the number of destinations, and have each copy of the message sent to each destination in one operation (Figure 1.1(b)). This method makes use of multiple point-to-point unicast transport connections. These two methods use a large amount of bandwidth resources only to send unnecessary copies of the message. The last method is the pure multicast, which is the sending of only one message in a single operation and have it duplicated where it is necessary (Figure 1.1(c)). This method uses network bandwidth more efficiently since there is only one message (or a copy of it) traversing the link, which means that it reduces the number of messages that are thrown into the network. However, pure multicast needs considerable explicit multicast support at some points, specifically at the routers that have multiple outgoing links, so that one (copy of) message is duplicated as the number of outgoing links to reach the destinations.

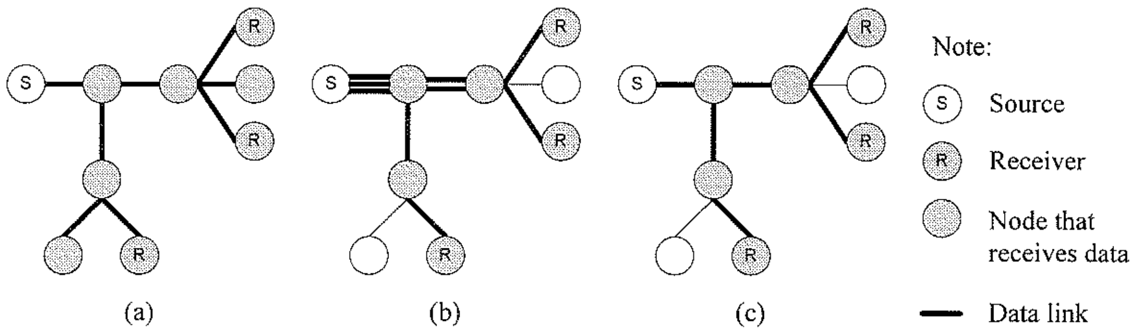

Figure 1.1: Three methods of implementing multicast. (a) Broadcast. (b) Multiple unicast. (c) Pure multicast.

What really distinguishes multicast in a dynamic network from multicast in a static network is that the network is now scattered with mobile nodes. Not only does the location of each mobile node change dynamically within a multicast group, but also the number of the member nodes in the multicast group changes as the current member nodes decide to leave the multicast group or other non-member nodes want to join the group. With multicast, only some mobile nodes that are intended to be the source and the receivers are grouped together. These mobile nodes are joined in a tree or in a mesh topology. Each of these topologies has its own advantages and disadvantages. Either approach is well suited for mobile wireless networks because they can handle the rapid changes of the networks. The problem, however, is to select the type of multicast approach that could improve data delivery over wireless networks by utilizing the available network resources. In addition, it is important to understand how mobility in wireless networks impacts the chosen multicast algorithm.

Multicast in dynamic networks should deal with obvious problems such as an unexpected disconnectivity even if it occurs only in a short duration of time. Continuous communication is required so that there is no delay or loss of information, which is crucial in mobile wireless communication. Therefore, two special characteristics of multicast in dynamic networks that are not required in static networks are: i) an ability to detect nodes that are not responding because they are out of range and ii) an ability to update multicast routes any time a change in topology is detected. This leads to an important issue of how multicast algorithms operate when mobile nodes pop up and down rapidly and unexpectedly.

In today's market, people increasingly use their mobile devices to communicate with others. It offers the freedom of being mobile while communicating. This includes data communication as well. However, when a user chooses to send data to multiple receivers, multicast could be a more effective means of accomplishing this. In general, multicast can help in reducing network overload in these situations.

1.2 Motivation

While mobile communication is becoming ubiquitous, technology used in mobile devices is also changing rapidly. Now, it is possible to use mobile devices even when there is no fixed infrastructure. This network environment, where the nodes pop up and down unpredictably, is known as an ad hoc network. Furthermore, since pairs of nodes in ad hoc networks are often outside the transmission range of each other, data must be relayed over several hops before reaching its final destinations. Multicast is attractive because it can support data distribution to many receivers where multihop wireless communication is the only feasible means.

In emergency situations, such as serious collisions in remote areas and disaster recovery, rapid-response is necessary to mitigate damage. When any collision or disaster happens in a remote place, all authorities must be advised. Since there is seldom fixed infrastructure in remote places, multicast is applicable because it can run in ad hoc networks. If all authorities are equipped with mobile devices so that they can be considered as a group, multicast is required to maintain communication between them. Multicast may also use non-member nodes to forward data reducing the dependency on fixed infrastructure. When multicast is applied in ad hoc networks, mobile devices need only hardware with which they connect to each other. Any pair of mobile nodes are connected with each other when they are within the transmission range of each other. If a single node goes down, mobile nodes simply find other on-the-fly nodes and forward multicast data. A major problem in ad hoc networks is that mobile nodes need enough power to stay alert and sufficient amount of bandwidth to deliver data. The optimum use of bandwidth is important because wireless links do not have as many resources as the wired ones. With multicast, network links use only a minimum amount of bandwidth and it protects data from being delayed or lost.

Many multicast routing algorithms and protocols have been developed and each has its own features. It is difficult to choose among existing multicast routing protocols because there is no simple standard approach for deciding which multicast algorithm is best for dynamic networks. Another major problem is that existing routing protocol metrics are dependent on the type of a protocol and its features. This problem can lead to an unfair assessment when comparing different types of multicast routing protocols. The only way to go further in determining which multicast algorithm to use is by having a common foundation from which to start. Since the most obvious thing that all dynamic networks have in common is a network with scattered mobile nodes, the arrangement of these nodes is an important factor that needs further exploration.

1.3 Objectives of the Research

Since there is an increasing demand for multicast technologies, the scope of this work could be very broad. This thesis deals with the routing of multicast in mobile wireless networks, specifically ad hoc networks. The focus is on introducing new metrics that can be used for comparing different routing algorithms.

The objectives of the research are as follows:

a. To propose new metrics that are analytically determined and can be used for measuring the performance of common multicast routing protocols.
b. To develop a method to assess any multicast algorithms based on the proposed parameters.
c. To study the effect of changing topology on the ability of mobile nodes to maintain communication.

1.4 Methodology

Literature is reviewed to assess the state of the art in multicast. In order to avoid any bias while comparing multicast algorithms, the focus is on the main feature that multicast routing algorithms have in common, which is the capability to deal with a dynamic network. New metrics are analytically determined for assessing common multicast routing algorithms. Based on these new parameters, the performance of the existing algorithms can be analyzed to study the efficiency of these algorithms and for making comparison with regard to different network topologies. Investigation is also done on how to use the new parameters to measure the changes of network topology and see if the arrangement of nodes affects multicast routing. A simulation study is used for doing the experiment to measure the quality of the existing, and even the future, multicast routing algorithms. The experiment involves the mobility of nodes so that the same algorithm can be tested to analyze its robustness towards different network topologies. Network topologies are randomly generated for a fixed number of nodes so that simulations run in consistency basis. The final step is to apply the chosen multicast routing algorithms to each of the topologies.

1.5 Thesis Organization

This thesis is organized as follows. Chapter 1 presents the introduction of this work. Chapter 2 presents a review and analysis of literature review on multicast routing in ad hoc networks and some other works in routing performance metrics. This chapter also presents the detail analysis of the multicast routing algorithms that are chosen to be compared. The factors that motivate this research are also found from this review of the literature. Chapter 3 presents the new metrics that could be used to compare the performance of existing multicast routing algorithms. The simulation variables are then presented in Chapter 4. Chapter 5 presents the evaluation of the performance of the algorithms based on the proposed routing metrics. Chapter 6 concludes this work by summarizing the major contribution of this thesis and it also presents future directions of potential development of performance comparison study.

Chapter 2

Literature Review

2.1 Ad Hoc Networks

An ad hoc network is a dynamic network environment where mobile hosts can form and deform a network “on-the-fly” without the need of any fixed infrastructures (Figure 2.1(b)). It is a type of wireless network; the other is based on fixed infrastructures that needs support from base stations (Figure 2.1(a)). The ad hoc type becomes important in industry because it enables mobile devices to communicate using whatever means available – in this case, other mobile devices – even in remote places where there is no base station.

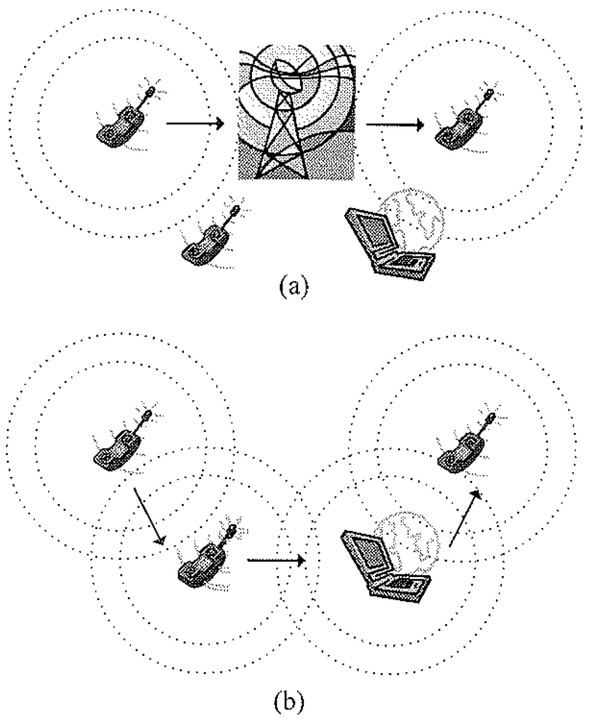

Figure 2.1: A wireless network that is (a) infrastructure-based, and (b) infrastructureless

A mobile ad hoc network is formally defined in RFC 2501 [26] as an autonomous system of mobile nodes that consists of mobile platforms that are free to move arbitrarily. It is also defined in [1] as an autonomous system of mobile hosts (also serving as routers) connected by wireless links, the union of which forms a communication network modeled in the form of an arbitrary communication graph. This definition emphasizes the distinguishing feature of ad hoc networks where each mobile host should be able to perform as a router when it is necessary.

Ramanathan and Redi [25] presented a brief overview of ad hoc networks. They reviewed the definition, the key assumptions, and the significant features of ad hoc networks. They also presented open problems in ad hoc networks and their future. One of the open problems presented is the scalability of a network when the number of nodes in the network increases. This problem was identified but left unresolved. They did not come up with any possible solution. Yet, this could actually be solved if there is a method for measuring the performance of routing algorithms for any number of nodes in the network and investigating the possibility that increasing number of nodes might not give major impacts to the algorithms performance.

The latest development of ad hoc networks is wireless sensor networks (WSNs). A WSN is a specific type of wireless ad hoc networks in which a collaboration of a large number of sensor nodes, that are scattered in a terrain of interest, interact with the environment to observe its ambient physical condition. The examples of the conditions being sensed and observed are temperature, light, sound, vibration, and radiation. Similar to ad hoc networks, WSNs also have to concern about limited resources availability and possibility of topological change. For this reason, multihop communication is also the obvious solution for solving power constraint. Mhatre and Rosenberg [22] presented various design guidelines for WSNs. They proposed a scheme that allows the sensor nodes to periodically switch between a single hop mode and a multihop mode. This scheme could be valuable when developing multicast routing in ad hoc networks.

2.2 Multicast in Ad Hoc Networks

de Morais Cordeiro *et al* [9] presented the present and future directions of multicast over wireless mobile ad hoc networks. They emphasized that multicasting enables people that reside at different places to participate in the same session through wireless and mobile devices. This means that multicast can be involved much more in the future because there will be more applications developed that require a network ability to provide service with sufficient bandwidth. They also stated that recent multicast protocols in ad hoc networks do not perform well in different applications. This statement implies that there is a need of methods that can provide an answer of which multicast protocol is the most appropriate to use in each application.

Royer and Toh [29] reviewed eight different routing protocols and classified them. They classified ad hoc routing protocols into two classes: i) table-driven and ii) demand-driven (source-initiated). Table-driven routing protocols maintain up-to-date routing information every some period of time. On the other hand, demand-driven protocols only create routes when it is demanded by the source node. However, they was not able to conclude whether protocols from one class are better than the other class or not.

Recently, Papavassiliou and An [24] also reviewed several multicast routing protocols and classified them into groups. They classified ad-hoc multicast routing protocols into proactive and reactive groups. Proactive multicast protocols are similar to table-driven routing protocols in Royer and Toh [29]. Reactive protocols are similar to demand-driven ones. Proactive and reactive multicast protocols are further classified as tree based and non-tree based (Figure 2.2). Some comparisons of those protocols were made but only on qualitave side. They were also not able to provide an answer of the more preferable protocol. They summarized some issues such as dynamic multihop topology, routing information (in)accuracy, resource usage efficiency, reliability, security, and group membership.

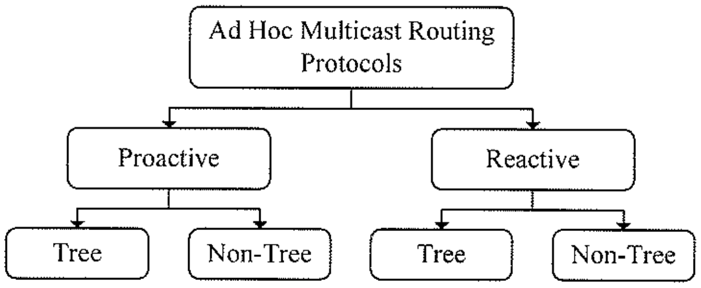

Figure 2.2: Classification of ad hoc multicast routing protocols

Sahasrabuddhe and Mukherjee [30] presented various multicast routing algorithms and their relationship with multicast routing protocols for packet-switched wide-area networks. They categorized multicast algorithms based on the property they attempted to optimize. They also examined various multicast protocols that are employed on the Internet.

Wang and Hou [38] emphasized the QoS (Quality of Service) requirements of continuous media applications that use multicast services. They classified multicast routing problems according to their optimization functions and performance constraints, presented routing algorithms in each problem class, and categorized existing multicast routing protocols. They underlined a set of challenging problems with multicast on the Internet and the importance of efficient solutions. However, they failed to recognize that the problems are actually also applied for multicast in ad hoc networks.

Chiang, Gerla, and Zhang [6] proposed the Adaptive Shared Tree Multicast routing protocol (ASTM) which is an adaptive scheme that combines shared tree and per-source tree benefits. It is a proactive multicast routing protocol and it maintains a single shared tree rooted at a Rendezvous Point (RP). RP is where sender sends multicast towards and receiver send join requests to. It is preferably selected among nodes with low mobility. It allows switchover between the shared tree and the per-source tree to reduce any delay due to possible shorter distance between the receiver and sender directly than the distance between receiver and RP. The performance metrics that they considered are throughput and control message overhead.

Another proactive protocol, but non-tree based, was proposed by Garcia-Luna-Aceves and Madruga [11]. They proposed the Core-Assisted Mesh Protocol (CAMP) that is an extension of the notion of core-based tree (CBT), which is used in static networks. CAMP defines a shared multicast mesh for each multicast group. Since it is a mesh, which provides at least one path from any node that is a source to any node that is a receiver; it provides richer connectivity than a tree-based topology even though the nodes within the group move frequently. It uses cores to limit the control traffic needed for receivers to join multicast groups. When a node wants to join the group, it selects the core towards which the join request may be sent. Any router that is a regular member of a multicast group and receives a join request is free to transmit a join acknowledgment (ACK) to the sending router. When the origin or a relay of a join request receives the first ACK to its request, the router becomes part of the group.

Royer and Perkins [28] extended the Ad-hoc On-Demand Distance Vector (AODV) routing protocol to offer novel multicast capabilities that follow naturally from the way AODV establishes unicast routes. This extended version is called AODVM and it falls into the category of reactive, tree-based protocols.

Another reactive, tree-based protocol was proposed by Bhattacharya and Ephremides [4]. The protocol is called the Distributed Multicast Routing Protocol (DMRP) and it is a distributed, source-initiated protocol that combines multicast routing with resource reservation and maintain connections to desired destinations. This algorithm tries to accomplish multicast routing by giving two degrees of freedom (the frequency at which to transmit and power level) to each node.

Ji and Corson [15] presented another reactive, tree-based protocol called the Lightweight Adaptive Multicast protocol (LAM). LAM builds a group-shared tree for each multicast group and takes the concept of core-based tree. The tree is centered at a pre-selected node called a CORE. If the source is not part of the tree, it forwards the data through the CORE.

LAM is coupled with a specific underlying unicast routing protocol TORA (Temporally-Ordered Routing Algorithm), which is source-initiated. It is lightweight because the tree maintenance phase does not utilize timers and event-triggered. LAM is claimed not to introduce any additional overhead but, unfortunately, only during stable topology and constant group membership, which is not the most common event in ad hoc networks.

Corson and Batsell [8] presented the Reservation-Based Multicast (RBM) which is a reactive, tree-based routing protocol. It combines multicast routing, resource reservation, and admission control. It uses the concept of Rendezvous Point (RP) and it is used for routing process that can be broken into two stages: source-to-RP and RP-to-destination.

Toh and Bunchua [33] proposed the Associativity-Based Ad hoc Multicast (ABAM) which has four components: i) multicast tree formation per multicast session, ii) handling host membership dynamics, iii) handling the mobility of the nodes, and iv) multicast tree deletion/expiration. They proposed a heuristic tree selection algorithm to derive the multicast tree.

The last category is reactive, non-tree protocols. Ho *et al.* [13] proposed the Reliable Multicast Routing Protocol (RMRP) based on the argument that keeping accurate state about membership of multicast group is not practical if the set of neighbours changes rapidly. RMRP uses plain flooding that requires each node to keep track only of its current neighbours.

Another reactive, non-tree based protocol is the Forwarding Group Multicast Protocol (FGMP). FGMP was proposed by Chiang, Gerla, and Zhang [7]. This protocol is a mixture between flooding and shortest tree multicast. Any node in a forwarding group is responsible of forwarding multicast packets between any pairs of group members so that when it receives a nonduplicate multicast packet, it will broadcast this packet to its neighbours but only the neighbours in the group can broadcast the packet consecutively. The major problem is how to elect and maintain the set FG of the nodes in the forwarding group.

Jetcheva and Johnson [16] presented Adaptive Demand-Driven Multicast Routing (ADMR) protocol and it is compared to On-Demand Multicast Routing Protocol (Subchapter 2.3) using the following metrics: packet delivery ratio, normalized packet overhead, forwarding efficiency, and delivery latency.

Bhattacharya and Ephremides [3] established the beginnings of a complete multicast algorithm that is capable of adapting to topological changes. The algorithm, which combines multicast routing with dynamic frequency allocation and power control, is intended to establish and maintain the maximum number of connection requests while making efficient use of available bandwidth and avoiding congestion which might lead to network collapse.

Gossain, de Morais Cordeiro, and Agrawal [12] presented multicast from its definition, the development of multicast support in the Internet, a detailed description of existing multicast protocols in wired and wireless environment and their comparison. They did not give any solution of which multicast protocols is more suitable for which applications. They only tried to give some basis on which people can select an appropriate multicast for their needs. Their focus was on solutions that make use of infrastructure provided by the wired network and not over infrastructureless ad hoc networks.

2.3 Multicast Routing Protocols

This section will discuss the detail analysis of three multicast routing protocols that are going to be used in the simulation study. Each protocol is chosen as a sample for each category in the classification of ad hoc multicast routing protocols.

Reactive protocols are believed to work better than the proactive ones. This can be seen from a large number of multicast protocols, either newly proposed or developed from the existing ones, that are reactive. It is caused by the on-demand nature of reactive protocols that preserves the use of bandwidth when there is no multicast data in the network. Hence, AMRIS (Ad hoc Multicast Routing protocol utilizing Increasing id-numberS) and ODMRP (On-Demand Multicast Routing Protocol) are chosen as two of the three multicast routing protocols for this comparison study. They are reactive and start to discover multicast routes once a source node has data to send. More specifically, they are chosen because of their structure difference. AMRIS is chosen because it creates a multicast tree and has an interesting feature, which is to use identification numbers in building the tree; while ODMRP is chosen because it creates a mesh for providing alternative routes. ODMRP is also the most widely used multicast routing protocol. The differences between a tree and a mesh are listed on Table 2.1.

Category	Structure	
	Tree	Mesh
Bandwidth required to build the structure	Small amount	Greater
Multiple routes	Not available	Available
Focus	Route-updating	Route-discovery

Table 2.1: Differences between a tree and a mesh

However, proactive protocols still deserve to be considered in multicast because they have a different strength than the reactive ones. Proactive protocols update multicast routes periodically. As a result, a path from the source and each destination is almost always available when there is multicast data in the network. Hence, the time required for message delivery with a proactive protocol is less than the reactive one. Reactive protocols have to discover routes from the beginning before they can forward multicast data. An example of where proactive and reactive protocols have their own advantage and disadvantage when time is divided into five equal intervals can be seen in Table 2.2.

Time interval	Multicast data exists	Bandwidth usage in creating routes		Messages' travel time from a source to receivers	
		Proactive	Reactive	Proactive	Reactive
1	V	V	V	Fast	Slower
2		V			
3		V			
4	V	V	V	Fast	Slower
5		V			

Table 2.2: An example of an advantage and a disadvantage of proactive and reactive protocols

AMRoute (Ad-hoc Multicast Routing) becomes the only algorithm that is proactive among the three. It is chosen because of its special feature, which is to create a tree out of a mesh.

2.3.1 AMRIS

Wu and Tay [39] proposed AMRIS that is a reactive, tree-based protocol. It does not require a separate unicast routing protocol. It creates a shared multicast tree to forward multicast data and this tree is rooted at a special node called Sid. Each node in a multicast session is dynamically assigned an ID number known as msm-id. This ordering number is used to direct the multicast flow. The msm-id increases in numerical value as any node radiates away from Sid. Sid, which is predetermined from amongst the senders, has the smallest msm-id.

There are two main mechanisms in AMRIS: tree initialization and tree maintenance. In tree initialization, a multicast session is created and advertised to all nodes in the ad hoc network. These nodes are differentiated as I-Nodes (nodes that are interested in joining the multicast session) and U-Nodes (the rest of the nodes). This mechanism begins when Sid broadcasts a NEW-SESSION message to its neighbours. Some of the content of this message are Sid's msm-id, multicast session id, and routing metrics. A node that receives the NEW-SESSION message generates its own msm-id that is larger and not consecutive, replaces the msm-id in the message with its own msm-id, and broadcasts the message again. Information derived from the NEW-SESSION message is kept in the Neighbour-Status table for up to T1 seconds to prevent broadcast storms. If a node receives multiple NEW-SESSION messages from its neighbours, it would keep the message with the best routing metrics and generates its own msm-id based on the values from that message.

A node X that wants to join the session would determine its neighbouring nodes that have smaller msm-ids than itself as its potential parent nodes. It sends a unicast JOIN-REQ to one of its potential parent nodes called Y. If Y is already on the delivery tree, Y sends a JOIN-ACK immediately back to X. Otherwise, Y would also try to locate its potential parent by sending another JOIN-ACK and this process is repeated until a node is found on the delivery tree to be the parent node and this node sends a JOIN-ACK that propagates back along the reverse path to X.

Tree maintenance ensures that a node remains connected to the multicast session delivery tree. When a link between two nodes breaks, the node with the larger msm-id is responsible for rejoining. The process is similar to the process of joining in tree initialization. A node that attempts to rejoin the tree executes the Branch Reconstruction (BR) that has two main subroutines: BR1, which is executed when this node has neighbouring potential parent nodes; and BR2, which is executed when this node does not have any neighbouring nodes that can be potential parents.

A node X executing BR1 sends a unicast JOIN-REQ to one of its potential parent nodes Y. If Y is on the tree, it sends a JOIN-ACK back to X, and X rejoins the tree. If Y is not on the tree, Y repeats the process of sending out its own JOIN-REQ to join the tree, provided it has at least one neighbouring potential parent node. Otherwise, Y sends a JOIN-NACK to X. If X receives a JOIN-NACK or timeouts of the reply, it proceeds to join with the next best potential parent node. If none are available, X executes BR2.

A node X executing BR2 broadcasts a JOIN-REQ. The broadcasted JOIN-REQ has a range field R that specifies only nodes within R hops of X are allowed to rebroadcast the JOIN-REQ. A satisfactory node Y that receives the JOIN-REQ sends a JOIN-ACK back to X. If X receives multiple JOIN-ACKs, it chooses one and sends a JOIN-CONF to that parent node Y so that Y may start to forward multicast data to X.

2.3.2 ODMRP

Bae *et al* [2] developed ODMRP that is a demand-driven, mesh-based multicast protocol and it uses the concept of forwarding group [6]. In ODMRP, when a source has data to send, it floods a JOIN DATA packet with data payload attached to construct the routes and establish the group. When a node receives a nonduplicate JOIN DATA packet, it stores the upstream ID and rebroadcasts the packet. When a receiver receives the JOIN DATA packet, it creates a JOIN TABLE packet and broadcasts it to its neighbors. Each node that receives the JOIN TABLE packet would check the next node ID of one of the entries. If it matches its own ID, it is on the right path to the source that means it is part of the forwarding group, and it sets an FG_FLAG and broadcasts its own JOIN TABLE built on matched entries. It is done until all JOIN TABLE packets reach the source and it creates the routes from each source to all receivers so that a mesh of nodes is built as a result. Any multicast group member that wants to send data just has to floods a JOIN DATA packet to refresh routes and membership information.

The strength of ODMRP is its simplicity because it does not need any explicit control packets if one node decides to join or leave the group. If a source wants to leave the group, it just has to stop sending JOIN DATA packets. And if a receiver wants to leave the group, it just has not to send any JOIN TABLE packet as a response of the JOIN DATA packet. Another strength of ODMRP is its ability to exist together with and operates as a unicast routing protocol. Bae *et al.* also concerned about the selection of timer values for route refresh and forwarding group timeout intervals.

2.3.3 AMRoute

Bommaiah, Liu, McAuley, and Talpade [5] proposed AMRoute that creates bidirectional shared multicast trees for data distribution using only the senders and receivers in the group as tree nodes. Even though AMRoute creates a mesh and a tree in order to multicast the data, it is a proactive, tree-based protocol. The mesh is first created, and then it creates a multicast distribution tree using a subset of the available mesh links. The major advantage of this procedure is that the tree does not need to be modified when any changes happen to the network topology provided that the tree routes exist via mesh links.

In mesh creation, receivers and senders join the group. Each group has at least one logical core that is responsible for maintaining the tree and group membership. New group members select themselves as cores initially. Each core periodically floods JOIN-REQ to discover other disjoint mesh segments for the group. When a member node receives a JOIN-REQ from a core of the same group but a different mesh segment, it replies with a JOIN-ACK and marks that node as a mesh neighbour. As a consequence of mesh mergers, a mesh will have multiple cores. One of these cores will emerge as the "winning" core of the unified mesh. The core resolution procedure will win the node with the highest IP address among all nodes.

After the mesh creation, each core sends a periodic control message TREE-CREATE in order to build a shared tree. When a member node receives a nonduplicate TREE-CREATE from one of its mesh links, it forwards the packet to all other mesh links except the incoming one, and marks the incoming and outgoing links as tree links. If it receives a duplicate TREE-CREATE, it sends a TREE-CREATE-NAK back to the incoming link and this link is designated as mesh link. Multicast data is forwarded over the tree, which is loop-free. If a node wants to leave the group, it has to send a JOIN-NAK to its neighbours and do not forward any data packets for the group.

2.4 Routing Metrics

In order to improve multicasting, a performance comparison study is required for investigating multicast protocols. Lee *et al* [19] compared the performance of some of multicast protocols mentioned in the previous subchapter based on the following metrics: packet delivery ratio, number of data packets transmitted per data packet delivered, number of control bytes transmitted per data bytes delivered, and number of control and data packets transmitted per data packet delivered. They did a simulation study under various scenarios such as variety of mobility speed and different number of senders. The simulation results showed that each multicast protocol has its own strength in some circumstances.

Moustafa and Labiod [23] proposed a new multicast routing protocol called Source Routing-based Multicast Protocol (SRMP) and compared it against ODMRP and ADMR under various scenarios using pause time. It is not obvious why pause time is used as a metric because pause time does not represent anything about the network except that the nodes sometimes stop moving for an interval of time.

Since there are many routing protocols in ad hoc networks and there are different metrics, it is difficult to measure the performance of routing protocols and compare them. In his work, Jacobson [14] states that a problem in existing routing protocol metrics is that they are based on simulation results, not on fixed input data. This problem causes a certain metric to be biased toward a group of routing protocols.

According to Jacobson, there are two types of metrics that have already existed. One is performance metrics, which depend on the simulation results; the other is scenario metrics, which are calculated from the input data to the simulation or from the input variables. Jacobson [14] introduces two new scenario metrics for ad hoc networks that are dependent on the current physical condition of the network. Those two new metrics are the density of the network and the direct connectivity rate for the nodes in the network. The density is defined as the weighted number of overlapping radio transmitter areas over time and is calculated as the total overlapping area of circles at any point in time. The direct connectivity is in place between two nodes where they are within the transmitter range of each other and is used to show the average number of other nodes that are in contact with each node over the simulation time.

Density is the only metric that he used in his simulations. He drew a conclusion that density and direct connectivity have similar values. Yet, he stated that direct connectivity could be a better metric to be used. And he also stated that density could accumulate without direct connectivity because a density value exists when there are overlapping areas between two radio transmitter ranges. There are some contradictive things behind those statements.

The density metric was varied indirectly by changing the scenario areas in a set of simulation of this work to provide illustrative values. We could improve this work by proposing new metrics that can be used for common routing protocols and that actually change directly with the changes in the network topology.

Another weakness of this approach can be found in the statement "the existing routing protocol metrics are focusing on the performance metrics and based on the simulation results". This is not entirely true because the performance comparison of different multicast algorithms need to be evaluated by simulating each multicast algorithm in different topologies. What actually exist now are routing protocol metrics that are dependent on the type and feature of the protocol. There is a need to analyze these metrics, particularly the effect of the application of any metrics to different algorithms.

2.5 Application of Multicast

Multicast can support many applications. One area where multicast support is an immediate need is mobile commerce [21, 234-36]. Some class of applications in mobile commerce are mobile and locational advertising, mobile auction, mobile entertainment, proactive service management, and mobile inventory management. Another application is presented by Bhattacharya and Ephremides [3], which is the use of multicast routing and resource allocation in the digital battlefield. Multicast can also be very useful in emergency situation such as search and rescue. Varshney [33] presented the requirements for different applications, such as application in military that requires minimal delay and security; application in distance education that requires high bandwidth and near-real-time; application in intelligent transportation systems that requires dynamic routing of individual vehicles; and application in commercial aircraft that requires current traffic information, the most direct and least time-consuming routes.

2.6 The Issues in Multicast Routing

Although there have been some algorithms and protocols available in multicast over mobile wireless networks, there is no standard approach of deciding which multicast algorithm is the best. It is difficult to establish this standard since those algorithms and protocols run differently in different environments. There is still scope to develop better algorithms and protocols if the factors in multicast in ad hoc networks that really influence the performance of those algorithms could be determined.

This research attempts to assess some of the most common multicast routing algorithms based on new metrics. Another factor that motivates this research is the lack of works that focus on network topology, yet a dynamic topology is one major feature of mobile wireless networks. Most works done in multicast only deal with the creation of new routing protocols that claim to be improvements on others. Two of the newest protocols are ReMHoc [31] and Family ACK Tree (FAT) [20]. Recently, as the latest effort to improve multicast, overlay multicasting in mobile ad hoc networks is investigated. This approach gets contradictory responses. Detti *et al* [10] investigated the effectiveness of overlay and they stated that to this point there are doubts on which multicast protocol is best. The lack of work in developing methods for assessing those protocols could cause an application to use a protocol that would actually be better off using another protocol.

Some of the performance metrics that are being used do not take the variety of the type and the feature of the protocol into account. The introduction of new metrics is the objective of this research since some metrics are only good for particular multicast protocols and some are not directly varied during a set of simulations. We should not evaluate the performance of those protocols only after we apply the algorithms in the network, we must anticipate the possible results before we apply them.

The main idea is to investigate how the arrangement of the mobile nodes in the early state of multicast session can be used to decide which multicast algorithm is more appropriate to use.

Chapter 3

Proposed Metrics

3.1 Background

New metrics proposed here can be used to compare common routing protocols and they have the ability of changing their values automatically during a set of simulations. The proposed metrics would be based mainly on the characteristics of the network topology. The random physical condition of the network is not dependent on any factor and the variation of the topology is influenced solely by its randomness. There are two factors that we exploit in order to build the new metrics, which are reachability and arrangement of the nodes.

3.1.1 Reachability

When any one node has a message to send to some other nodes, the former takes a role of a source while the latter destinations. Since the networks are infrastructureless, as long as the source has at least one other node that is within reach, this other node can facilitate the message sending to one or more destinations that are not within reach.

Reachability of a node, in mobile wireless networks, can be measured by the power level of transmission that a node has. When transmission occurs, the signal power declines as it travels further from its source.

3.1.2 Arrangement of the nodes

The arrangement of the nodes might be the most important factor that influences the performance of any multicast algorithms. This factor is strongly related to the reachability factor mentioned previously. The location of any pair of mobile nodes has a direct effect on the possibility for both nodes to communicate with each other. In a bigger picture, the location of all the nodes in the network affects the likelihood that any mobile node can receive the message intended for it from another mobile node located outside its transmission range.

3.2 Proposed Metrics

In order to study the effect of network topology on the network and the feasibility of maintaining communication between nodes in mobile wireless networks, first we have to characterize a network topology as the combination of the following properties that would become our proposed scenario metrics (3.2.1 and 3.2.2).

3.2.1 Average number of arcs per node

Since our purpose is to observe the performance of multicast algorithms in different kinds of topology, we should focus on the arrangement of the mobile nodes. This property considers the direct reachability of each node. Any node can use different levels of power to transmit data. The higher the power level, the farther the data can be transmitted. Since the power level determines the range of transmission of a node, it can be illustrated as a circle that can be defined as the radius of the node, i.e. for any node i, any other nodes within radius r can be reached directly by node i.

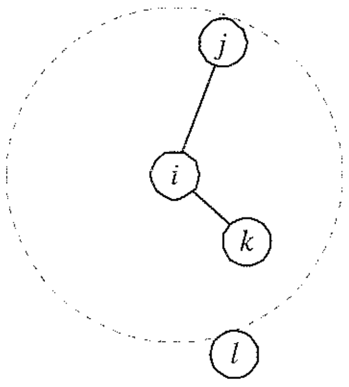

Figure 3.1: Direct reachability from node *i* to node *j* and node *k*

Figure 3.1 shows that node *i*, with radius *r*, can reach node *j* and node *k* directly because they are within its transmission range. We draw a line between those nodes to show their connectivity. When a line connects two nodes, it means that either node can be reached directly by the other node. We cannot connect node *i* with node *l* because node *l* is outside node *i*'s transmission range.

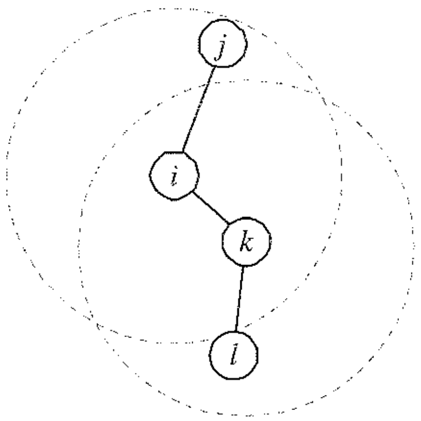

Figure 3.2: Indirect reachability from node *i* to node *l*, or multihop

Since any node outside that radius cannot be reached directly, it needs one or more intermediate nodes for transmission to be accomplished. The example is shown in Figure

3.2, where node *l* can be reached by node *i* only through intermediate node *k*. So even though node *l* is outside the range of node *i*, node *l* is still within the radius of node *k*, which is connected directly to node *i*. In this case, node *i* needs node *k* to transmit data to node *l*.

Since direct and indirect reachability is influenced by the radius of a node, this radius is called the "node-reachability"-radius. This radius, which emerges because of the power level of each node and is assumed to be uniform for each node in the network, is important to see the connectivity among nodes. This radius will affect the average number of arcs per node as a result of the connection that can be made between nodes in the network, which depends on the number of nodes that can be reached directly and or indirectly.

We can formulate the average number of arcs per node (NUMofARCS) with:

$$\text{NUMofARCS} = \frac{\sum a * f_a}{N}$$

where:

a = number of arcs

f_a = frequency of nodes that has a number of arcs

N = number of nodes in the network

The value of a does not necessarily have to be limited within a range. But to shorten calculation time, the range of a is from the minimum to the maximum number of arcs among the existing number of arcs that belongs to each node in the network.

Figure 3.3 gives a simple example of a seven-node network where every node has uniform propagation range. As a result of limited propagation range, those nodes are not connected properly. Node 3 is not able to communicate with any other node.

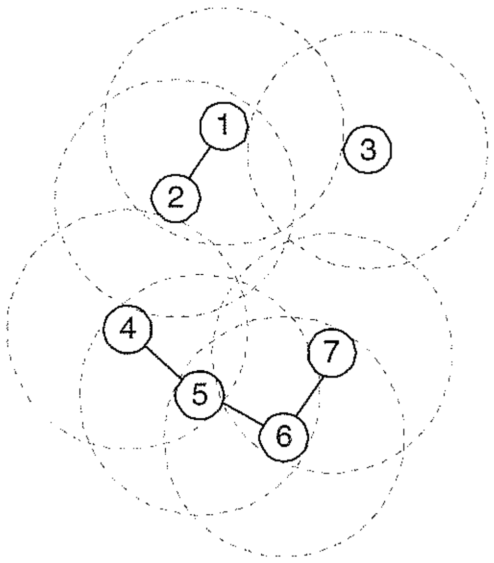

Figure 3.3: An example of connectivity in a network

We can determine the NUMofARCS from the example in Figure 3.3 by building a table below.

Number of arcs (a)	**Frequency (f_a)**	**Multiplication (a^*f_a)**
0	1	0
1	4	4
2	2	4
3	0	0
Total	**7**	**8**

Table 3.1: An example of data that can be used to determine NUMofARCS

When we divide the total value of multiplication in Table 3.1 with the total number of nodes in this particular network, we get the average number of arcs per node, which is:

$$\text{NUMofARCS} = \frac{8}{7} = 1.1429$$

Figure 3.4(a) shows the same network in Figure 3.3 with the arrow indicating that a node is moving in the direction of the arrow is pointing. Node 2 moves into the transmission range of node 4 and node 3 into of node 7. Figure 3.4(b) shows how this movement can affect the connectivity between nodes in the network. It creates a new topology with a connectivity that is different from the one in Figure 3.3 (NUMofARCS = 1.7143).

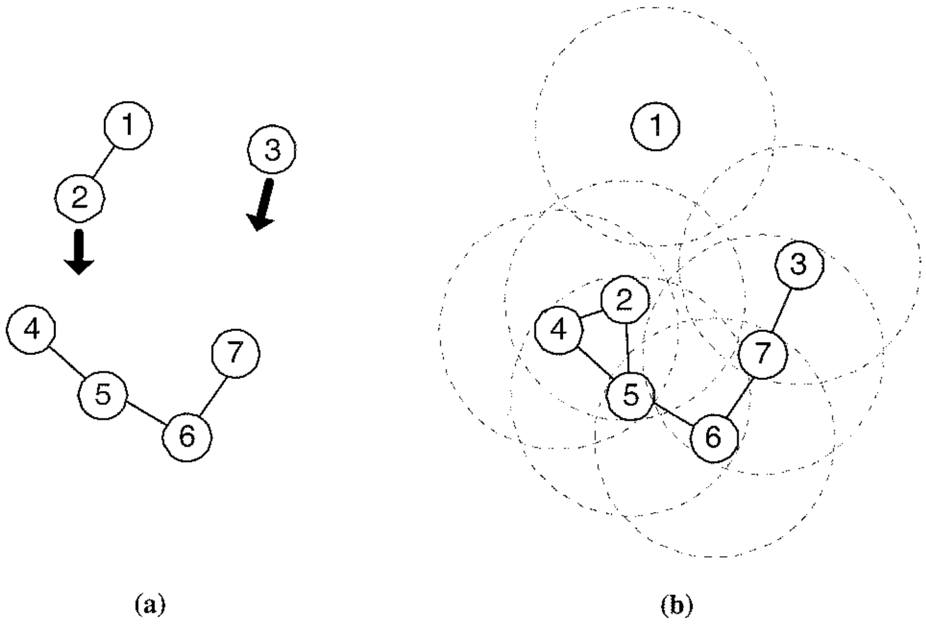

Figure 3.4: An example of movement effect on connectivity. (a) Nodes are moving. (b) The new connectivity.

It is easy to observe that when a node moves closer to other on-the-fly nodes, it can reach and be reached by more surrounding nodes and build stronger connectivity.

In summary, when a network has a large number of arcs per node, it has a greater possibility that the network is more robust with regard to the likelihood that each node has more options to route data to other nodes, especially the ones that are not within its propagation range.

3.2.2 Radius of the network topology

The nodes in a mobile wireless network can be grouped as a set of mobile nodes. In order to cluster these mobile nodes into a group, we have to create a circle that bounds all these nodes.

For example, Figure 3.5(a) shows the beginning of a network where there are seven nodes that are simply scattered. Figure 3.5(b) then shows the same network now with a circle indicating the smallest circle that contains all seven nodes.

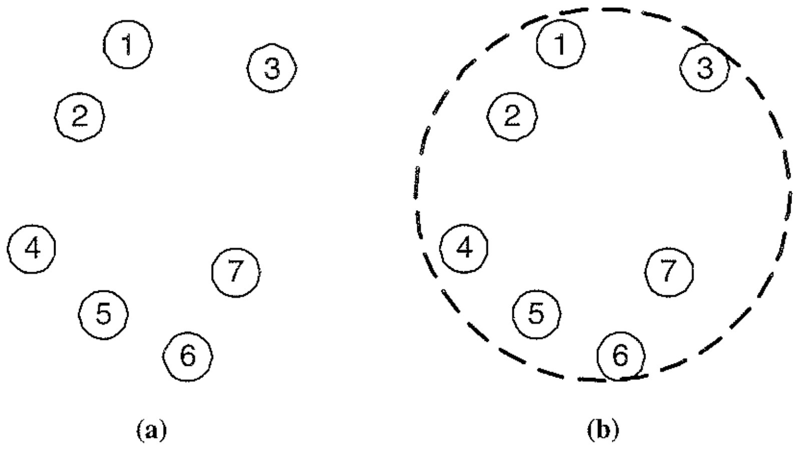

Figure 3.5: An example of a network with (a) scattered nodes only, and (b) the smallest circle that contains all nodes in the network

When a smallest circle that contains all the nodes in the network is created, it can be used to measure the area where the nodes are scattered (Figure 3.5(b)). We can determine this circle by using the easy bounding circle algorithm by Rokne [27].

This smallest circle can be defined as the radius of the network topology. We can use the radius of this circle as another property to characterize the network topology.

The changes in the radius of the network can determine the relative change in network topology to how scattered the mobile nodes are. For example, if any of the nodes in Figure 3.5(b) moves out of the current radius of the network topology, we should enlarge this radius in order to handle this new topology. On the contrary, if any one of the most "outside" nodes in Figure 3.5(b) moves in and the radius of the network topology can become smaller, the coverage of this network will be better. Thus, any node located inside the circle can move freely with no effect on the radius of the network topology as long as it does not move out the circle (this movement might still affect the connectivity between nodes, though). The movement of any node currently located on the edge of the circle possibly affects the radius of the network topology. The latter leads us to the importance of the rate of the changes of the radius of the network topology.

In summary, the rate of the changes of this radius should be considered in each network topology. This radius, which might change in radius per time unit, more likely gives better performance of the network because when it becomes smaller, it means that some nodes would have a better chance to find any other on-the-fly nodes to maintain the connection with other nodes. We characterize the network topology by both properties above and use them as direct variables input.

3.2.3 The difference between the new metrics and the others for network characterization

The new metrics are developed for characterizing a network topology. These metrics will then be used for assessing multicast routing protocols. Their values represent the nodes connectivity and the coverage area of the network.

In comparing different multicast routing algorithms, these metrics are required so that the performance of each algorithm can be measured under various scenarios. The metrics that have been used in most comparison study are mobility speed [19], different number of senders [19], and pause time [23]. The last metric proposed is density [14]. However, those metrics do not really represent what is the most important factor in routing, which is the connectivity of nodes in the network.

When connectivity of the nodes in the network is known, it could facilitate the prediction of how the algorithm would perform. Hence, connectivity is an important factor in routing. The proposed metrics, that characterize the network, represent connectivity.

Mobility speed [19] and pause time [23] have been used as metrics in comparing multicast protocols. They may be related with each other but they do not necessarily characterize the network topology. Neither does different number of senders [19].

Density, which is represented by the shaded area in Figure 3.6, does not necessarily represent connectivity either. As illustrated in Figure 3.6, there exists a scenario where density is present but there is no connectivity between the nodes. Recall that connectivity exists when the nodes are within the transmission range (a node's transmission range is represented by a circle surrounding node) of each other. Density value can increase and decrease such that there is still no connectivity between the nodes. Therefore it is not an appropriate metric for characterizing the network. On the other hand, even a slight change in the values of the new metrics provides information about modification happening in the network.

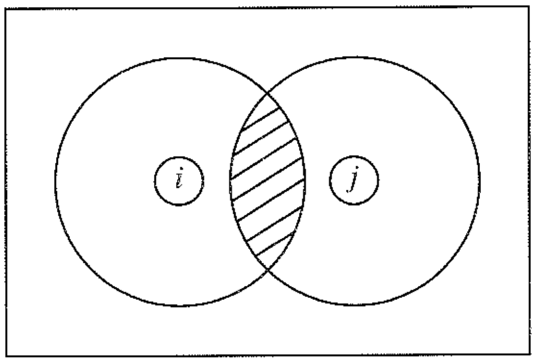

Figure 3.6: An example of density with no connectivity

3.3 Example

In order to understand how the proposed metrics work, we give some illustration in Figure 3.7. Figure 3.7 shows the network with seven mobile nodes. The gray nodes represent the nodes in their original position. The average number of arcs per node in this topology is 3.2 arcs per node, which is obtained from the sum of the number of arcs that each node has divided by the number of nodes. The circle with the dashed line represents the radius of this topology which is 226.20 meters. The procedures used for determining those values will be explained in Subchapter 3.4.

The white nodes represent the mobile nodes after moving from their original position. The nodes are more scattered than before and as a result it has less connectivity. The average number of arcs is now 2.0 arcs per node and the radius of the topology, which is represented by the circle with the solid line, is now 234.77 meters.

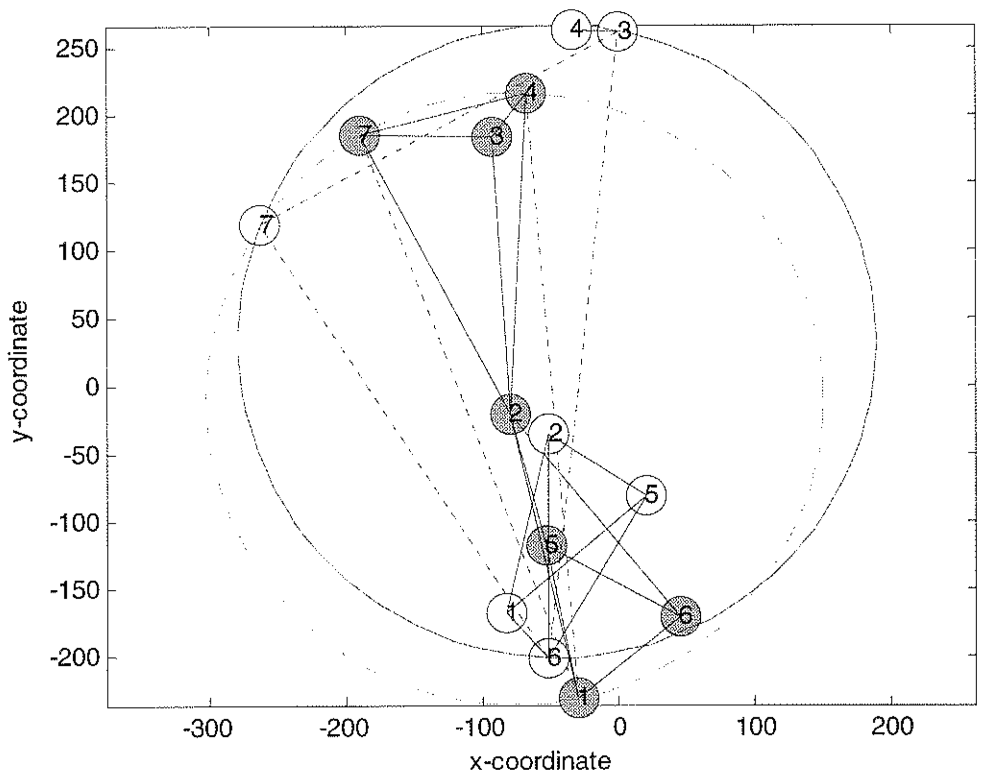

Figure 3.7: An example of the use of the proposed metrics

This illustration shows that the movement of the nodes changes the arrangement of the nodes and also their reachability. The changes in the connectivity and the radius show that they might have significant effects on the performance of the multicast routing algorithms.

3.4 Assumptions of Variables Input

3.4.1 Network Topology

For a fixed number of nodes, network topologies are randomly generated using Matlab. The location of each node is assumed to be on the two-dimensional coordinate system (x, y) and each location is generated randomly in a uniform distribution. We use a uniform distribution so that the generated point sets are well distributed.

The variables input that we need in order to create a set of random topologies are:

a. The fixed number of nodes in the network.

b. The number of network topologies created in a set of simulation.

c. The initial network area to begin with.

d. The maximum network area that is considered to be feasible as the expansion of the initial network area.

Included in the creation of the random topologies is the maximum distance that any node can travel per time unit (d_{max}). This is important for creating the next random topology that has fundamental correlation with the current topology. We assume that each node can pick up any random number in the interval $[0, d_{max}]$ and move as far as that distance in random direction.

3.4.2 Distance between two adjacent nodes

In order to run a multicast algorithm in any network topology, first we have to specify a criterion to link two adjacent nodes. It is important to determine which nodes could be categorized as neighbouring nodes of one node since multicast involves broadcast to initiate the routing to find the multicast membership.

Assume that the criterion to connect one node to its neighbouring nodes is the maximum distance that can be reached by the former node. We represent this criterion by:

$$\left\{(i,j)\middle|\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}\le R\right\}$$

where (*i*,*j*) represents two adjacent nodes *i* and *j* such that the distance between those two nodes (represented by the coordinate of each node) is less than or equal to the transmission range (*R*).

3.4.3 Transmission range

Transmission power level that each node has is assumed to be equal for all nodes in the network. Since it is the characteristic that is represented by a circle centered at each node, the transmission range is therefore the *R* mentioned in the previous subsection.

The wireless transmission range commonly used is between 100m – 250m. Sanchez *et al* [31] stated that the optimum transmission range in ad hoc wireless networks cannot be represented by a fixed number because there are many factors influence the transmission. Different fading mechanisms have been studied and different network operational conditions, such as atmospheric condition and man-made obstacles, are found to affect the transmission range.

3.5 The Code

The code that is written in Matlab that show how to determine the values of the proposed metrics can be seen in the Appendix.

Chapter 4

Simulation

4.1 Description

The simulations are written in Matlab. Matlab is chosen because the routing problem can be explained in a simple way given the vector and matrix oriented nature of Matlab. It allows us to express the existing multicast algorithms mathematically. It also gives us the freedom to vary any variables involved.

The Matlab program that is made as a tool to assess the performance of multicast algorithms has the following flexibility and limitation:

1. The input to a number of mobile nodes can be changed easily with a lower limit of three. This limitation exists due to the use of the easy bounding circle algorithm [27], which determines the radius of the network topology. The creation of a circle using this algorithm requires at least three nodes in the 2-D plane.
2. The representation of the transmission range of each node and other variables input can be changed to match user's requirements.

The goal of this simulation is to create and investigate a set of random network topologies, run different multicast routing algorithms in each topology, and evaluate their performance. Simulations are important to determine if the proposed metrics are effective in terms that they can actually be used to compare different routing algorithms.

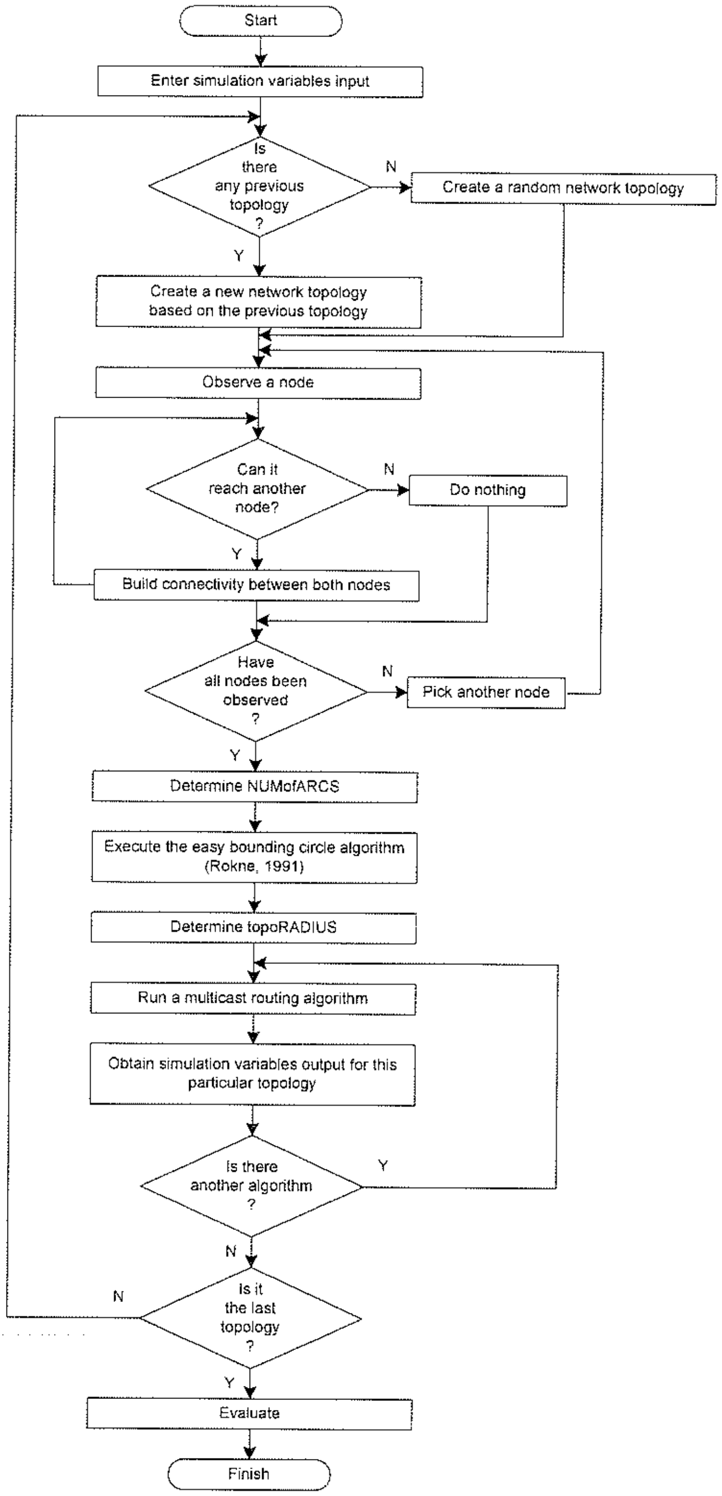
Start
Enter simulation variables input
Is there any previous topology ?
N
Create a random network topology
Y
Create a new network topology based on the previous topology
Observe a node
Can it reach another node?
N
Do nothing
Y
Build connectivity between both nodes
Have all nodes been observed ?
N
Pick another node
Y
Determine NUMofARCS
Execute the easy bounding circle algorithm (Rokne, 1991)
Determine topoRADIUS
Run a multicast routing algorithm
Obtain simulation variables output for this particular topology
Is there another algorithm ?
Y
N
Is it the last topology ?
N
Y
Evaluate
Finish

Figure 4.1: Flowchart of the simulation

The flowchart in Figure 4.1 illustrates briefly how the simulation works. Further details of the simulation procedure are explained below.

1) Simulation variables input is required.
2) From the number of nodes provided, a random network topology is created. At the beginning, which means when there is no previous topology, the random network topology is created from a set of random nodes that are uniformly distributed. Otherwise, the topology would still be created randomly but based on the position of the nodes in the previous topology. The reason behind this is that the assumption that any mobile node could only change the position within a limited distance in an interval of time. So the program would generate for each node a random distance and a random direction of movement from the node's location in the previous topology.
3) For each node, connectivity is built. When any pair of nodes is located within a distance that is based on the transmission range provided, they would be connected. Otherwise, do nothing.
4) The graph resulted from the previous step would be a direct variable input to calculate the average number of arcs per node. The larger the number is, the stronger the connectivity between the mobile nodes is.
5) The random position of each node would be a direct variable input to calculate the radius of this topology. The calculation is done repeatedly until one of the possible results exists. If there is any triangle with three acute angles can be formed from three of the nodes, the radius would be calculated based on these three nodes. Otherwise, the radius would be obtained from two of the nodes which are located farthest away

from each other. A detail description of this easy bounding circle algorithm used for determining the topology radius can be found in [27].

6) Each multicast algorithm would be applied to this topology. Additional information regarding the algorithms will be explained in Subsection 4.3. When the algorithm needs a source to initiate the session, the random input of the source is required.
7) After the application of each algorithm, the simulation variables output would be provided. The output includes the performance measurement of the algorithms in this particular topology. In a set of simulations, the performance is measured using the number of messages generated and the number of transmission loss in the multicast routing session. They will be defined at the end of this chapter.
8) The next simulation would run when the nodes are in different position. If the total number of topologies created up to this point is still less than the number of topology fixed as an input at the beginning of the simulation, go back to step 2 of this procedure. Otherwise, go to the next step.
9) Graphs are created based on all the final variables obtained until the previous step and observe them to evaluate the performance of the algorithms.

4.2 Simulation Variables Input

Table 4.1 summarizes the variables used as input for a set of simulations of running three multicast routing algorithms.

At the beginning of the simulation, a source node is randomly selected. This selection is important for two of the three algorithms, which are AMRIS and ODMRP, to initiate a multicast session. The assumption is that during every period of time, the same node has messages to send to other nodes. Every new topology would indicate the position of all nodes at the beginning of the observed time.

Variable	**Value**
Number of nodes	10
Number of random topologies	50
Transmission range	250m
Initial network area	500mx500m
Maximum network area	1000mx1000m
Maximum distance of a node's movement	150m (per time unit)
Number of sender	1

Table 4.1: Simulation variables input

Compared to the density metric proposed by Jacob [14], the metrics proposed here would vary automatically as the topology changes. We use 250m as the transmission range since it is typical for mobile devices. The scenario area of the simulation is permitted to expand according to the random topology generator during a set of simulations. This expansion needs to be considered so as not to limit the random movement of nodes. However, maximum network area is limited to 1000mx1000m to reduce the likelihood of no connectivity at all when the nodes are allowed to go further than that. This consideration is based on the number of nodes that is 10. The node position affects both connectivity and radius of the topology.

Radio irregularity factors are not considered in these simulations. Therefore, if node *i* can send a message to node *j*, node *j* can also send a message to node *i*.

4.3 Selected Algorithms in Simulation Environment

The description of each algorithm chosen to be compared has been discussed in Subchapter 2.3. The following is the additional information of how each algorithm is applied in the simulations.

4.3.1 AMRIS simulations

All nodes in the network share a multicast tree and each node is dynamically assigned an ID number called an msm-id. The multicast session is initiated by the node that has messages to send. This node is called Sid and it is chosen randomly. The msm-id for Sid could be 0 or 1, depending on what is preferred as the lowest number. The tree is built and maintained by the use of msm-ids that are generated randomly in sequence.

In the first topology, Sid executes the Tree Initialization procedure to build a shared multicast tree among the nodes connected to Sid. At the beginning of the rest of the network topologies, each node uses a beaconing mechanism to find out about the neighbours' position. If any of the nodes detects any link failure, where there are disconnected pair of nodes, Tree Maintenance procedure is executed. Any detached nodes would attempt to rejoin the tree.

4.3.2 ODMRP simulations

The multicast session is also initiated by the node that has messages to send. This node is also chosen randomly but the node must be identical with Sid in AMRIS. This is important for a consistency purpose. The nodes that want to receive the messages also broadcast their messages to make sure that the messages are not lost.

At the beginning of each topology, the source floods the network with JOIN DATA messages in order to build a mesh for the network.

4.3.3 AMRoute simulations

Since this protocol is proactive, periodic join request is broadcast to the entire network by each node. The building of the tree is initiated by the node that is called the Core. IP address for each node is randomly assigned.

At the beginning of each topology, each node in the network sees itself as a core until the Core is determined. The determination of the Core is based on the highest IP address of all the nodes.

4.4 Performance Measurement

While running the algorithms on different network topologies, the same measurement is needed for the performance. The performance of each algorithm can be measured by:

1. Number of messages involved in one multicast routing session.

This represents the number of packets sent from initiating a multicast session to the establishment of the routes from the sender to the receivers. The value of this performance metric is obtained from the total number of directed arcs emanate and enter all the nodes in the network. A lower number indicates that network bandwidth is well-preserved.

2. The number of transmission loss.

   A transmission loss is defined as the number of intended destination nodes that fail to receive the multicast data sent by the source. This number should be minimized to study the effectiveness of the multicast routing algorithm in any topology. The number of transmission loss is used instead of the delivery ratio because the delivery ratio does not really show whether there are intended destinations that do not receive messages. Transmission loss used here enables easy observation of topological behaviour and how the arrangement of the nodes affects data forwarding.

Chapter 5

Evaluation of the Performance Comparison

5.1 Evaluation

Table A (Appendix) presents the simulation variables output that are a result of 50 simulations. This section contains with the explanation of graphs that represent the performance of each multicast algorithm as a whole.

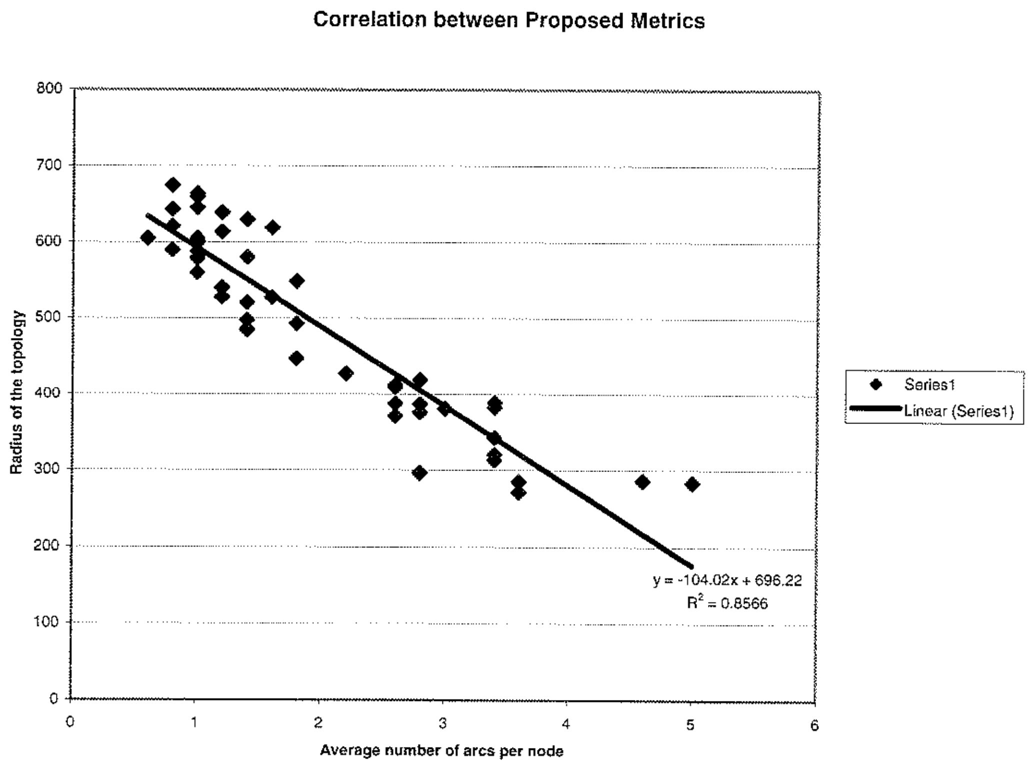

Figure 5.1: Correlation between the average number of arcs and radius of the topology

Since many variables exist, the performance of the multicast routing algorithms can be evaluated from different points of view. Therefore, it is important to determine if there is any connection between the proposed metrics shown in Figure 5.1.

The correlation coefficient between the metrics is 0.925527 ($=\sqrt{0.8566}$). This value indicates that we can use a single metric instead of both in evaluating the multicast algorithms performance. Either metric can meet the need of interpreting the result because the other metric would show virtually the same information. The average number of arcs is used as the x-axis throughout this analysis. However, a combination of both metrics might be useful when there is a need to emphasize one metric over the other.

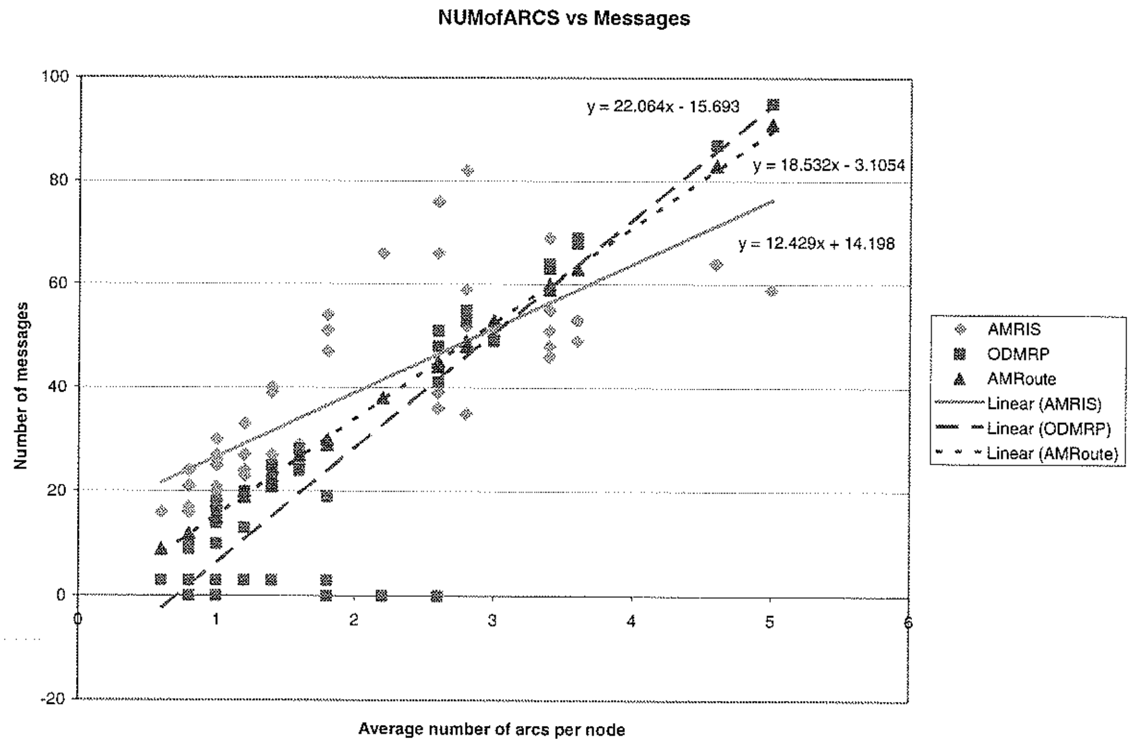

Figure 5.2: Number of messages of all three algorithms

Figure 5.2 is the result of plotting the number of messages of all three algorithms in one graph. For each set of data, which represents the number of messages for each algorithm, we draw a regression line. These lines can be used when comparing the algorithms.

With three regression lines across all sets of data in Figure 5.2, there are a small set of ODMRP plots on the bottom left of the lines and a small set of AMRIS plots on the top left of the lines. Since they can mean something, it will be discussed afterward.

As seen in Figure 5.2, the lines intersect with each other around the same point. From the regression equations, the exact intersection between each pair of the regression lines can be found as shown in Table 5.1 below.

Intersection Point	**AMRIS**	**ODMRP**	**AMRoute**
AMRIS	-	(3.102,52.757)	(2.835,49.435)
ODMRP	(3.102,52.757)	-	(3.564,62.947)
AMRoute	(2.835,49.435)	(3.564,62.947)	-

Table 5.1: Intersection points between regression lines of the algorithms

What is expected from multicast routing is as few messages generated as possible. From this point of view and the regression lines, the graph shows that from low to medium connectivity level (represented by NUMofARCS ranging from 0.6 to around 3), ODMRP has better behaviour than AMRoute, which has better behaviour than AMRIS. However, from medium to high connectivity level, the opposite behaviour is noticeable. ODMRP behaves the worst while AMRIS show fewer messages generated. AMRoute seems stable by generating a number of messages that is between the number of messages generated by the other two algorithms. So when the nodes are in situations where connectivity level is low, ODMRP is more preferred than AMRIS; while in other situations, AMRIS is more preferred. However, AMRoute could be a safer choice because the highest number of messages generated by this algorithm never exceeds the highest number of messages generated by the other two.

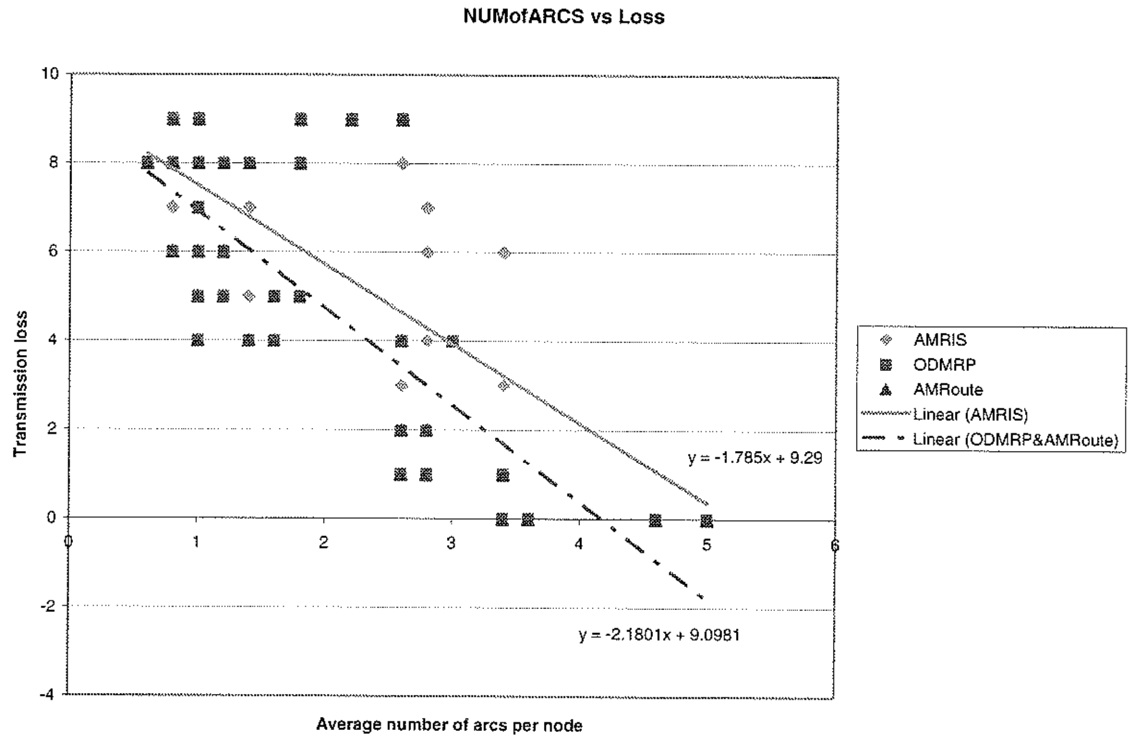

Figure 5.3: Transmission loss of all three algorithms

The graph in Figure 5.3 shows the transmission loss for each multicast routing algorithm. There seems to be only two instead of three sets of data in Figure 5.3. This occurs because the data set of ODMRP overlaps the data set of AMRoute. Recall that in Chapter

4 the transmission loss is defined as the number of destination nodes that fail to receive multicast data sent by the source.

The transmission loss is also expected to be as low as possible. From the regression lines drawn for both sets of data in Figure 5.3, it is obvious that ODMRP and AMRoute are more preferred if it based on the transmission loss.

But since the straight line drawn for each set of data might not actually be the best fit for each set of the data, the performance of the algorithms is going to be analyzed separately. Individual analysis is useful for studying in more detail how the algorithm performs based on the proposed metrics.

Each of the three following subsections is a study of an individual multicast algorithm.

5.1.1 AMRIS

In analyzing the graphs, a good fit between any pair of parameters is tried to be found by examining the scatter plots of data. This is an important step in evaluating the correlation which is needed for explaining the behaviours of the algorithm so that they can be used in the future for improving the performance of the algorithm.

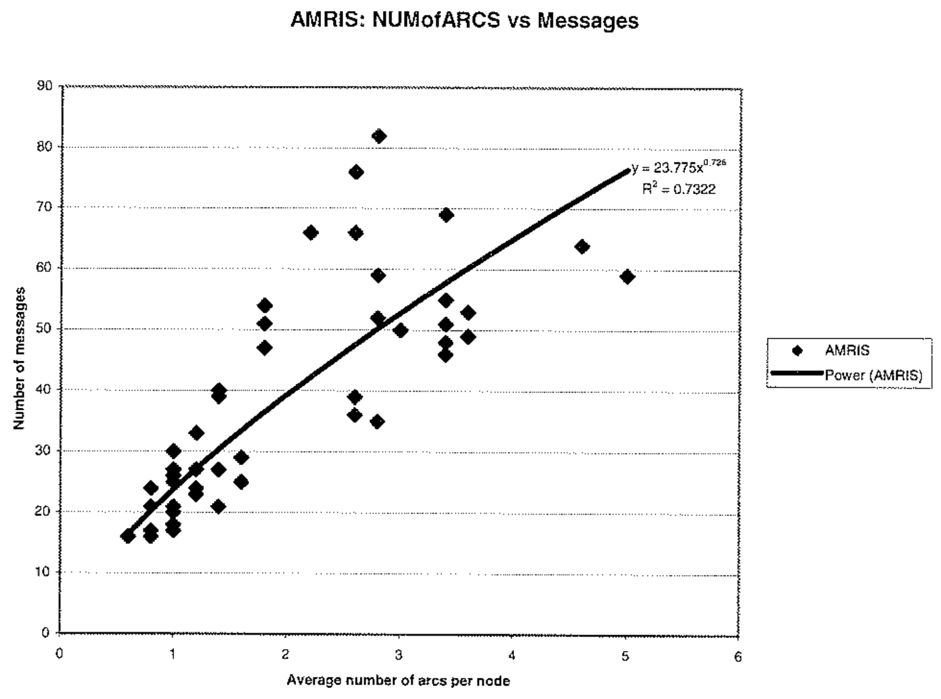

Figure 5.4: Number of messages of AMRIS

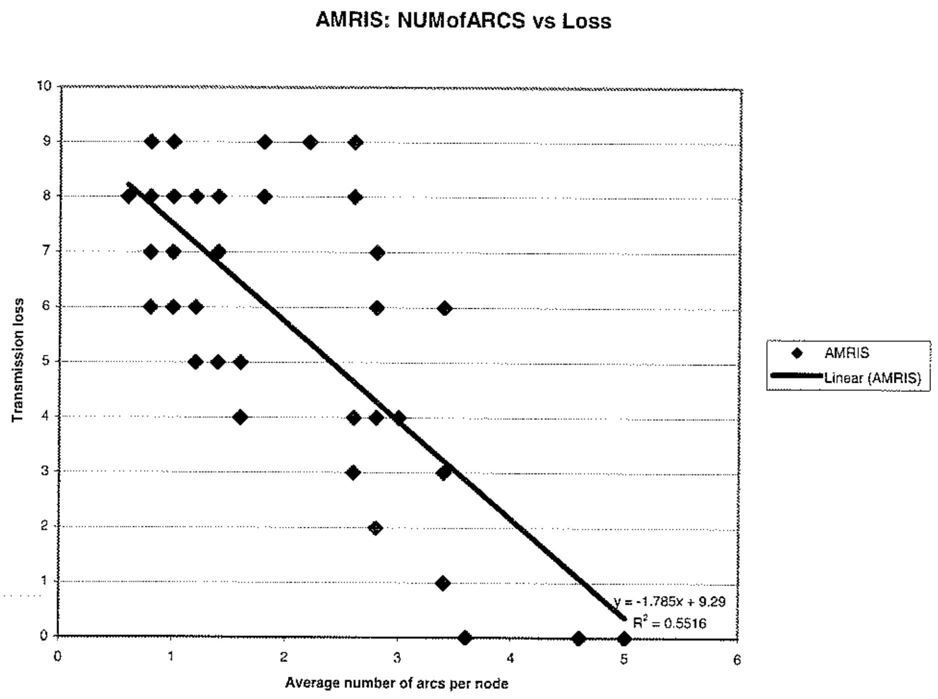

Figure 5.5: Transmission loss of AMRIS

So after trying to fit the data in Figure 5.4 with a linear function and different nonlinear functions, the best fit is a power function with coefficient of determination of 0.7322. The same thing is done for data in Figure 5.5 and a straight line is the best fit with coefficient of determination of 0.5516. The weak relationship between these pairs of parameters is intelligible because the plots are very scattered which show that the performance of AMRIS is very unpredictable.

However, there is a need to find any relationship between the number of messages generated in AMRIS and the number of transmission loss. And when we look closely and match the plots on Figure 5.4 with the plots on Figure 5.5, we can actually divide the set of AMRIS data into two groups. The smaller group represents the distinctive performance of AMRIS in low to medium connectivity.

Figure 5.6 and Figure 5.7 show the distinctive performance of AMRIS. Group 2 represents the distinctive performance of AMRIS in low to medium connectivity network when it generates more messages and loses more transmission than Group 1. The grouping of the plots also gives us better results of data fitting.

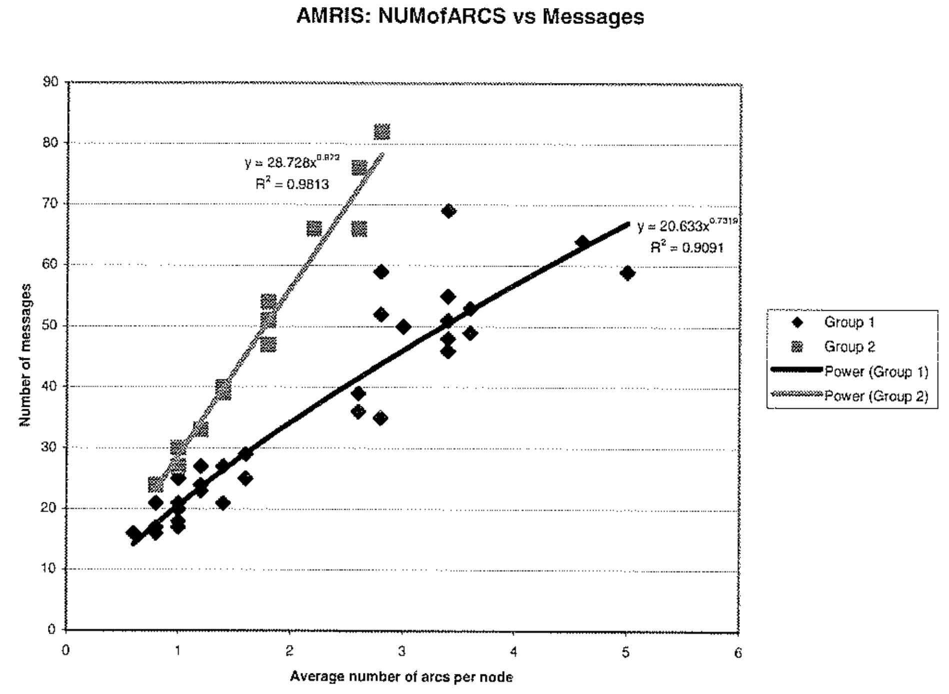

Figure 5.6: Number of messages of AMRIS (plots are divided into groups)

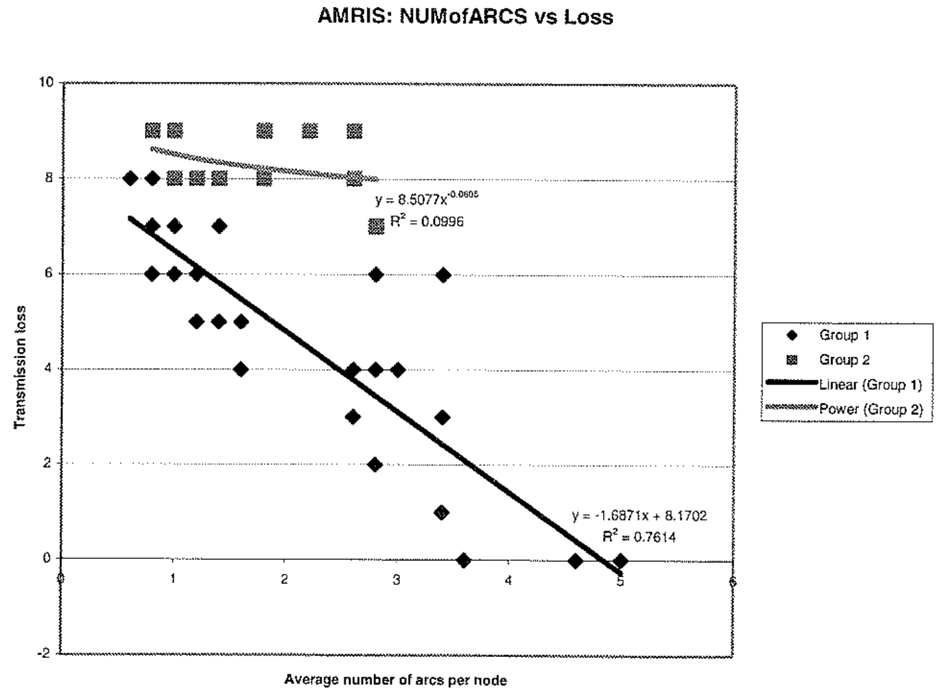

Figure 5.7: Transmission loss of AMRIS (plots are divided into groups)

5.1.1.1. NUMofARCS vs Number of messages

The performance of Group 1 and Group 2 is best described with a power function with coefficient of determination of 0.9091 and 0.9813, respectively. The area between the two regression lines show that AMRIS has high unpredictability when the network has medium connectivity of nodes. As the two lines go from low to higher connectivity, the area between the regression lines becomes broader. This shows that when the network has medium connectivity the algorithm has tendency to generate most of its highest number of messages.

Observing the performance of AMRIS specifically on this area, we find that the medium connectivity can be obtained from different scenarios of topology. The topologies where the source has no or very little connection with other nodes are the scenarios that generate the plots in Group 2. This occurs because the nodes are trying to rejoin the tree. This attempt is made difficult in this kind of scenarios because the source is some kind of *remote*; yet the source node is the root of the multicast tree. On the other hand, the topologies where the source still has some neighbouring nodes generate significantly fewer messages. The latter are the scenarios that generate the plots in the Group 1. The regression line of Group 1 is directed toward the plots when the network has strong connectivity. It happens because the nodes can find their potential parent nodes more easily due to the availability of more neighbouring nodes.

5.1.1.2. NUMofARCS vs Transmission loss

The performance of Group 1 is best described with a linear function with coefficient of determination of 0.7614, while Group 2 a power function with coefficient of determination of 0.9813. A straight line as the best fit for Group 1 is easily understood because as the network has stronger connectivity, the number of transmission loss becomes less. The unpredictability of AMRIS in losing transmission also occurs when the network has medium connectivity as shown in Figure 5.7.

5.1.2 ODMRP

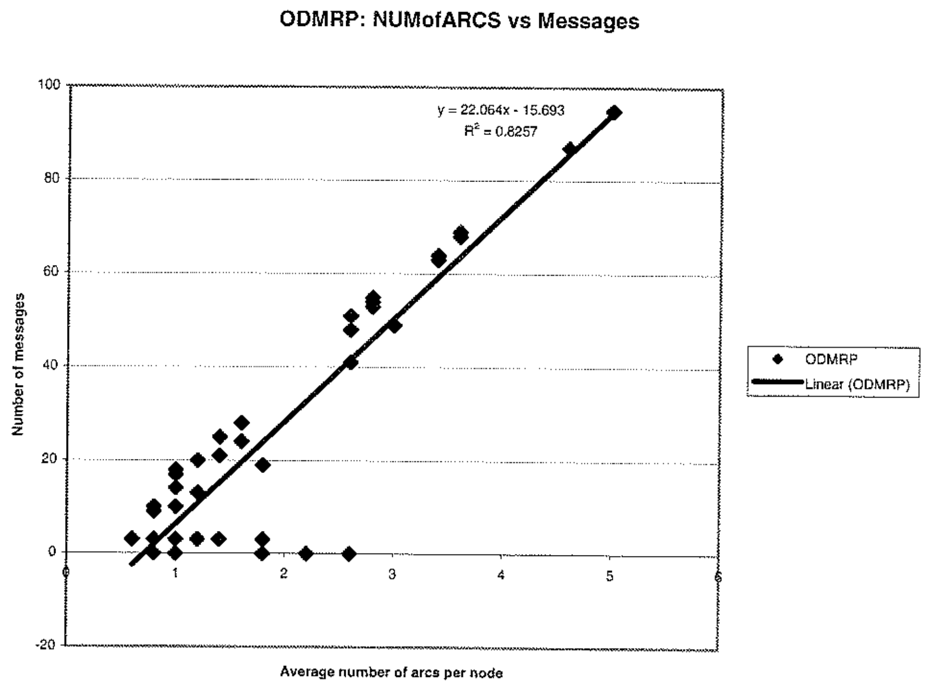

Figure 5.8: Number of messages of ODMRP

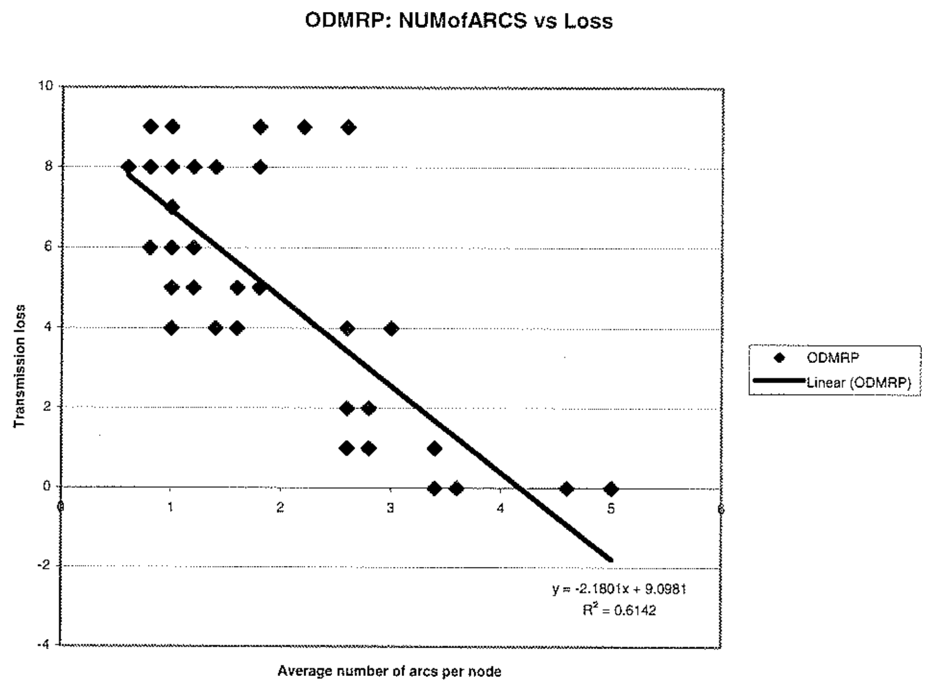

Figure 5.9: Transmission loss of ODMRP

After trying to fit the data in Figure 5.8 and Figure 5.9 with a linear function and different nonlinear functions, the best fit for both set of data is a straight line with coefficient of determination of 0.8257 and 0.6142, respectively. The better relationship between these pairs of ODMRP parameters than AMRIS parameters is intelligible because the plots are less scattered.

There is also a need to find the relationship between the number of messages generated in ODMRP and the number of transmission loss. So the plots in Figure 5.8 and Figure 5.9 are observed and they can be divided into two groups as shown in Figure 5.10 and Figure 5.11.

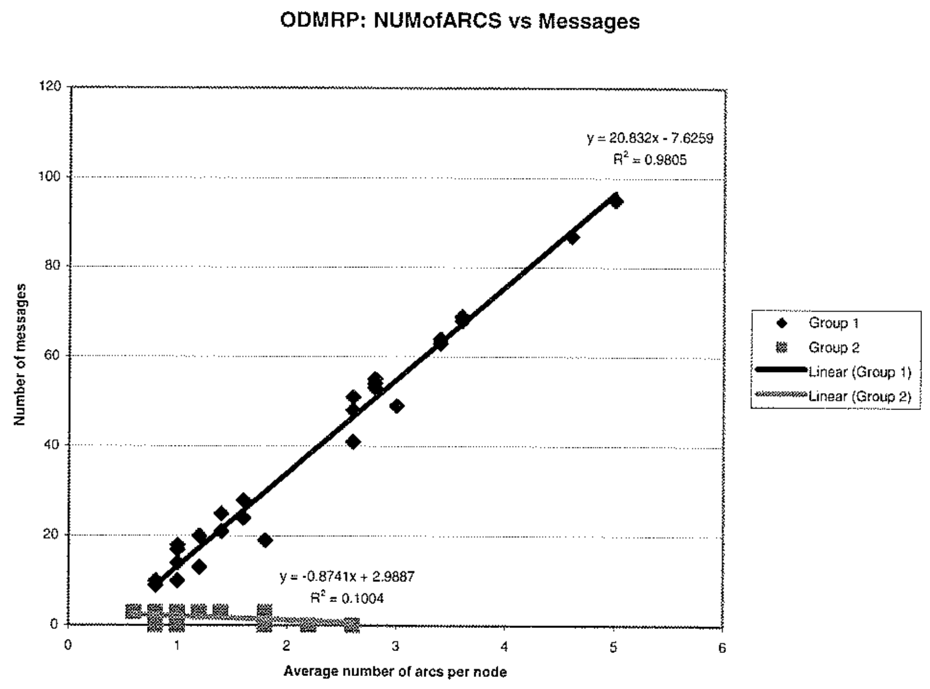

Figure 5.10: Number of messages of ODMRP (plots are divided into groups)

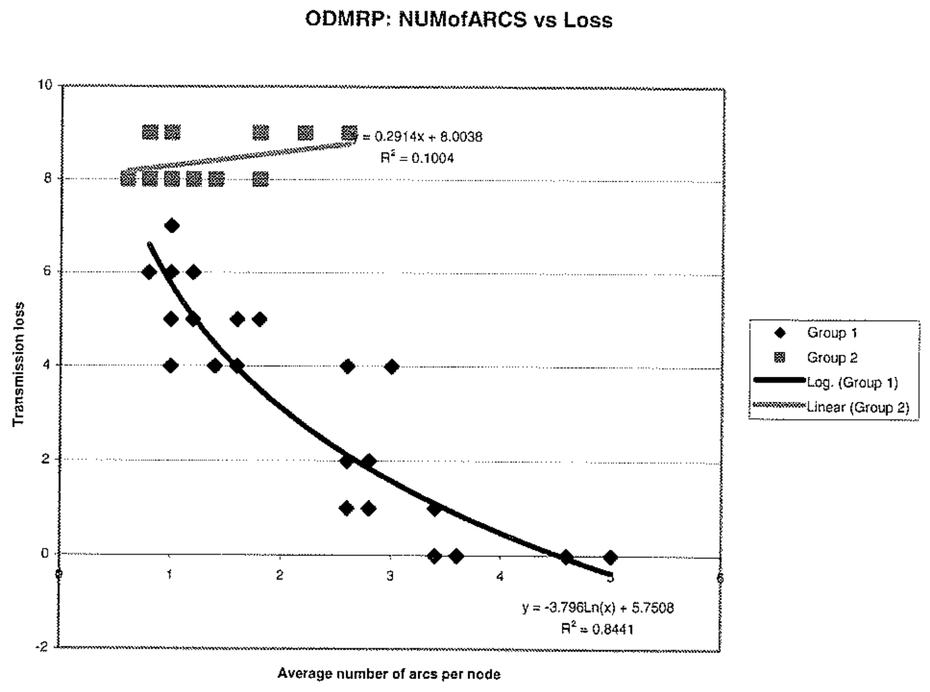

Figure 5.11: Transmission loss of ODMRP (plots are divided into groups)

As in AMRIS, Figure 5.10 and Figure 5.11 show that there is some patterns in the performance of ODMRP. Group 2 represents the distinctive performance of ODMRP in low to medium connectivity network when it loses more transmission than Group 1. But ODMRP shows a bit of opposite performance than AMRIS because Group 2 generates fewer messages than Group 1 in the similar range of connectivity. The grouping of the plots also gives us better results of data fitting.

5.1.2.1. NUMofARCS vs Number of messages

The performance of Group 1 and Group 2 is best described with a linear function with coefficient of determination of 0.9805 and 0.1004, respectively. Like AMRIS, there is also some unusualness from the plotted performance of ODMRP. As illustrated in Figure 5.8, Group 2 shows that ODMRP is quite unpredictable when the network has low to medium connectivity but Group 1 shows relatively constant performance.

The performance here is described as constant when the number of messages increases as the number of arcs increases. The constant part of ODMRP performance is revealed by the correlation coefficient between the average number of arcs and the number of ODMRP messages, which is 0.9902. Compared to AMRIS, Group 2 of ODMRP generates much less messages in the same connectivity level. As a performance metric, the less messages generated is the more preserved the bandwidth is and it is actually good for the network. However, it is important to discuss further the possible cause of this situation and its impact in subchapter 5.2.

5.1.2.2. NUMofARCS vs Transmission loss

The performance of Group 1 is best described with a logarithmic function with coefficient of determination of 0.8441, while Group 2 a linear function with coefficient of determination of 0.1004 (as in 5.1.1.1). A curve line as the best fit for Group 1 instead of a straight line as in AMRIS shows that ODMRP does not have as much transmission loss as AMRIS as the connectivity decreases. The coefficient of determination of ODMRP also represents less unpredictability of the transmission loss of ODMRP than AMRIS.

5.1.3 AMRoute

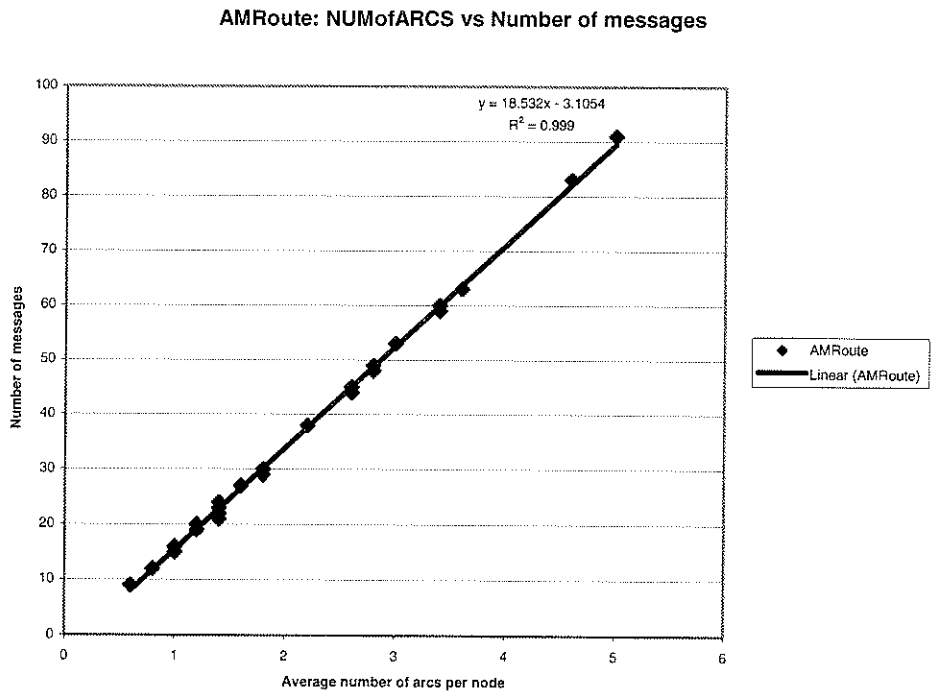

Figure 5.12: Number of messages of AMRoute

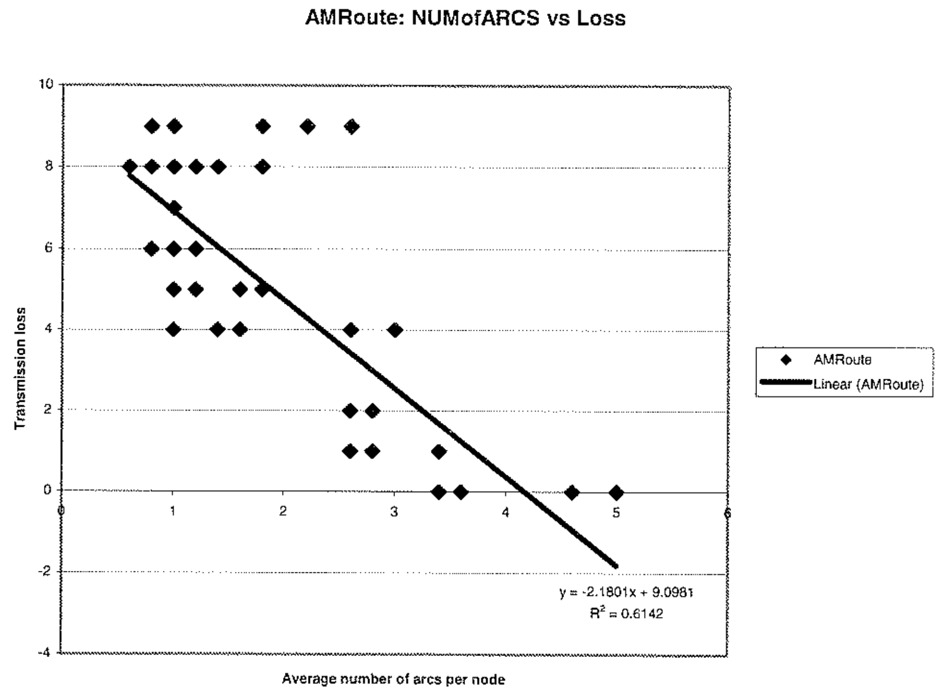

Figure 5.13: Transmission loss of AMRoute

Figure 5.12 shows that a straight line will fit the plots almost perfectly. It is also revealed by the correlation coefficient (*r*) of 0.9995 between the average number of arcs and the number of AMRoute messages. The best fit for Figure 5.13 is the same as Figure 5.9 which is a straight line with coefficient of determination of 0.6142.

There is no need in dividing the plots into groups as is done to the other two algorithms because there is nothing extraordinary about Figure 5.12 that can be related to Figure 5.13. But Figure 5.12 proves that the most distinctive feature about AMRoute is its predictability. The number of messages generated by this algorithm can be estimated in any level of connectivity.

5.2 Discussion

Overall, AMRoute shows a very predictable performance in creating multicast routes. The scatter plot in Figure 5.12 reveals a strong positive relationship between the average number of arcs per node and the number of messages generated by AMRoute. The result shows that the behaviour of AMRoute is easily predicted which means that the more connected the mobile nodes are (or the smaller the radius of the topology is) the more messages are exchanged between them.

AMRIS shows the worst performance in terms that its behaviour is very unpredictable when the network has low to medium connectivity. The use of this algorithm in this connectivity level would not allow any advanced prediction of how it will behave when the topology changes randomly. This was especially so when the radius of topology is small because it could be under or over predicted. But the grouping could help in anticipating the tendency of AMRIS of generating huge amount of messages and still losing many transmissions.

There are situations where AMRIS shows tendency of generating huge amount of messages while ODMRP shows tendency of generating very few and sometimes no messages at all. But both distinctive cases of the two algorithms result in similar behaviour of losing the transmission. AMRoute shows the most stable performance by generating messages within the least range among the three algorithms. The proposed metrics are therefore useful to predict the number of messages that could be generated in multicast routing sessions by any multicast algorithm. The metrics also identify when a distinctive performance of an algorithm should be anticipated.

The transmission loss would not be a behaviour that is as easily predicted as the number of messages from the chosen three algorithms. By observing the data only, AMRIS has the worst performance because it shows more loss than ODMRP and AMRoute. The mode of the transmission loss of the three algorithms is 8, but the median of AMRIS transmission loss is 7 while the other two is 6. Because of the mesh created by ODMRP and AMRoute (AMRoute create a mesh first before building the tree), the number of transmission loss for both algorithms is the same.

By grouping the plots whenever possible, there are some patterns might be found that can be used for predicting the performance of each algorithm. This kind of pattern helps the study of the algorithm so that the unusualness of performance and in what situations that unusualness occurs could be anticipated. It can also be used to understand the importance of the arrangement of the nodes in the network that builds the topology.

Since there are some good patterns exist between the average number of arcs and transmission loss, the proposed metrics might be suitable parameters to predict the transmission loss of the multicast algorithm performance. Even though the prediction of this transmission loss might not be as accurate as the prediction of the number of messages, they still enable an observation of how the algorithms perform in changing topologies.

Sometimes the final result is quite bias when the simulations are executed for many times. When the exchange of messages caused by each multicast algorithm in each of the fifty topologies is observed, there are some of the least messages resulted from ODMRP is caused by the occurrence of network partitions. The network partition is the reason why a distinctive line can be drawn across some ODMRP plots and put them into a group (Group 2) as shown in Figure 5.10. When the network is partitioned due the movement of the nodes, ODMRP would have the source to be able to flood only the subset of the network where the source is located. The more the network is partitioned, the more ODMRP is biased toward the number of messages involved in the related topology. For example, once the network is partitioned such that the source is only connected with only one other node. The exchange of messages during the multicast session in this particular topology occurs only between those two nodes. The rest of the nodes are inactive because the activities should only be initiated by the source.

On the other hand, even when the network is divided into many sub networks, AMRIS and AMRoute would trigger each node to try to connect itself with any available neighbouring nodes. AMRIS triggers the detached nodes to find a way to rejoin the tree and does nothing when the tree cannot be reconfigured. AMRoute triggers each node to create a mesh with its surrounding nodes in order to create any possible tree from that mesh. For this reason, if there are multiple senders during the multicast session instead of a single sender used in this specific simulation, AMRoute could show the best performance because each node keeps attempting to create any possible tree with its neighbours and each sender can at least send its messages to any nodes that are attached to it.

AMRIS involves too many procedures in its attempt to maintain the tree (Recall that it uses different kinds of messages: NEW-SESSION, JOIN-REQ, JOIN-ACK, JOIN-NACK, JOIN-CONF, and beaconing). The biggest problem in AMRIS is its idea of using the msm-id in order to create and reconfigure the tree. Most of the time, the detached nodes are still connected to any nodes that are already on the tree but the latter cannot allow the former to rejoin the tree because the msm-id of the former nodes are smaller than the latter's. This causes AMRIS to simply have more transmission loss than the other algorithms. AMRIS should let the detached nodes to modify its msm-id when it cannot find any "feasible" parent after all attempts (Branch Reconstruction 1 and 2). This might help reduce the magnitude of the top curve of the unpredictable area of AMRIS performance.

However, when the network has strong connectivity and the change in the topology radius is small, AMRIS could show the best performance because it does not use flooding as its basic concept. This situation will lead to the movement of the nodes to be always around the previous position and the Tree Maintenance procedure would only need any detached node to send JOIN-REQ message and receive a JOIN-ACK as a permission of rejoining the tree.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

The proposed metrics are introduced as a new approach for comparing different routing protocols by exploiting the characteristics of the network itself. The average number of arcs per node represents the connectivity level of the nodes in the network and the larger the number is, the stronger the connectivity becomes. This metric is shown to have strong correlation with the other metric, which is the radius of the topology. As can be predicted, when the nodes in the network have uniform transmission range, the larger the network area is, the lower the connectivity is.

These metrics are needed to provide a new method for assessing the performance of multicast routing algorithms. Most of the work in multicast has focused on the introduction of new algorithms. There is lack of work on developing methods to compare these algorithms properly.

Since our metrics change as the topology changes randomly, it can represent a real wireless network where the nodes are usually mobile in random directions. Furthermore, there is no need in maintaining a good simulation area so that the metrics can provide illustrative values. This becomes the basis of our set of simulations and it can be used to measure the quality and performance of any multicast routing algorithms.

Of the three algorithms that are selected to be compared, AMRoute shows the most stable performance compared to AMRIS and ODMRP. When AMRoute is evaluated by itself, it appears to be the only algorithm that shows very strong correlation between the average number of arcs and the number of messages and therefore it is predictable.

AMRIS and ODMRP show opposite performance by taking turn in generating large amount of messages over the other and over AMRoute. If evaluated separately, AMRIS and ODMRP show some distinctive patterns while performing in low to medium connectivity. The distinctiveness of each of the two algorithms is measured in both the number of messages and the transmission loss.

Even though the result presented here comes from only one set of simulations, there are actually several other sets of simulations executed and they result in similar patterns for each multicast routing algorithm. And so from the graphs that are the results of simulations, the proposed metrics are not only useful in comparing the algorithms but also in stimulating the appearance of unusual patterns of the algorithms by evaluating the algorithm separately. A distinctive performance of an algorithm can help in studying in details the reason why it behaves the way it does. This study is very important for different reasons such as to recognize the predictability of an algorithm, to anticipate the performance of the algorithm in different situations, and most importantly, to improve the algorithm.

6.2 Future Directions

The major thing that should be focused on in the future is to include more metrics in our method. The use of more metrics can be helpful in conducting more thorough assessment. It might also stimulate the emergence of different patterns in the performance of multicast algorithms that would lead to more detail analysis of the algorithm.

It would be very useful to consider real-life scenario such as various frequency of communications between the source and the destinations. This is because the source possibly communicates more frequently with a subset of its destinations and this could affect the whole performance of the algorithms.

Since this comparison study involves only three different multicast routing algorithms, more simulations to measure the performance of other algorithms are encouraged.

And finally, since there now exists a method that can be used to measure and study the multicast routing algorithms with their own patterns, the future work is to use this tool to improve the existing algorithms and to develop new ones.

References

[1] Agrawal, D. P., & Zeng, Q-A. (2003). *Introduction to Wireless and Mobile Systems*. Brooks/Cole, 2003.

[2] Bae, S. H., Lee, S.-J., Su, W., & Gerla, M. (2000, January/February). The design, implementation, and performance evaluation of the on-demand multicast routing protocol in multihop wireless networks. *IEEE Network*, 70-77.

[3] Bhattacharya, R., & Ephremides, A. (1997). Multicast routing and resource allocation in a mobile wireless network like the digital battlefield. *Center for Sattelite and Hybrid Communication Networks* (CSHCN T.R. 97-16 (ISR T.R. 97-48)).

[4] Bhattacharya, R., & Ephremides, A. (1998). A distributed multicast routing protocol for ad-hoc (flat) mobile wireless networks. *ACM/Baltzer Journal of Cluster Computing: Special Issue on Mobile Computing*, **1**(2).

[5] Bommaiah, E., Liu, M., McAuley, A., & Tapalde, R. (1998, August). AMRoute: ad-hoc multicast routing protocol. *Internet-Draft*, draft-talpade-manet-amroute-00.txt, August 1998, Work in progress.

[6] Chiang, C.-C., Gerla, M., & Zhang, L. (1998). Adaptive shared tree multicast in mobile wireless networks. *Proc. of IEEE Globecom'98*, 1817-1822.

[7] Chiang, C.-C., Gerla, M., & Zhang, L. (1998). Forwarding group multicast protocol (FGMP) for multihop, mobile wireless networks. *Cluster Computing*, 1, 187-196.

[8] Corson, M. S., & Batsell, S. G. (1995). A reservation-based multicast (RBM) routing protocol for mobile networks: initial route construction phase. *ACM/Baltzer Wireless Networks*, 1(4), 427-450.

[9] de Morais Cordeiro, C., Gossain, H., & Agrawal, D. P. (2003). Multicast over wireless mobile ad hoc networks: present and future directions. *IEEE Network*, 17 (1), 52-59.

[10] Detti, A., Loreti, C., & Loreti, P. (2004). Effectiveness of overlay multicasting in mobile ad-hoc network. *2004 IEEE International Conference on Communications*, 7, 3891-3895.

[11] Garcia-Luna-Aceves, J. J., & Madruga, E. L. (1999, March). A multicast routing protocol for ad-hoc networks. *Proc. IEEE INFOCOM'99*, New York, NY, 784-792.

[12] Gossain, H., de Morais Cordeiro, C., & Agrawal, D. P. (2002, June). Multicast: wired to wireless. *IEEE Communication Magazine*, 116-123.

[13] Ho, C., Obraczka, K., Tsudik, G., & Viswanath, K. (1999). Flooding for reliable multicast in multi-hop ad hoc networks. *Proc. of MOBICOM'99.*

[14] Jacobson, A. (2000). Metrics in ad hoc networks. Master's thesis, Lulea University of Technology, Lulea, Sweden.

[15] Ji, L., & Corson, M. S. (1998). A lightweight adaptive multicast algorithm. *Proc. of Globecom'98*, 1036-1042.

[16] Jetcheva, J. G., & Johnson, D. B. (2001, October). Adaptive deman-driven multicast routing in multi-hop wireless ad hoc networks. *Proc. of the 2001 ACM International Symposium on Mobile ad hoc networking and computing*, 33-44.

[17] Kurose, J. F., & Ross, K. W. (2001). *Computer Networking: A Top-Down Approach Featuring in Internet.* Addison-Wesley, 2001.

[18] Lee, S.-J., Su, W., & Gerla, M. (2002). On-demand multicast routing protocol in multihop wireless mobile networks. *Mobile Networks and Applications*, 7, 441-453.

[19] Lee, S.-J., Su, W., Hsu, J., Gerla, M., & Bagrodia, R. (2002). A performance comparison study of ad hoc wireless multicast protocols. *IEEE INFOCOM 2000*, 565-574.

[20] Liao, W., & Jiang, M-Y. (2003). Family ACK Tree (FAT): supporting reliable multicast in mobile ad hoc networks. *IEEE Transactions on Vehicular Technology*, 52(6), 1675-1685.

[21] Malloy, A. D., Varshney, U., & Snow, A. P. (2002). Supporting mobile commerce applications using dependable wireless networks. *Mobile Networks and Applications*, 7, 225-234.

[22] Mhatre, V., & Rosenberg, C. (2004). Design guidelines for wireless sensor networks: communication, clustering and aggregation. *Ad Hoc Networks*, 2, 45-63.

[23] Moustafa, H., & Labiod, H. (2003). A performance comparison of multicast routing protocols in ad hoc networks. *Proc. of the 14th IEEE International Symposiums on Personal, Indoor and Mobile Radio Communications, PIMRC 2003*, 1, 497-501.

[24] Papavassiliou, S., & An, B. (2002). Supporting multicasting in mobile ad-hoc wireless networks: issues, challenges, and current protocols. *Wireless Communications and Mobile Computing*, 2(2), 115-130.

[25] Ramanathan, R., & Redi, J. (2002). A brief overview of ad hoc networks: challenges and directions. *IEEE Communication Magazine*, 50th Anniversary Commemorative Issue, 20-22.

[26] RFC 2501, "*Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*", S. Corson and J. Macker, January 1999.

[27] Rokne, J. (1991). An easy bounding circle. From *The Graphic Gems II*. James Arvo (Editor), 1994.

[28] Royer, E. M., & Perkins, C. E. (1999). Multicast operation of the ad-hoc on-demand distance vector routing protocol. *Proc. of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 207-218.

[29] Royer, E. M., & Toh, C.-K. (1999). A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, 6(2), 46-55.

[30] Sahasrabuddhe, L. H., & Mukherjee, B. (2000, January/February). Multicast routing algorithms and protocols: a tutorial. *IEEE Network*, 14, 90-102.

[31] Sanchez, M., Manzoni, P., & Haas, Z. J. (1999). Determination of critical transmission range in ad-hoc networks. *Proc. of Multiaccess Mobility and Teletraffic for Wireless Communications'99*. Venice, Italy, October 1999.

[32] Sobeih, A., Baraka, H., & Fahmy, A. (2004). ReMHoc: a reliable multicast protocol for wireless mobile multihop ad hoc networks. *First IEEE Consumer Communications and Networking Conference, CCNC 2004*, 146-151

[33] Toh, C.-K., & Bunchua, S. (2001). Ad hoc multicast routing using the concept of long-lived routes. *Wireless Communications and Mobile Computing*, 1, 361-379.

[34] Varshney, U. (2002, December). Multicast over wireless networks. *Communications of the ACM*, 45(12), 31-37.

[35] Varshney, U. (2002, February). Multicast support in mobile commerce applications. *Computer*, 35, 115-117.

[36] Varshney, U., & Vetter, R. (2002). Mobile commerce: framework, applications and networking support. *Mobile Networks and Applications*, 7, 185-198.

[37] Varshney, U., Vetter, R. J., & Kalakota, R. (2000, October). Mobile commerce: a new frontier. *Computer*, 33, 32-38.

[38] Wang, B., & Hou, J. C. (2000, January/February). Multicast routing and its QoS extension: problems, algorithms, and protocols. *IEEE Network*, 22-35.

[39] Wu, C. W., & Tay, Y. C. (1999). AMRIS: a multicast protocol for ad-hoc wireless network. *Proc. of MILCOM'99*, 1, 25-29.

[40] Z. J. Haas, Sanchez, M., & Manzone, P. (1999). Determination of critical transmission range in ad hoc networks. *Multiaccess Mobility and Teletraffic for Wireless Communications 1999 Workshop (MMT'99)*, October 1999.

Appendix

Table A
Simulation Variables Output

Topology	NUMofARCS	topoRADIUS	V1	V2	V3	V4	V5	V6
1	4.6	286.172671	64	0	87	0	83	0
2	5	283.833839	59	0	95	0	91	0
3	3.6	271.073729	53	0	68	0	63	0
4	3.4	320.406079	46	1	63	0	59	0
5	3.6	284.903309	49	0	69	0	63	0
6	2.8	296.504453	35	2	53	2	49	2
7	3.4	313.133712	55	6	64	1	60	1
8	3.4	342.530186	69	3	64	1	60	1
9	3	380.411245	50	4	49	4	53	4
10	2.6	412.011919	39	4	41	4	45	4
11	3.4	388.48864	51	3	63	1	60	1
12	2.6	370.941819	36	3	48	2	45	2
13	3.4	382.27491	48	3	63	1	60	1
14	2.8	375.398144	52	6	54	2	49	2
15	2.8	386.256873	59	4	53	1	48	1
16	2.6	408.805042	66	9	0	9	45	9
17	2.8	418.827484	82	7	55	1	48	1
18	2.2	427.058586	66	9	0	9	38	9
19	2.6	387.145868	76	8	51	1	44	1
20	1.8	446.850381	47	8	19	5	29	5
21	1.8	492.698882	54	9	0	9	30	9
22	1.4	484.224656	40	8	3	8	23	8
23	1.4	485.639483	39	8	3	8	24	8
24	1.4	497.503411	39	8	3	8	23	8
25	1.2	527.363629	33	8	3	8	19	8
26	1.8	548.27918	51	8	3	8	30	8
27	1.4	580.288274	39	8	3	8	21	8
28	1	588.085938	25	7	10	6	15	6
29	1	601.245343	27	8	3	8	15	8
30	0.8	589.871037	21	8	3	8	12	8
31	0.8	621.007818	24	9	0	9	12	9
32	0.6	604.96604	16	8	3	8	9	8
33	1	599.541211	30	9	0	9	15	9
34	1	579.834867	30	9	0	9	15	9
35	1	600.666675	26	7	10	7	16	7
36	1.2	613.091869	24	6	13	6	19	6
37	1.2	638.854399	23	6	13	6	19	6
38	1	605.077966	20	7	10	7	16	7
39	1	645.044274	20	6	17	4	15	4
40	0.8	643.63268	17	7	10	6	12	6
41	0.8	674.325007	16	6	9	6	12	6
42	1	663.360411	18	7	14	6	16	6
43	1	659.237101	21	6	14	5	15	5
44	1.4	629.619757	27	5	21	4	22	4
45	1.6	618.807385	29	5	24	5	27	5
46	1	578.612782	17	6	18	5	16	5
47	1	559.52682	17	6	18	5	16	5
48	1.2	540.192993	27	5	20	5	20	5
49	1.6	526.9089	25	4	28	4	27	4
50	1.4	520.160395	21	7	25	4	23	4

Note:
V1, V3, V5 = Number of messages of AMRIS, ODMRP, AMRoute, respectively
V2, V4, V6 = Transmission loss of AMRIS, ODMRP, AMRoute, respectively

```
% AMRFIG is part of the AMRIS algorithm
%       It is a short version for display purpose only
%       It is needed by AMRTRYOUT.m
%       It needs GENERATEIDNEW.m in order to run properly
%
% Author: A. Radyastuti
% v6.5 26-Jan-04

%=================================
% create the graph
%=================================
figure
for i = 1:n
    plot(x(i),y(i),'o','MarkerEdgeColor','k',...
        'MarkerFaceColor','c','MarkerSize',15);
    str=num2str(i);
    text((x(i)-0.5),y(i),str);
    ti = procedure;
    title(['Random Points in Topology ',num2str(ti)]);
    axis equal;
    grid;
    hold on
end
grid on
hold on
sen=sender;
plot(x(sen),y(sen),'o','MarkerEdgeColor','k',...
       'MarkerFaceColor','g','MarkerSize',15);
str=num2str(sen);
text((x(sen)-1),y(sen),str);
hold on
for j=1:n
    v=x(j);
    w=y(j);
    for k=(j+1):n
        d(j,k)=sqrt((x(k)-v)^2+(y(k)-w)^2);
        d(k,j)=d(j,k);
        if (d(j,k)<=trans_range)
            line([v x(k)],[w y(k)])
            adjacency_matrix(j,k)=1;
            adjacency_matrix(k,j)=adjacency_matrix(j,k);
        else
            adjacency_matrix(j,k)=0;
            adjacency_matrix(k,j)=adjacency_matrix(j,k);
        end
    end
end
hold on
d;
adjacency_matrix;
a = adjacency_matrix;

%=================================
% find all connected nodes
%=================================
ii=sender;
NODE_NUMBER=sender
adjm=adjacency_matrix;
connect % call connect.m
pause(1)
% end of finding all connected nodes

%=================================
% broadcast NEW_SESSION message
%=================================
sender;
connection;
Parent=[];
if (sender<=n)
    msm_sender=1;
end
msm_i=msm_sender;
nodes=sender;
nodesparent=0;
Allmsm_i=msm_i;
```

```
Parent=sender;
exParent=[];
AllChild=[];
NEW_SESSION=0;
warn=0;
adjacency_matrix;
connection;
while (length(Parent)~=0)   % when we still have a Parent
    for pr=1:length(Parent)
        Parent;
        jnow = Parent(pr);

        PR=jnow;
        Parent;
        exParent;
        pause(0.1)
        if (find(exParent==PR))
            Parent;
            pr;
            warn=1;
            Parent(pr)=[];
            pause(1)
            break
        end
        Children=[];
        connection(find(connection==PR))=[];
        for k=1:n   % finding the Children
            if (adjacency_matrix(jnow,k)==0)
                Children(k)=0;  % it is not connected or itself
            elseif (adjacency_matrix(jnow,k)==1)
                Children(k)=k;
                if (length(connection)~=0)
                    ff3= find(connection==k);
                    if (ff3)
                        Children(k)=k;  % it is still available
                    else
                        Children(k)=0;
                    end
                else
                    Children(k)=0;  % no node is available
                end
                if (length(exParent)~=0)
                    exParent;
                    ff= find(exParent==k);
                    if (ff)
                        Children(k)=0;  % it is in the exParent
                    end
                end
            end
        end
        Children;
        Children(find(Children==0))=[];
        Children;
        pause(0.1)

        %===================================
        % generate msm_id for each node
        %===================================
        generateidnew  % call generateidnew.m

        for kk=1:n
            if (adjacency_matrix(jnow,kk)==0)
                bro_child(kk)=0;
            else
                bro_child(kk)=kk;
            end
        end
        bro_child(find(bro_child==0))=[];
        Children;
        for bcc=1:length(bro_child)
            if (find(Children==bro_child(bcc)))
                last_ep(bcc)=0;
            else
                last_ep(bcc)=bro_child(bcc);
            end
```

```
end
last_ep(find(last_ep==0))=[];

if (length(last_ep)~=0)
    for bc=1:length(last_ep)
       plot_arrow(x(PR),y(PR),x(last_ep(bc)),...
               y(last_ep(bc)),'headwidth',0.08,...
               'headheight',0.15,'color',[1 0 0],...
               'facecolor',[1 1 0]);
        NEW_SESSION=NEW_SESSION+1;
    end
end

bro_child=[];
last_ep=[];

pr;
PR;
exParent(length(exParent)+1)=PR;
pause(0.1)
% when the Parent has no Children
if ((length(Children)==0)&(pr==length(Parent)))
    Parent=[];
    AllChild;
    if ((length(AllChild)~=0)&(length(connection)~=0))
        AllChild(find(AllChild==PR))=[];
        AllChild;
        Parent=AllChild;
    else
        Parent=[];
    end
elseif (length(Children)~=0)   % when there are children
    while (length(Children)~=0)
        % distributing NEW_SESSION message to each node of the Children
        if (length(Children)~=0)
            for ch = 1: length(Children)
                xline=[x(PR) x(Children(ch))]';
                yline=[y(PR) y(Children(ch))]';
                   plot_arrow(x(PR),y(PR),x(Children(ch)),…
                              y(Children(ch)),'headwidth',0.08,...
                              'headheight',0.15,'color',...
                              [1 0 0],'facecolor',[1 0 0]);
                NEW_SESSION=NEW_SESSION+1;
                nodes(length(nodes)+1)=Children(ch);
                nodesparent(length(nodesparent)+1)=PR;
                hold on
            end
        end
        if (pr==1)
            AllChild=Children;
        else
            for ac=1:length(AllChild)
                Children(find(Children==AllChild(ac)))=[];
            end
            AllChild;

               AllChild(length(AllChild)+1:length(AllChild)+length(Children))=Child
               ren;
        end

        PR;
        % eliminating the Parent from the AllChild
        if (length(AllChild)~=0)
            AllChild(find(AllChild==PR))=[];
        end
        AllChild;

        % end of one set of Parent and determining the new Parent
        if (pr==length(Parent))
            Parent=AllChild;
            AllChild=[];
        end
        exParent;
        Parent;
        nodewith_id;
```

```
                msm_id;
                connection;
                Children=[];
                if (length(connection)==0)
                    pr;
                    if (pr==length(Parent))
                        Parent=[];
                    end
                    break
                end
            end
        end
    end
end
nodewith_id;
msm_id;
itsParent2;
nodes;
nodesparent;
hold on
pause(0.1)
```

```
% AMROUTETRY is AMROUTE run with metricsbreak3
%
% Author: A. Radyastuti
% v6.5 16-Jun-04
% v7.0 Last-Modified: 25-Jun-04

%===================================
% initialization
%===================================
all_node=1:n
TJR=0;  % TJR =Total_JOIN_REQ
TTC=0;  % TTC = Total_TREE_CREATE
TTCN=0; % TTCN = Total_TREE_CREATE_NAK
JOIN_REQ=0;
TREE_CREATE=0;
TREE_CREATE_NAK=0;
repeat=0;
messages_ar=0;

%===================================
% create the graph
%===================================

figure
for i = 1:n
        plot(x(i),y(i),'o','MarkerEdgeColor','k',...
                'MarkerFaceColor','c','MarkerSize',15);
        str=num2str(i);
        text((x(i)-0.5),y(i),str);
        ti = procedure;
        xlabel('x-coordinate');
        ylabel('y-coordinate');
        title(['AMRoute in Topology ',num2str(ti)]);
        axis equal;
        grid on
        hold on
end
grid on
hold on
for j=1:n
        v=x(j);
        w=y(j);
        for k=(j+1):n
                d(j,k)=sqrt((x(k)-v)^2+(y(k)-w)^2);
                d(k,j)=d(j,k);
                if (d(j,k)<=trans_range)
                        line([v x(k)],[w y(k)])
                        adjacency_matrix(j,k)=1;
                        adjacency_matrix(k,j)=adjacency_matrix(j,k);
                else
                        adjacency_matrix(j,k)=0;
                        adjacency_matrix(k,j)=adjacency_matrix(j,k);
                end
        end
end
hold on
d;
adjacency_matrix;
a = adjacency_matrix;
sen=sender;
plot(x(sen),y(sen),'o','MarkerEdgeColor','k',...
        'MarkerFaceColor','g','MarkerSize',15);
str=num2str(sen);
text((x(sen)-1),y(sen),str);
hold on

%==================
% assign IP address
%==================
ip_address=[];
ip1=1;
ip2=255;
ip_address=round(ip1+(ip2-ip1)*rand(1,n))

% find the group
```

```
while (length(all_node)~=0)
        repeat=repeat+1;
        NODE_NUMBER=all_node(1);
        adjm=adjacency_matrix;
        connect % call connect.m
        group=connection;
        pause(1)
        if (find(group==sender))
                mrouteReceivers=length(group)-1
                if (procedure==1)
                        num_receiver_ar=AmrouteReceivers
                end
        end
        for abc=1:length(connection)
                if (find(all_node==connection(abc)))
                        all_node(find(all_node==connection(abc)))=[];
                end
        end
        all_node;
        fprintf('========= Start Group %g ===========\n',repeat);

        %==========================
        % broadcast JOIN_REQ message
        %==========================
        exParent=[];
        AllChild=[];
        Parent=connection(1);
        ip_address
        Core=Parent;
        ip_Core=ip_address(Parent);
        Non_core=[];
        nodes=Parent;
        nodesparent=0;
        while (length(Parent)~=0)   % when we still have a Parent
        for pr=1:length(Parent)
                Parent;
                jnow = Parent(pr);
                PR=jnow;
                ip_PR=ip_address(PR);
                fprintf('*** New Parent = %g\n',PR);  % uncomment while observing
                Parent;
                exParent;
                pause(0.1)
                if (find(exParent==PR))
                        Parent;
                        pr;
                        warn=1;
                        Parent(pr)=[];
                        pause(1)
                        break
                end
                Children=[];
                connection(find(connection==PR))=[];
                for k=1:n   % finding the Children
                        if (adjacency_matrix(jnow,k)==0)
                                Children(k)=0;  % it is not connected or itself
                        elseif (adjacency_matrix(jnow,k)==1)
                                Children(k)=k;
                        end
                end
                Children;
                Children(find(Children==0))=[];
                Children;  % eliminate semicolon while observing

                pr;
                PR;
                exParent(length(exParent)+1)=PR;
                pause(0.1)

                %=====================================
                % Core Resolution Procedure
                %=====================================
                if (length(exParent)>1)
                        if (ip_PR>ip_Core)
                                fprintf('Node %g wins the core resolution ',PR);
                fprintf('and becomes the new core node\n');
                fprintf('Node %g becomes a non-core node\n',Core);
                Non_core(length(Non_core)+1)=Core;
                Core=PR;
                ip_Core=ip_PR;
                fprintf('*****************\n');
        elseif (ip_PR<=ip_Core)
                fprintf('Node %g loses the core resolution ',PR);
                fprintf('and becomes a non-core node\n');
                fprintf('Node %g is still the core\n',Core);
                Non_core(length(Non_core)+1)=PR;
                Core=Core;
                ip_Core=ip_Core;
                fprintf('*****************\n');
        end
        pause(0.1)
end
% end of Core Resolution Procedure

% when the Parent has no Children
if ((length(Children)==0)&(pr==length(Parent)))
        Parent=[];
        AllChild;
        if ((length(AllChild)~=0)&(length(connection)~=0))
                AllChild(find(AllChild==PR))=[];
                AllChild;
                Parent=AllChild;
        else
                Parent=[];
        end
elseif (length(Children)~=0)   % when there are children
        while (length(Children)~=0)
        % distributing JOIN_REQ message to each node of the Children
                if (length(Children)~=0)
                        for ch = 1: length(Children)
                                plot_arrow(x(PR),y(PR),x(Children(ch)), ...
                                        y(Children(ch)),'headwidth',0.08,...
                                        'headheight',0.15,'color',...
                                        [1 0 0],'facecolor',[1 0 0]);
                                JOIN_REQ=JOIN_REQ+1;
                                nodes(length(nodes)+1)=Children(ch);
                                nodesparent(length(nodesparent)+1)=PR;
                                hold on
                        end
                        JOIN_REQ;  % eliminate semicolon while observing
                end
                exParent;
                %fprintf('eliminate exParent from Children\n')
                for ep=1:length(exParent)
                        Children(find(Children==exParent(ep)))=[];
                end
                if (pr==1)
                        AllChild=Children;
                else
                        %fprintf('eliminate Children & AllChild\n');
                        for ac=1:length(AllChild)
                                Children(find(Children==AllChild(ac)))=[];
                        end
                        AllChild;
                        AllChild(length(AllChild)+1:length(AllChild)+length(C
                        hildren))=Children;
                        %pause(0.1)
                end

                PR;
                % eliminating the Parent from the AllChild
                if (length(AllChild)~=0)
                        AllChild(find(AllChild==PR))=[];
                end
                AllChild;
                pause(0.1)

                % end of one set of Parent and determining the new Parent
                if (pr==length(Parent))
                        Parent=AllChild;
```

```
                                        %fprintf('End of pr. New set of Parent = ');
                                        %disp(Parent);
                                        %fprintf('\n');
                                        %pause(3)
                                        AllChild=[];
                                end
                                exParent;
                                Parent;
                                connection;
                                Children=[];
                                if (length(connection)==0)
                                        pr;
                                        if (pr==length(Parent))
                                                Parent=[];
                                        end
                                        break
                                end
                        end
                end
        end
end
Core
ip_Core
Non_core
pause(1)
nodes;
nodesparent;
JOIN_REQ
nodes=[];
nodesparent=[];
Children=[];

%=================================
% core sends out TREE_CREATE
%=================================
fprintf('+++++++ TREE_CREATE procedure ++++++++\n');
NAKnode=[];
exParent=[];
AllChild=[];
Parent=Core;
Core=Parent
pause(1)
nodes=Parent;
nodesparent=0;
while (length(Parent)~=0)   % when we still have a Parent
        for pr=1:length(Parent)
            Parent;
            jnow = Parent(pr);
            PR=jnow;
            Parent;
            exParent;
            pause(0.1)
            if (find(exParent==PR))
                Parent;
                pr;
                warn=1;
                Parent(pr)=[];
                pause(1)
                break
            end
            Children=[];
            connection(find(connection==PR))=[];
            for k=1:n   % finding the Children
                if (adjacency_matrix(jnow,k)==0)
                    Children(k)=0;  % it is not connected or itself
                elseif (adjacency_matrix(jnow,k)==1)
                    Children(k)=k;
                    if (find(exParent==Children(k)))
                        Children(k)=0;
                    end
                end
            end
            Children;
            Children(find(Children==0))=[];
            Children;  % eliminate semicolon while observing
```

```
pr;
PR;
exParent(length(exParent)+1)=PR;
%pause(0.1)

% when the Parent has no Children
if ((length(Children)==0)&(pr==length(Parent)))
    Parent;
    Parent=[];
    AllChild;
    pause(0.2)
    if (length(AllChild)~=0)
        Children;
        AllChild(find(AllChild==PR))=[];
        AllChild;
        Parent=AllChild;
        pause(0.1)
    else
        Parent=[];
    end
elseif (length(Children)~=0)   % when there are children
    while (length(Children)~=0)
        % distributing TREE_CREATE message to each node of the Children
        if (length(Children)~=0)
            for ch = 1: length(Children)
                  plot_arrow(x(PR),y(PR),x(Children(ch)),y(Children(ch)),...
                          'headwidth',0.08,'headheight',0.15,'color',...
                          [0 1 0],'facecolor',[0 1 0]);
                  TREE_CREATE=TREE_CREATE+1;
                  PR;
                  Children;
                  fprintf('TREE_CREATE from node %g to node %g\n',PR,
                  Children(ch));
                  if (find(nodes==Children(ch)))
                    plot_arrow(x(Children(ch)),y(Children(ch)),...
                          x(PR),y(PR),'headwidth',0.08,'headheight',...
                          0.15,'color',[1 0 1],'facecolor',[1 0 1]);
                    TREE_CREATE_NAK=TREE_CREATE_NAK+1;
                    fprintf('TREE_CREATE_NAK from %g to %g\n',Children(ch),PR);
                    fprintf('A mesh link is created between node %g ',PR);
                    fprintf('and node %g\n',Children(ch));
                    NAKnode(length(NAKnode)+1)=PR;
                else
                    fprintf('A tree link is created between node %g ',PR);
                    fprintf('and node %g\n',Children(ch));
                end
                fprintf('**********************************\n');
                nodes(length(nodes)+1)=Children(ch);
                nodesparent(length(nodesparent)+1)=PR;
                hold on
            end
            TREE_CREATE;  % eliminate semicolon while observing
            %pause(0.1)
        end
        exParent;
        for ep=1:length(exParent)
            Children(find(Children==exParent(ep)))=[];
        end
        pause(0.1)
        if (pr==1)

            AllChild=Children;
        else
            for ac=1:length(AllChild)
                Children(find(Children==AllChild(ac)))=[];
            end
          AllChild;
          AllChild(length(AllChild)+1:length(AllChild)+length(Children))=Child
          ren;
        end

        PR;
        % eliminating the Parent from the AllChild
        if (length(AllChild)~=0)
```

```
                        AllChild(find(AllChild==PR))=[];
                    end
                    AllChild;
                    %pause(0.1)

                    % end of one set of Parent and determining the new Parent
                    if (pr==length(Parent))
                        Parent=AllChild;
                        AllChild=[];
                    end
                    exParent;
                    Parent;
                    connection;
                    pause(0.1)
                    Children=[];
                end
            end
        end
        end
        Core;
        Non_core;
        nodes;
        nodesparent;
        JOIN_REQ;
        TREE_CREATE;
        TREE_CREATE_NAK;
        fprintf('*** The Number of Messages exchanged in Group %g ***\n',repeat);
        fprintf('JOIN_REQ = %g, TREE_CREATE = %g ',JOIN_REQ,TREE_CREATE);
        fprintf('TREE_CREATE_NAK = %g\n',TREE_CREATE_NAK);
        pause(1)
        messages_ar;
        messages_ar=messages_ar+JOIN_REQ+TREE_CREATE+TREE_CREATE_NAK
        TJR(length(TJR)+1)=JOIN_REQ;
        TTC(length(TTC)+1)=TREE_CREATE;
        TTCN(length(TTCN)+1)=TREE_CREATE_NAK;
        JOIN_REQ=0;
        TREE_CREATE=0;
        TREE_CREATE_NAK=0;
        connection=[];
        group=[];
        pause(0.1)
end

allMessages_ar(length(allMessages_ar)+1)=messages_ar;
messages_ar=[];
repeat_all=1:repeat;
TJR(1)=[];
TJR;
TTC(1)=[];
TTC;
TTCN(1)=[];
TTCN;
tableTotal=[repeat_all;TJR;TTC;TTCN];
fprintf('------------------------------------------------------------------------\n');
disp('A table of Group, Number of JOIN_REQ, TREE_CREATE & TREE_CREATE_NAK');
fprintf('------------------------------------------------------------------------\n');
fprintf('          %g                    %g           %g           %g
\n',tableTotal);
fprintf('------------------------------------------------------------------------\n');
AmrouteReceivers
allMessages_ar
```

```
% AMRTRYOUT starts the AMRIS algorithm
%       It is a short version for display purpose only
%       It needs AMRFIG.m in order to run properly
%
% Author: A. Radyastuti
% v6.5 19-Jan-04

format short
% n=input('Enter n: ');
% x=100*(rand(n,1) -0.5);
% y=100*(rand(n,1) -0.5);

[x,y];

fprintf('=== Start algorithm: AMRIS ===\n');

amrfig % call amrfig.m the new_session figure

tablenode = [nodewith_id;msm_id;itsParent2];
fprintf('--------------------------------------------------\n');
disp('A table of node number, its msm_id, and its parent');
fprintf('--------------------------------------------------\n');
fprintf('                %g         %g            %g      \n',tablenode);
fprintf('--------------------------------------------------\n');
NumberOfNEW_SESSION=NEW_SESSION;
fprintf('The total number of NEW_SESSION message = %g\n',NumberOfNEW_SESSION);
fprintf('\n');
pause(1)

nodewith_id;
nodewith_id2=reverse(nodewith_id);
nodewith_id3=nodewith_id;
nodewith_id3(1)=[];
nodewith_id3;
msm_id;
msm_id2=reverse(msm_id);
msm_id3=msm_id;
msm_id3(1)=[];
msm_id;
itsParent2;
itsParent22=reverse(itsParent2);
itsParent3=itsParent2;
itsParent3(1)=[];
itsParent3;
x;
y;
JOIN_REQ=0;
JOIN_ACK=0;

axis;
for i=1:length(nodewith_id2)-1
    xn(i)=x(nodewith_id2(i));
    xnp(i)=x(itsParent22(i));
    xnnp=[xn(i),xnp(i)];
    yn(i)=y(nodewith_id2(i));
    ynp(i)=y(itsParent22(i));
    ynnp=[yn(i),ynp(i)];
    plot_arrow(xn(i),yn(i),xnp(i),ynp(i),'headwidth',0.08,'headheight',0.15,...
        'color',[0 1 0],'facecolor',[0 1 0]);
    JOIN_REQ=JOIN_REQ+1;
end
JOIN_REQ;
fprintf('The total number of JOIN_REQ message = %g\n',JOIN_REQ);

pause(1)

for i=1:length(nodewith_id3)
    xn3(i)=x(nodewith_id3(i));
    xnp3(i)=x(itsParent3(i));
    xnnp3=[xn3(i),xnp3(i)];
    yn3(i)=y(nodewith_id3(i));
    ynp3(i)=y(itsParent3(i));
    ynnp3=[yn3(i),ynp3(i)];
    plot_arrow(xnp3(i),ynp3(i),xn3(i),yn3(i),'headwidth',0.08,'headheight',0.15,...
        'color',[0 0 0],'facecolor',[0 0 0]);
    JOIN_ACK=JOIN_ACK+1;
end
JOIN_ACK;
fprintf('The total number of JOIN_ACK message = %g\n',JOIN_ACK);
if (procedure==1)
    allMessages=NEW_SESSION+JOIN_REQ+JOIN_ACK;
else
    allMessages(length(allMessages)+1)=NEW_SESSION+JOIN_REQ+JOIN_ACK;
end
fprintf('*** The total number of all messages = %g ***\n',allMessages)
fprintf('\n');
numreceiver=length(nodewith_id)-1;
fprintf('Source: %g; number of receivers= %g\n',sender,numreceiver);
fprintf('\n');
fprintf('=== End of algorithm: AMRIS ===\n');
NEW_SESSION=0;
JOIN_REQ=0;
JOIN_ACK=0;
pause(1)
```

```
% BR1 is an addition procedure for BROKENLINKS3
%
% Author: A. Radyastuti
% v6.5 19-Jun-04

finalparent;
NTR=finalparent
potentparent=[];
connectTOjoinreqmany=0;
for lp=1:length(nodesparent)
    if (nodes(lp)==NTR)
        potentparent(lp)=nodesparent(lp);
    else
        potentparent(lp)=0;
    end
end
potentparent;
potentparent(find(potentparent==0))=[];
potentparent
pause(1)
if (length(potentparent)~=0)
    fprintf('Node %g has potential parent ',NTR)
    disp(potentparent);
    for po=1:length(potentparent)
        if (am(NTR,potentparent(po))==0)
            fprintf('   It is not connected with node %g\n',potentparent(po));
            finalparent(po)=0;
        elseif (am(NTR,potentparent(po))==1)
            fprintf('   It is connected with node %g\n',potentparent(po));
            finalparent(po)=potentparent(po);
        end
    end
    finalparent;
    finalparent(find(finalparent==0))=[];
    finalparent
    pause(1)
    if (length(finalparent)==0)
        fprintf(' It has no access to another node\n');
        NTR
        finalparent

    %==================================================
    % send JOIN_REQ to a potential parent
    %==================================================
    elseif (length(finalparent)==1)
        plot_arrow(x(NTR),y(NTR),x(finalparent),y(finalparent),...
            'linewidth',5,'headwidth',0.08,'headheight',0.15,...
            'color',[1 1 0],'facecolor',[1 1 0]);
        fprintf('-> Node %g sends a JOIN_REQ to node %g\n',NTR,finalparent);
        fprintf('In br1.m\n');
        pause(1)
        JOIN_REQ=JOIN_REQ+1;
        pause(0.1)
        thispart=0;
        node_route(length(node_route)+1)=NTR
        node_route_parent(length(node_route_parent)+1)=finalparent
        pause(1)

        %==================================================
        % Parent sends JOIN_ACK to a new Child
        %==================================================
        if (find(node_tree==finalparent))
            joinack % call joinack.m
            while (length(node_route)~=0)
                joinack % call joinack.m
            end
        elseif (sender==finalparent)
            joinack % call joinack.m
            while (length(node_route)~=0)
                joinack % call joinack.m
            end
        elseif (length(did_br)~=0)
            if (find(did_br==finalparent))
                if (find(node_tree==finalparent))
                    pause(1)
                    joinack % call joinack.m
                else
                    plot_arrow(x(finalparent),y(finalparent),x(NTR),y(NTR),...
                        'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                        'color',[0 0 0],'facecolor',[0 0 0]);
                    JOIN_NACK=JOIN_NACK+1;
                    fprintf('    -> Node %g replies with a JOIN_NACK',finalparent);
                    fprintf('back to node %g\n',NTR);
                    did_br(length(did_br)+1)=NTR;
                    node_tree
                    node_tree_parent
                    pause(1)
                end
            end
        elseif (find(node_to_recover==finalparent))
            %===========================
            % call BR1
            %===========================
            br1 % call br1.m
        end
    elseif (length(finalparent)>=2)
        connectTOjoinreqmany=1;
        pause(1)
        smallerid   % call smallerid.m
    end
end
```

```
% BR2
%
% Author: A. Radyastuti
% v6.5 21-Jun-04

fprintf('=== BR2 (Branch Reconstruction 2) ===\n');
did_br(length(did_br)+1)=NTR;
ack_br2=[];
ack_br2_node=[];
am = adjacency_matrix;
br2_node = find(am(NTR,:)==1)
NTR_id=msm_id_origin(find(nodewith_id_origin==NTR))
if (length(br2_node)~=0)
        for brn=1:length(br2_node)
                broadcast = br2_node(brn);
                plot_arrow(x(NTR),y(NTR),x(broadcast),y(broadcast),...
                        'linewidth',3,'headwidth',0.08,'headheight',0.15,...
                        'color',[0 1 1],'facecolor',[0 1 1]);
                fprintf('-> Node %g sends a JOIN_REQ to node %g\n',NTR,broadcast);
                JOIN_REQ=JOIN_REQ+1;
                pause(1)
        end
        for brn2=1:length(br2_node)
                broadcast2 = br2_node(brn2);
                broadcast2_id=msm_id_origin(find(nodewith_id_origin==broadcast2));
                if (broadcast2==sender)
                        plot_arrow(x(broadcast2),y(broadcast2),x(NTR),y(NTR),...
                                'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                                'color',[1 0 1],'facecolor',[1 0 1]);
                        JOIN_ACK=JOIN_ACK+1;
                        if (length(ack_br2)==0)
                                ack_br2=1;
                        else
                                ack_br2(length(ack_br2)+1)=ack_br2(length(ack_br2))+1;
                        end
                        ack_br2_node(length(ack_br2_node)+1)=broadcast2;
                elseif (find(node_tree==broadcast2))
                        NTR;
                        NTR_id;
                        broadcast2;
                        broadcast2_id;
                        fprintf('msm_id node %g is %g; \n',NTR,NTR_id);
                        fprintf('msm_id node %g is %g\n',broadcast2,broadcast2_id);
                        pause(1)
                        if (NTR_id>broadcast2_id)
                                plot_arrow(x(broadcast2),y(broadcast2),x(NTR),y(NTR),...
                                        'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                                        'color',[1 0 1],'facecolor',[1 0 1]);
                                JOIN_ACK=JOIN_ACK+1;
                                if (length(ack_br2)==0)
                                        ack_br2=1;
                                else
                                        ack_br2(length(ack_br2)+1)=ack_br2(length(ack_br2))+1;
                                end
                                ack_br2_node(length(ack_br2_node)+1)=broadcast2;
                        else
                                node_tree;
                                node_tree_parent;
                                fprintf('Node %g cannot be a parent because it ',broadcast2);
                                fprintf('has a larger msm_id ');
                                fprintf('than node %g => JOIN_NACK\n',NTR);
                                pause(1)
                                plot_arrow(x(broadcast2),y(broadcast2),x(NTR),y(NTR),...
                                        'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                                        'color',[0 0 0],'facecolor',[0 0 0]);
                                JOIN_NACK=JOIN_NACK+1;
                                fprintf('    -> Node %g replies with a JOIN_NACK ',broadcast2);
                                fprintf('back to node %g\n',NTR);
                                node_tree
                                node_tree_parent

                        end
                else
                        node_tree;
                        node_tree_parent;
                        plot_arrow(x(broadcast2),y(broadcast2),x(NTR),y(NTR),...
                                'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                                'color',[0 0 0],'facecolor',[0 0 0]);
                        JOIN_NACK=JOIN_NACK+1;
                        fprintf('    -> Node %g replies with a JOIN_NACK ',broadcast2);
                        fprintf('back to node %g\n',NTR);
                        node_tree
                        node_tree_parent
                        pause(1)
                end
        end

else
        fprintf('There is no node around node %g\n',NTR);
end
ack_br2
NTR;
pause(1)
if (length(ack_br2)~=0)
        if (length(ack_br2)==1)
                broadcast2=ack_br2_node
        elseif (length(ack_br2)>=2)
                nodewith_id_origin
                msm_id_origin
                for abr2=1:length(ack_br2_node)
                        FF=find(nodewith_id_origin==ack_br2_node(abr2));
                        id_abr2(abr2)=msm_id_origin(FF);
                end
                ack_br2_node
                id_abr2
                fprintf('=====\n');
                [id_abr2,IND]=sort(id_abr2)
                ack_br2_node([IND]);

                ack_br2_node=ack_br2_node([IND])
                broadcast2=ack_br2_node(1)
                id_abr2=[];
                plot_arrow(x(NTR),y(NTR),x(broadcast2),y(broadcast2),...
                        'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                        'color',[0 1 0],'facecolor',[0 1 0]);
                JOIN_CONF=JOIN_CONF+1
                pause(2)
        end
        node_tree(length(node_tree)+1)=NTR
        node_tree_parent(length(node_tree_parent)+1)=broadcast2
        pause(1)
        if (length(broken_node)~=0)
                if (find(broken_node==NTR))
                        broken_node
                        NTR
                        broken_child
                        pause(1)
                        while (find(broken_node==NTR))
                                bn=find(broken_node==NTR);
                                lbn=length(node_tree)+length(bn);
                                node_tree_parent(length(node_tree_parent)+1:lbn)=NTR
                                node_tree(length(node_tree)+1:lbn)=broken_child(bn)
                                pause(1)
                                for lbbn=1:length(bn)
                                        ln=broken_child(bn(lbbn));
                                        plot_arrow(x(NTR),y(NTR),x(ln),y(ln),...
                                                'linewidth',3,'headwidth',0.08,...
                                                'headheight',0.15,'color',...
                                                [1 0 1],'facecolor',[1 0 1]);
                                end
                                broken_child(bn)=[]
                                broken_node(find(broken_node==NTR))=[]
                                pause(1)
                                if (length(broken_node)==1)
                                        if (find(node_tree==broken_node))
                                                NTR=broken_node;
                                        end
                                end
                        end
                end
```

```
          end
      end
```

```
% BROKENLINKS3 is trying to recover the tree
%       if the tree becomes disconnected
%
% Author: A. Radyastuti
% v6.5 Last-Modified: 18-Jun-04

am=adjacency_matrix
Sid=sender
nodewith_id
node_broken=nodewith_id;
node_broken(1)=[];
node_broken;
msm_id
msm_broken=msm_id;
msm_broken(1)=[];
msm_broken;
itsParent2
itsPar_broken=itsParent2;
itsPar_broken(1)=[];
itsPar_broken;
check_sign=0;
node_okay=[];
node_to_recover=[];
node_left=[];
broken_parent=[];
node_tree=[];
node_tree_parent=[];
node_route=[];
node_route_parent=[];
msm_id_tree=[];
connectTOjoinreqmany=0;
broken_node=[];
broken_child=[];
did_br=[];
JOIN_REQ=0;
JOIN_ACK=0;
JOIN_NACK=0;
JOIN_CONF=0;
TreeLink=0;
connectTOjoinreqmany=0;
br1br2=0;
skip_step=0;
beacon=0;

%=====================
% beaconing mechanism
%=====================
adjacency_matrix;
for beac1=1:n
    for beac2=1:n
        if (adjacency_matrix(beac1,beac2)==1)
            plot_arrow(x(beac1),y(beac1),x(beac2),y(beac2),'headwidth',0.08,...
                'headheight',0.08,'color',[0 0 1],'facecolor',[0 0 1]);
            hold on
            beacon=beacon+1;
        end
    end
end
beacon;
fprintf('The number of BEACON messages: %g\n',beacon);
pause(1)

%=================
% check if Parent and Children are still connected
%=======================
for ju=1:length(node_broken)
    for jun=1:n
        if (am(node_broken(ju),jun)==0)
            june(jun)=0;
        else
            june(jun)=jun;
        end
    end
    june;
    if (find(june==itsPar_broken(ju)))
```

```
                fprintf('Child %g is still connected ',node_broken(ju));
                fprintf('with Parent %g\n',itsPar_broken(ju));
                ipb=itsPar_broken(ju);
                nb=node_broken(ju);
                plot_arrow(x(ipb),y(ipb),x(nb),y(nb),'headwidth',0.08,...
                    'headheight',0.15,'color',[1 0 1],'facecolor',[1 0 1]);
                hold on
                TreeLink=TreeLink+1;
                if (length(node_okay)==0)
                    if (ipb==sender)
                        node_tree(length(node_tree)+1)=nb;
                        node_tree_parent(length(node_tree_parent)+1)=ipb;
                    else
                        broken_node(length(broken_node)+1)=ipb;
                        broken_child(length(broken_child)+1)=nb;
                    end
                elseif (length(node_okay)>=1)
                    if (find(node_tree==ipb))
                        node_tree(length(node_tree)+1)=nb;
                        node_tree_parent(length(node_tree_parent)+1)=ipb;
                    elseif (ipb==sender)
                        node_tree(length(node_tree)+1)=nb;
                        node_tree_parent(length(node_tree_parent)+1)=ipb;
                    else
                        broken_node(length(broken_node)+1)=ipb;
                        broken_child(length(broken_child)+1)=nb;
                    end
                end
                node_okay(length(node_okay)+1)=nb;
            else
                node_to_recover;
                fprintf('Broken links between node %g ',node_broken(ju));
                fprintf('and node %g\n',itsPar_broken(ju));
                check_sign=check_sign+1; % the sign that there is failure
                node_to_recover(length(node_to_recover)+1)=node_broken(ju);
                broken_parent(length(broken_parent)+1)=itsPar_broken(ju);
            end
        end
        pause(0.1)
        if (find(node_tree_parent==sender))
            fprintf('Sid is a part of the tree\n');
        else
            fprintf('Sid is not connected with any of its children\n');
            newtopo=1;
            transloss2=numreceiver;
            pause(1)
        end
        fprintf('Based on the previous tree: \n');
        if (check_sign>0)
            fprintf('There are %g link failures\n',check_sign);
        elseif (check_sign==0)
            fprintf('The tree needs NO reconfiguration\n');
        end
        node_tree
        node_tree_parent
        node_to_recover % Nodes with broken links
        broken_node
        broken_child
        pause(1)
        node_okay;
        node_okay(length(node_okay)+1)=nodewith_id(1)
        nodewith_id_origin
        for jk=1:length(nodewith_id_origin)
            if (find(nodewith_id==nodewith_id_origin(jk)))
                node_left;
            else
                node_left(length(node_left)+1)=nodewith_id_origin(jk);
            end
        end
        node_left
        pause(0.1)

        %===================
        % Nodes that need link reconfiguration
        %===================
for jkl=1:length(node_left)
    node_to_recover;
    node_left(jkl);
    if (find(node_to_recover==node_left(jkl)))
        node_left(jkl)=0;
    else
        node_left(jkl)=node_left(jkl);
    end
end
node_left;
node_left(find(node_left==0))=[];
if (length(node_left)~=0)
    tcl=length(node_to_recover)+length(node_left);
    node_to_recover(length(node_to_recover)+1:tcl)=node_left
    broken_parent(length(broken_parent)+1:tcl)=0
end

%=====================================
% BR1 (Branch Reconstruction 1)
%=====================================
node_to_recover
broken_parent
nodes;
nodesparent;
pause(0.1)
for ntc=1:length(node_to_recover)
    node_to_recover
    broken_parent
    NTR=node_to_recover(ntc)
    fprintf('*** New beginning. NTR = %g\n',NTR);
    if (skip_step==1)
        skip_step=0;
    end
    pause(0.2)
    BP=broken_parent(ntc)
    if (length(node_tree)~=0)
        if (find(node_tree==NTR))
            fprintf('Node %g has been recovered\n',NTR);
            pause(1)
            if (ntc==length(node_to_recover))
                break
            else
                skip_step=1;
                pause(0.1)
            end
        end
    end
    if (skip_step==0)
        potentparent=[];
        for npr=1:length(nodesparent)
            if (nodes(npr)==NTR)
                potentparent(npr)=nodesparent(npr);
            else
                potentparent(npr)=0;
            end
        end
        potentparent;
         potentparent(find(potentparent==0))=[];
        potentparent(find(potentparent==BP))=[];
        potentparent;
        if (length(potentparent)~=0)
            for ptp=1:length(potentparent)
                if (find(broken_child==potentparent(ptp)))
                    fprintf('potentparent in broken_child\n');
                    potentparent(ptp)=0;
                end
            end
        end
        potentparent(find(potentparent==0))=[];
        if (length(potentparent)~=0)
            fprintf('Node %g has another potential parent ',NTR)
            disp(potentparent);
            for po=1:length(potentparent)
                if (am(NTR,potentparent(po))==0)
                    fprintf('   It is not connected with node
                %g\n',potentparent(po));
             finalparent(po)=0;
         elseif (am(NTR,potentparent(po))==1)
             fprintf('   It is connected with node
                %g\n',potentparent(po));
             finalparent(po)=potentparent(po);
         end
     end
     finalparent;
     finalparent(find(finalparent==0))=[];
     finalparent
     pause(1)
     if (length(finalparent)==0)
         fprintf(' It has to find access to any other node\n');
         %==================
         % BR2
         %==================
         fprintf('Call br2 in brokenlinks3\n');
         pause(0.1)
         br2 % call br2.m
     %==================================================
     % send JOIN_REQ to a potential parent
     %==================================================
     elseif (length(finalparent)==1)
         plot_arrow(x(NTR),y(NTR),x(finalparent),y(finalparent),...
             'linewidth',5,'headwidth',0.08,'headheight',0.15,...
             'color',[1 1 0],'facecolor',[1 1 0]);
         fprintf('-> Node %g sends a JOIN_REQ to node %g\n',NTR,finalparent);
         JOIN_REQ=JOIN_REQ+1;
         pause(0.1)
         thispart=0;
         %==================================================
         % Parent sends JOIN_ACK to a new Child
         %==================================================
         if (find(node_tree==finalparent))
             joinack % call joinack.m
         elseif (sender==finalparent)
             joinack % call joinack.m
         elseif (length(did_br)~=0)
             if (find(did_br==finalparent))
                 if (find(node_tree==finalparent))
                     fprintf('Did br but on the tree\n');
                     joinack % call joinack.m
                 else
                     plot_arrow(x(finalparent),y(finalparent),...
                             x(NTR),y(NTR),'linewidth',2,...
                             'headwidth',0.08,'headheight',0.15,...
                             'color',[0 0 0],'facecolor',[0 0 0]);
                     JOIN_NACK=JOIN_NACK+1;
                     fprintf('    -> Node %g replies with a JOIN_NACK',finalparent);
                     fprintf('back to node %g\n',NTR);
                     did_br(length(did_br)+1)=NTR;
                     node_tree
                     node_tree_parent
                     pause(0.1)
                 end
             end
         else
             %==============================
             % call BR1
             %==============================
             node_route(length(node_route)+1)=NTR
             node_route_parent(length(node_route_parent)+1)=finalparent
             fprintf('BR1 in brokenlinks3.m\n');
             pause(0.1)
             br1 % call br1.m
         end
     elseif (length(finalparent)>=2)
         connectTOjoinreqmany=1;
         fprintf('We need smallerid in brokenlinks3.m\n');
         pause(0.1)
         smallerid   % call smallerid.m
     end
 elseif (length(potentparent)==0)
     fprintf('NO potentparent AT ALL. br2 in brokenliks3.m\n');
```

```
                pause(0.1)
                br2 % call br2.m
            end
        elseif (skip_step==1)
            pause(0.1)
        end
    end
    node_tree_parent
    node_tree
    for msid=1:length(node_tree)
        msm_id_tree(msid)=msm_id_origin(find(nodewith_id_origin==node_tree(msid)));
    end
    msm_id_tree
    pause(1)

    if (length(node_tree)~=0)
           if (length(node_tree)<numreceiver)
            transloss2=numreceiver-length(node_tree);
            fprintf('After reconfiguration attempt, Sid can ');
            fprintf('multicast data to nodes\n');
            disp(node_tree);
            fprintf('==> There is %g transmission loss',transloss2);
            fprintf('\n');
        elseif (length(node_tree)==numreceiver)
            transloss2=0;
            fprintf('==> There is no transmission loss\n');
        end
        pause(1)
    end
    if (newtopo==1)
        fprintf('No connection for Sid -> transloss2 = %g\n',transloss2);
    end

    transLoss(length(transLoss)+1)=transloss2;
    JOIN_REQ;
    JOIN_ACK;
    JOIN_NACK;
    JOIN_CONF;
    TreeLink;
    fprintf('Number of each message:\n');
    fprintf('JOIN_REQ=%g; JOIN_ACK=%g; JOIN_NACK=%g',JOIN_REQ,JOIN_ACK,JOIN_NACK);
    fprintf('JOIN_CONF=%g; TreeLink=%g; beacon=%g\n',JOIN_CONF,TreeLink,beacon);
    allMessages2=JOIN_REQ+JOIN_ACK+JOIN_NACK+TreeLink+beacon;
    allMessages(length(allMessages)+1)=allMessages2;
    fprintf('Total number of messages = %g\n\n',allMessages2);
    pause(1)
    JOIN_REQ=0;JOIN_ACK=0;JOIN_NACK=0;TreeLink=0;beacon=0;

    nodewith_id=[];
    nodewith_id=node_tree;
    nodewith_id(2:length(nodewith_id)+1)=nodewith_id;
    nodewith_id(1)=Sid;
    nodewith_id;
    msm_id=[];
    msm_id=msm_id_tree;
    msm_id(2:length(msm_id)+1)=msm_id;
    msm_id(1)=1;
    msm_id;
    itsParent2=[];
    itsParent2=node_tree_parent;
    itsParent2(2:length(itsParent2)+1)=itsParent2;
    itsParent2(1)=0;
    itsParent2;
    tablenode = [nodewith_id;msm_id;itsParent2];
    fprintf('---------------------------------------------------\n');
    disp('A table of node number, its msm_id, and its parent');
    fprintf('---------------------------------------------------\n');
    fprintf('                  %g         %g            %g      \n',tablenode);
    fprintf('---------------------------------------------------\n');
    node_broken=[];
    msm_broken=[];
    itsPar_broken=[];
    pause(1)
```

```
% CONNECT determines connection
% CONNECT includes all nodes that are connected
%       with the source (including the source itself)
%
% Author: A. Radyastuti
% v6.5 Last-Modified: 21-Jun-04

if (NODE_NUMBER>n)
    looping=0;
else
    NN=NODE_NUMBER;
    atoz=NN;
    connection=zeros(1,n);
    connection(NN)=NN;
    while (length(atoz)~=0)
        NN=atoz(1);
        for cn=1:n
            if (find(adjm(NN,cn)==1))
                if (find(connection==cn))
                    atoz;
                else
                    atoz(length(atoz)+1)=cn;
                    connection(cn)=cn;
                end
            end
        end
        atoz(1)=[];
    end
end
connection;
connection(find(connection==0))=[];
connection
```

```
% GENERATEIDNEW generates own msm-id
%       It is a short version to run
%       with AMRFIG.m and AMRTRYOUT.m
%
% Author: A. Radyastuti
% v6.5 27-Jan-04

if (PR==sender)
    msm_sender=1;
    nodewith_id=PR;
    msm_id=msm_sender;
    itsParent2=0;
elseif (PR~=sender)
    delta=round(10*rand(1));
    while (delta<=1)
        delta=round(10*rand(1));
    end
    for np=1:length(nodesparent)
               if (nodes(np)==PR)
                   potparent(np)=nodesparent(np);
               else
                   potparent(np)=0;
        end
       end
       potparent;
    PotentialParent=potparent;
    PotentialParent(find(PotentialParent==0))=[];
    node=PR;
    PotentialParent;
    if (length(PotentialParent)==1)
        usemsm_id=msm_id(nodewith_id==PotentialParent);
        itsParent=PotentialParent;
    elseif (length(PotentialParent)>1)
        %fprintf('==There are more than 1 potential parent==\n')
        list_node=nodewith_id;
        list_id=msm_id;
        length(list_node);
        itsParent=PotentialParent;
        for ls=1:length(list_node)
            if (find(itsParent==list_node(ls)))
                list_node(ls)=list_node(ls);
                list_id(ls)=list_id(ls);
            else
                list_node(ls)=0;
                list_id(ls)=0;
            end
        end
        list_node;
        list_node(find(list_node==0))=[];
        list_node;
        list_id;
        list_id(find(list_id==0))=[];
        list_id;
        usemsm_id=min(list_id);
        itsParent=list_node(list_id==min(list_id));
        if (length(itsParent)>1)
            itsParent=itsParent(1);
            usemsm_id=list_id(1);
        end
    end
    nodewith_id(length(nodewith_id)+1)=PR;
    delta;
    msm_id(length(msm_id)+1)=usemsm_id+delta;
    itsParent;
    itsParent2(length(itsParent2)+1)=itsParent;
    pause(0.1)
end
potparent=[];
PotentialParent=[];
list_node=[];
list_id=[];
itsParent=[];
```

```
% JOINACK is a program to plot the JOIN_ACK
%       messages in the graph
%
% Author: A. Radyastuti
% v6.5 Last-Modified: 19-Jun-04

fprintf('JOINACK starts\n')
bingung=0;
if (length(node_route)~=0)
        while (length(node_route)~=0)
                node_route_parent;
                node_route_parent=reverse(node_route_parent)
                node_route;
                node_route=reverse(node_route)
                finalparent=node_route_parent(1);
                NTR=node_route(1);
                pause(1)
                plot_arrow(x(finalparent),y(finalparent),x(NTR),y(NTR),...
                        'linewidth',3,'headwidth',0.08,'headheight',0.15,...
                        'color',[1 0 1],'facecolor',[1 0 1]);
                JOIN_ACK=JOIN_ACK+1;
                fprintf('    -> Node %g replies with a JOIN_ACK
                ',finalparent);
                fprintf('back to node %g\n',NTR);
                node_tree(length(node_tree)+1)=NTR
                node_tree_parent(length(node_tree_parent)+1)=finalparent
                if (length(node_route)~=0)
                        node_route_parent(1)=[];
                        node_route(1)=[];
                        pause(1)
                end
                node_route_parent
                node_route
                pause(3)
                if (length(broken_node)~=0)
                        if (find(broken_node==NTR))
                                while (find(broken_node==NTR))
                                        bn=find(broken_node==NTR)
                                        lbn=length(node_tree)+length(bn);
                                        node_tree_parent(length(node_tree_parent)+1:lbn)=NTR
                                        node_tree(length(node_tree)+1:lbn)=broken_child(bn)
                                        broken_child(bn)=[]
                                        broken_node(find(broken_node==NTR))=[]
                                        pause(1)
                                        if (length(broken_node)==1)
                                                if (find(node_tree==broken_node))
                                                        NTR=broken_node;
                                                end
                                        elseif (length(broken_node>=2))
                                                NTR=broken_node(1)
                                                pause(5)
                                        end
                                end
                        end
                end
                finalparent=[];
                NTR=[];
                if (length(node_route)==0)
                        bingung=1;
                        pause(0.5)
                        break
                end
        end
end
if (bingung==0)
        plot_arrow(x(finalparent),y(finalparent),x(NTR),y(NTR),...
                'linewidth',3,'headwidth',0.08,'headheight',0.15,...
                'color',[1 0 1],'facecolor',[1 0 1]);
        JOIN_ACK=JOIN_ACK+1;
        fprintf('    -> Node %g replies with a JOIN_ACK ',finalparent);
        fprintf('back to node %g\n',NTR);
        node_tree(length(node_tree)+1)=NTR;
        node_tree_parent(length(node_tree_parent)+1)=finalparent;
        if (length(broken_node)~=0)
                if (find(broken_node==NTR))
```

```
                        while (find(broken_node==NTR))
                            bn=find(broken_node==NTR)
                            lbn=length(node_tree)+length(bn);
                            node_tree_parent(length(node_tree_parent)+1:lbn)=NTR
                            node_tree(length(node_tree)+1:lbn)=broken_child(bn)
                            broken_child(bn)=[]
                            broken_node(find(broken_node==NTR))=[]
                            pause(1)
                            if (length(broken_node)==1)
                                if (find(node_tree==broken_node))
                                    NTR=broken_node;
                                end
                            elseif (length(broken_node>=2))
                                NTR=broken_node(1)
                                pause(2)
                            end
                        end
                    end
                end
                finalparent=[];
                NTR=[];
                if (length(node_route)~=0)
                    node_route_parent(1)=[];
                    node_route(1)=[];
                end
            end
            bingung=0;
            finalparent=[];
```

```
% JOINREQMANY is to support smallerid
%
% Author: A. Radyastuti
% v6.5 19-Jun-04

fprintf('JOINREQMANY starts\n');
finalparent2
pause(1)
for jrq=1:length(finalparent2)
        finalparent=finalparent2(jrq);
        plot_arrow(x(NTR),y(NTR),x(finalparent),y(finalparent),...
                'linewidth',5,'headwidth',0.08,'headheight',0.15,...
                'color',[1 1 0],'facecolor',[1 1 0]);
        fprintf('-> Node %g sends a JOIN_REQ to node %g\n',NTR,finalparent);
        pause(1)
        JOIN_REQ=JOIN_REQ+1;
        pause(0.1)
        thispart=0;
        if (connectTOjoinreqmany==1)
                node_route(length(node_route)+1)=NTR
                node_route_parent(length(node_route_parent)+1)=finalparent
                pause(1)
                connectTOjoinreqmany=0;
        end
        %================================================
        % Parent sends JOIN_ACK to a new Child
        %================================================
        if (find(node_tree==finalparent))
                joinack % call joinack.m
                pause(1)
                break
        elseif (sender==finalparent)
                joinack % call joinack.m
                pause(1)
                break
        elseif (length(did_br)~=0)
                if (find(did_br==finalparent))
                        if (find(node_tree==finalparent))
                                joinack % call joinack.m
                        else
                                plot_arrow(x(finalparent),y(finalparent),x(NTR),y(NTR),...
                                        'linewidth',2,'headwidth',0.08,'headheight',0.15,...
                                        'color',[0 0 0],'facecolor',[0 0 0]);
                                JOIN_NACK=JOIN_NACK+1;
                                fprintf('    -> Node %g replies with a JOIN_NACK',
                                finalparent);
                                fprintf('back to node %g\n',NTR);
                                did_br(length(did_br)+1)=NTR;
                                pause(1)
                                node_route=[];
                                node_route_parent=[];
                                node_tree
                                node_tree_parent
                                pause(1)
                        end
                end
        end
end
finalparent=[];
finalparent2=[];
```

```
% METRICSBREAK3
%   Each network topology is measured to find:
%       the number of arcs per node
%       the radius of the network
%       It needs NUMARC.m and NETRADIUS.m in order
%           to run properly
%
% Author: A. Radyastuti
% v6.5 Last-Modified: 07-Jan-04
% v7.0 Last-Modified: 29-Jun-04

%==================================================
% input
%==================================================
format short
n = input('Enter a number of nodes in the network: ');
prcd=input('How many network topologies? ');
area=input('Enter the start length of the network area (in meter): ');
area_max=input('Enter the maximum length of the network area (in meter): ');
trans_range=input('Enter the transmission range (in meter): ');
move_max=input('Enter the maximum distance a node can move (in meter): ');

%================================
% random topology generator
%================================
x = area*(rand(n,1) -0.5)   % generate x-coordinate of random points
y = area*(rand(n,1) -0.5)   % generate y-coordinate of random points
[x,y];  % uncomment to see the random coordinates

%==================================================
% initialization
%==================================================
procedure = 1;
start=1;
arcnumber=[]; % initialize arc metric
radtop=[]; % initialize radius metric
newtopo=0;
allMessages_o=[];
allMessages_ar=[];

while (procedure<=prcd) % repeating topology
        figure
        for i = 1:n
                plot(x(i),y(i),'or','MarkerEdgeColor','k',...
                        'MarkerFaceColor','r','MarkerSize',15);
                str = num2str(i);
                text((x(i)-1),y(i),str);
                xlabel('x-coordinate');
                ylabel('y-coordinate');
                ti = procedure;
                title(['Random Points in Topology ',num2str(ti)]);
                %legend('Mobile Nodes');
                axis equal;
                hold on
        end
        grid on
        hold on

        %==================================================
        % connect nodes within range
        %==================================================
        for j=1:n
                v=x(j);
                w=y(j);
                for k=(j+1):n
                        d(j,k)=sqrt((x(k)-v)^2+(y(k)-w)^2);
                        d(k,j)=d(j,k);
                        if (d(j,k)<=trans_range)
                                line([v x(k)],[w y(k)])
                                adjacency_matrix(j,k)=1;
                                adjacency_matrix(k,j)=adjacency_matrix(j,k);
                        else
                                adjacency_matrix(j,k)=0;
                                adjacency_matrix(k,j)=adjacency_matrix(j,k);
                        end
        end
end
hold on
d;
adjacency_matrix;
a = adjacency_matrix;
fprintf('\n');

pause(0.5)

numarc  % call numarc.m

arcnumber(length(arcnumber)+1)=avgline;

if (start==1)
sender=input('Enter source node: ');
        fprintf('\n');
else
        sender=sender;
        fprintf('Source: %g\n',sender);
end
pause(0.5)

netradius   % call netradius.m

radtop(length(radtop)+1)=r;

if (procedure==1)
        %============
        % run AMRIS
        %============
        amrtryout
        if (procedure==1)
                fprintf('Procedure=1\n');
                transLoss=0;
                nodewith_id_origin=nodewith_id
                msm_id_origin=msm_id
                itsParent2_origin=itsParent2
        end
        pause(2)

        %============
        % run ODMRP
        %============
        fprintf('### Preparing for ODMRP...\n');
        pause(0.5)
        odmrptry    % call odmrptry.m
        fprintf('### End of ODMRP ###\n');
        pause(3)
        if (procedure==1)
                num_receiver_o=length(receiver_o)
                transLoss_o=0
                pause(2)
        end
        if (newtopo==1)
                transLoss_o(length(transLoss_o)+1)=0
                num_receiver_o=length(receiver_o)
                pause(2)
        end
        receiver_o=[];

        %============
        % run AMRoute
        %============
        fprintf('### Preparing for AMRoute...\n');
        pause(0.5)
        amroutetry    % call amroutetry.m
        fprintf('### End of AMRoute ###\n');
        if (procedure==1)
                transLoss_ar=0
                pause(2)
        end
        if (newtopo==1)
                num_receiver_ar=length(AmrouteReceivers)
                transLoss_ar(length(transLoss_ar)+1)=0
                pause(2)
        end
        mrouteReceivers=[];

elseif (procedure>=2)   % topology changes
        hold on
        if (newtopo==1)
                newtopo=0;
        end
        for i = 1:n
                plot(x(i),y(i),'or','MarkerEdgeColor','k',...
                        'MarkerFaceColor','r','MarkerSize',15);
                str = num2str(i);
                text((x(i)-1),y(i),str);
                xlabel('x-coordinate');
                ylabel('y-coordinate');
                ti = procedure;
                title(['Random Points in Topology ',num2str(ti)]);
                %legend('Mobile Nodes');
                axis equal;
                hold on
        end
        grid on
        hold on
        hold on
        sen=sender;
        plot(x(sen),y(sen),'o','MarkerEdgeColor','k',...
                'MarkerFaceColor','g','MarkerSize',15);
        str=num2str(sen);
        text((x(sen)-1),y(sen),str);
        hold on
        for j=1:n
                v=x(j);
                w=y(j);
                for k=(j+1):n
                        d(j,k)=sqrt((x(k)-v)^2+(y(k)-w)^2);
                        d(k,j)=d(j,k);
                        if (d(j,k)<=trans_range)
                                line([v x(k)],[w y(k)])
                                adjacency_matrix(j,k)=1;
                                adjacency_matrix(k,j)=adjacency_matrix(j,k);
                        else
                                adjacency_matrix(j,k)=0;
                                adjacency_matrix(k,j)=adjacency_matrix(j,k);
                        end
                end
        end
        hold on

        brokenlinks3 % call brokenlinks.m

        %============
        % run ODMRP
        %============
        fprintf('### Preparing for ODMRP...\n');
        pause(0.5)
        odmrptry    % call odmrptry.m
        fprintf('### End of ODMRP ###\n');
        num_receiver_o
        if (length(receiver_o)<num_receiver_o)
                transLoss_o(length(transLoss_o)+1)=num_receiver_o-length(receiver_o)
        else
                transLoss_o(length(transLoss_o)+1)=0
        end
        pause(2)
        receiver_o=[];

        %============
        % run AMRoute
        %============
        fprintf('### Preparing for AMRoute...\n');
        pause(0.5)
        amroutetry    % call amroutetry.m
        fprintf('### End of AMRoute ###\n');
        AmrouteReceivers
```

```
                num_receiver_ar
                if (AmrouteReceivers<num_receiver_ar)
                        transLoss_ar(length(transLoss_ar)+1)=num_receiver_ar-
                        AmrouteReceivers
                else
                        transLoss_ar(length(transLoss_ar)+1)=0
                end
                pause(2)
                AmrouteReceivers=[];
        end

        if (procedure~=prcd) % limit of iteration
                fprintf('CREATING TOPOLOGY %g\n',procedure+1);
                fprintf('-- The nodes move randomly from their previous location --\n');
                newcolor=0;
                deltax = move_max*(rand(n,1) -0.5);
                deltay = move_max*(rand(n,1) -0.5);
                area_max_half=area_max/2;
                for delx=1:n
                        xn=x(delx)+deltax(delx);
                        while ((xn>area_max_half) | (xn<-(area_max_half)))
                                deltax(delx)=move_max*(rand(1,1) -0.5);
                                xn=x(delx)+deltax(delx);
                        end
                end
                for dely=1:n
                        yn=y(dely)+deltay(dely);
                        while ((xn>area_max_half) | (xn<-(area_max_half)))
                                deltay(dely)=move_max*(rand(1,1) -0.5);
                                yn=y(dely)+deltay(dely);
                        end
                end
                x = x + deltax;
                y = y + deltay;
                start=start+1;
                newcolor=0;
        else
                fprintf('\n');
                fprintf('End of iteration\n\n');
        end
        procedure=procedure+1;
        x
        y
end %while-----------------------

fprintf('===== Final variables =====\n');
arcnumber
radtop
allMessages
transLoss
allMessages_o
transLoss_o
allMessages_ar
transLoss_ar
M1=arcnumber;
M2=radtop;
V1=allMessages;
V2=transLoss;
V3=allMessages_o;
V4=transLoss_o;
V5=allMessages_ar;
V6=transLoss_ar;
topoNUM=1:length(arcnumber);
tablenode_final = [topoNUM;M1;M2;V1;V2;V3;V4;V5;V6];
fprintf('-------------------------------------------------\n');
disp('Topology, NUMofARCS, topoRADIUS, V1, V2, V3, V4, V5, V6');
fprintf('-------------------------------------------------\n');
fprintf('%g       %f    %f     %g      %g      %g     %g     %g     %g      \n',tablenode_final);
fprintf('-------------------------------------------------\n');
figure
met3d   % call met3d.m
metricsresultnew % call metricsresultnew.m
```

```
% NETRADIUS is a procedure of determining the radius
%       of a bounding circle around all points
%
% Author: A. Radyastuti
% v6.5 Sep-03

xd = x;
yd = y;
ymin = min(yd);              % find the smallest y-coordinate
ym = find(yd==ymin);
p1 = xd(ym);
p2 = yd(ym);
P = [p1,p2];                 % a point P with the smallest y-coordinate
xd(ym)=[];
yd(ym)=[];
Old=[xd,yd];
for qq=1:(n-1)
        qx=Old(qq,1);
        qy=Old(qq,2);
        alq=[qx,qy];
        if (qx==p1)
                degree1(qq)=90;
                break
        else
                if (qx<p1)
                        imnode=[p1-1,p2];
                elseif (qx>p1)
                        imnode=[p1+1,p2];
                end
                vec1q=imnode-P;
                vec2q=alq-P;
                angle1(qq) = acos(((vec1q)*vec2q')/(norm(vec1q)*norm(vec2q)));
                degree1(qq) = angle1(qq)*180/pi;
        end
end
degree1;
degree2 = min(degree1);
mdd = find(degree1==degree2);
q1 = Old(mdd,1);
q2 = Old(mdd,2);
Q = [q1,q2];                 % a point Q such that the angle of the line
                             % segment PQ with the x axis is minimal
xd(mdd) = [];
yd(mdd) = [];
Old = [xd,yd];
degreeP=inf;
degreeQ=inf;
degreeR=inf;
% procedure below continues until we find triangle PQR with acute angles
% only
while (degreeP>90) | (degreeQ>90) | (degreeR>90)
    P = [p1,p2];
    Q = [q1,q2];
    T = [P;Q];
    Old=[xd,yd];
    for rp=1:(n-2)           % check the remaining points
        Old;                 % coordinate of remaining points
        xd(rp)=Old(rp,1);
        yd(rp)=Old(rp,2);
        nodelast=[xd(rp),yd(rp)];
        f=[p1 q1 xd(rp) p1];
        g=[p2 q2 yd(rp) p2];
        vec1 = P-nodelast;
        vec2 = Q-nodelast;
        norm(vec1);
        norm(vec2);
        angle(rp) = acos(((vec1)*vec2')/(norm(vec1)*norm(vec2)));
        degree(rp)=angle(rp)*180/pi;
    end
    degreeR = min(degree);   % find a minimum angle (in degree)
    md = find(degree==degreeR);
    if (degreeR<=90)
        r1=Old(md,1);
        r2=Old(md,2);
        R =[r1,r2];
    RPQ1=R-P;
    RPQ2=Q-P;
    norm(RPQ1);
    norm(RPQ2);
    angleP = acos(((RPQ1)*RPQ2')/(norm(RPQ1)*norm(RPQ2)));
    degreeP=angleP*180/pi;
    PQR1 = P-Q;
    PQR2 = R-Q;
    norm(PQR1);
    norm(PQR2);
    angleQ = acos(((PQR1)*PQR2')/(norm(PQR1)*norm(PQR2)));
    degreeQ=angleQ*180/pi;
    if (degreeP>90)
        xd(md)=[];
        yd(md)=[];
        xd(n-2)=p1;
        yd(n-2)=p2;
        p1 = r1;         % replace P by R
        p2 = r2;
        P = [p1,p2];
        T = [P;Q];
        Old=[xd,yd];
     elseif (degreeQ>90)
        xd(md)=[];
        yd(md)=[];
        xd(n-2)=q1;
        yd(n-2)=q2;
        q1 = r1;         % replace Q by R
        q2 = r2;
        Q = [q1,q2];
        T = [P;Q];
        Old=[xd,yd];
     end
else                       % a circle is determined by maximum distance PQ
    x_centre = 1/2*(p1+q1);
    y_centre = 1/2*(p2+q2);
    centre = [x_centre,y_centre];
    r = 1/2*sqrt((p1-q1)^2+(p2-q2)^2);

    % in order not to display too many figures
    % start comment from this point while running with metricsbreak3.m
    figure
    plot(xd,yd,'or','MarkerEdgeColor','k',...
            'MarkerFaceColor','r','MarkerSize',12);
    axis equal;
    grid;
    hold on
    f2 = [p1 q1];
    g2 = [p2 q2];
    plot(f2,g2,'-ob','MarkerEdgeColor','k',...
            'MarkerFaceColor','r','MarkerSize',12);
    hold on
    plot(x_centre,y_centre,'*k');
    axis equal;
    grid;
    hold on
    % stop comment at this point while running with metricsbreak3.m

    centre = [x_centre,y_centre];
    NOP = 100;
    THETA = linspace(0,2*pi,NOP);
    RHO = ones(1,NOP)*r;
    [X,Y] = pol2cart(THETA,RHO);
    X = X + centre(1);
    Y = Y + centre(2);

    % in order not to display too many figures
    % start comment from this point while running with metricsbreak3.m
    plot(X,Y,'-.m');    % draw a bounding circle
    axis equal;
    grid;
    fprintf('------------------------------------------------------\n');
    fprintf('OUTPUT\n');
    fprintf('------------------------------------------------------\n');
    fprintf('Maximum distance is between the following nodes:\n');
```

```
            fprintf('P(%.4f,%.4f) and Q(%.4f,%.4f)\n',p1,p2,q1,q2);
            fprintf('*** Radius of this network topology is %.4f ***\n',r);
            fprintf('-----------------------------------------------------\n');
            % stop comment at this point while running with metricsbreak3.m

            break               % exit the loop
        end
    end

    % a circle is determined through the three points P, Q, and R
    if (degreeP<=90) & (degreeQ<=90) & (degreeR<=90)
            A = q1 - p1;
            B = q2 - p2;
            C = r1 - p1;
            D = r2 - p2;
            E = A*(p1 + q1) + B*(p2 + q2);
            F = C*(p1 + r1) + D*(p2 + r2);
            G = 2.0*(A*(r2 - q2) - B*(r1 - q1));
            x_centre = (D*E - B*F) / G;
            y_centre = (A*F - C*E) / G;
            r = sqrt((p1 - x_centre)^2 + (p2 - y_centre)^2);

            % in order not to display too many figures
            % start comment from this point while running with metricsbreak3.m
            figure
            f1=[p1 q1 r1 p1];
            g1=[p2 q2 r2 p2];
            plot(f1,g1,'-or','MarkerEdgeColor','k',...
                    'MarkerFaceColor','r','MarkerSize',12);
            axis equal;
            grid;
            hold on
            % stop comment at this point while running with metricsbreak3.m

            centre = [x_centre,y_centre];
            NOP = 100;
            THETA = linspace(0,2*pi,NOP);
            RHO = ones(1,NOP)*r;
            [X,Y] = pol2cart(THETA,RHO);
            X = X + centre(1);
            Y = Y + centre(2);

            % in order not to display too many details and tables
            % start comment from this point while running with metricsbreak3.m
            plot(x,y,'or','MarkerEdgeColor','k',...
                    'MarkerFaceColor','r','MarkerSize',12);
            axis equal;
            grid;
            hold on
            plot(x_centre,y_centre,'*k');
            hold on
            plot(X,Y,'-.m');        % draw a bounding circle
            axis equal;
            grid;
            fprintf('-----------------------------------------------------\n');
            fprintf('OUTPUT\n');
            fprintf('-----------------------------------------------------\n');
            fprintf('P(%.4f,%.4f) and the degree at P = %.3f\n',p1,p2,degreeP);
            fprintf('Q(%.4f,%.4f) and the degree at Q = %.3f\n',q1,q2,degreeQ);
            fprintf('R(%.4f,%.4f) and the degree at R = %.3f\n',r1,r2,degreeR);
            % stop comment at this point while running with metricsbreak3.m

            fprintf('*** Radius of this network topology is %.4f ***\n',r);
            fprintf('-----------------------------------------------------\n\n');
    end
```

```
% NUMARC is a procedure of determining the
%       average number of arcs per node
%
% Author: A. Radyastuti
% v6.5 Sep-03

for j = 1:n
        a = find(d(j,:)>0 & d(j,:)<=trans_range);
        b(j) = numel(a);
end
node = [1:n];
b;
table = [node;b];
fprintf('---------------------------------------------\n');
disp('A table of node numbers and number of lines');
fprintf('                       %g              %g      \n',table);
fprintf('---------------------------------------------\n');
mi = min(b);
ma = max(b);
numnode0 = 0;
numnode = [];
for m = mi:ma
        if (mi==0)
                mm0 = find(b==0);
                numnode0 = numel(mm0);
                mi = mi+1;
        else
                mm = find(b==m);
                numnode(m) = numel(mm);
        end
end
if (numnode0>0)
    numline = [0:ma];
    numnode(2:(end+1)) = numnode;
    numnode(1,1) = numnode0;
    numnode;
else
    numline = [1:ma];
    numnode;
end
table2 = [numline; numnode];

% in order not to display the following table
% start comment from this point while running with metricsbreak3.m
fprintf('---------------------------------------------------\n');
disp('A table of number of lines and number of nodes');
fprintf('---------------------------------------------------\n');
fprintf('                       %g                  %g      \n',table2);
fprintf('---------------------------------------------------\n');
% stop comment at this point while running with metricsbreak3.m

avgline = numline*(numnode')/n;
fprintf('-> The average number of lines per node: %.4f\n\n',avgline);

numline = [];
numnode = [];
```

```
% ODMRPtry
%
% Author: A. Radyastuti
% v6.5 March-2004
% v7.0 Last-Modified: 25-Jun-04

% start comment from this point while running with metricsbreak3.m

format short
n = input('Enter a number of nodes in the network: ');
x = 100*(rand(n,1) -0.5);   % generate x-coordinate of random points
y = 100*(rand(n,1) -0.5);   % generate y-coordinate of random points
[x,y];  % uncomment to see the random coordinates
% stop comment at this point while running with metricsbreak3.m


%==================================
% initialization
%==================================
allMessages_od=[];
receiver_o=[];
ortu=[];

%==================================
% create the graph
%==================================
figure
for i = 1:n
    plot(x(i),y(i),'o','MarkerEdgeColor','k',...
        'MarkerFaceColor','c','MarkerSize',15);
    str=num2str(i);
    text((x(i)-0.5),y(i),str);
    ti = procedure;
    xlabel('x-coordinate');
    ylabel('y-coordinate');
    title(['ODMRP in Topology ',num2str(ti)]);
    axis equal;
    grid;
    hold on
end
grid on
hold on
for j=1:n
    v=x(j);
    w=y(j);
    for k=(j+1):n
        d(j,k)=sqrt((x(k)-v)^2+(y(k)-w)^2);
        d(k,j)=d(j,k);
        if (d(j,k)<=trans_range)
            line([v x(k)],[w y(k)])
            adjacency_matrix(j,k)=1;
            adjacency_matrix(k,j)=adjacency_matrix(j,k);
        else
            adjacency_matrix(j,k)=0;
            adjacency_matrix(k,j)=adjacency_matrix(j,k);
        end
    end
end
hold on
d;
adjacency_matrix;
a = adjacency_matrix;
% sender=input('Enter source node: ');
sen=sender;
plot(x(sen),y(sen),'o','MarkerEdgeColor','k',...
        'MarkerFaceColor','g','MarkerSize',15);
str=num2str(sen);
text((x(sen)-1),y(sen),str);
hold on

%==================================
% find all connected nodes
%==================================
ii=sender;
NODE_NUMBER=sender
```

```
adjm=adjacency_matrix;
connect % call connect.m
pause(1)
connection_original=connection;
% end of finding all connected nodes

%===================================
% broadcast JOIN_DATA message
%===================================
sender;
connection;
Parent=[];
if (sender<=n)
    msm_sender=1;
end
msm_i=msm_sender;
nodes=sender;
nodesparent=0;
Allmsm_i=msm_i;
Parent=sender;
fprintf('\n');
exParent=[];
AllChild=[];
JOIN_DATA=0;
warn=0;
while (length(Parent)~=0)   % when we still have a Parent
    for pr=1:length(Parent)
        Parent;
        jnow = Parent(pr);
        PR=jnow;
        %fprintf('*** New Parent = %g\n',PR);  % uncomment while observing
        Parent;
        exParent;
        pause(0.1)
        if (find(exParent==PR))
            Parent;
            pr;
            warn=1;
            Parent(pr)=[];
            pause(1)
            break
        end
        Children=[];
        connection(find(connection==PR))=[];
        for k=1:n   % finding the Children
            if (adjacency_matrix(jnow,k)==0)
                Children(k)=0;  % it is not connected or itself
            elseif (adjacency_matrix(jnow,k)==1)
                Children(k)=k;
            end
        end
        Children;
        Children(find(Children==0))=[];
        Children;  % eliminate semicolon while observing

        pr;
        PR;
        exParent(length(exParent)+1)=PR;
        %fprintf('-------\n');
        pause(0.1)
        % when the Parent has no Children
        if ((length(Children)==0)&(pr==length(Parent)))
            Parent=[];
            %pause(2)
            AllChild;
            if ((length(AllChild)~=0)&(length(connection)~=0))
                AllChild(find(AllChild==PR))=[];
                AllChild;
                Parent=AllChild;
            else
                Parent=[];
            end
        elseif (length(Children)~=0)   % when there are children
            while (length(Children)~=0)
                % distributing JOIN_DATA message to each node of the Children
                if (length(Children)~=0)
                    for ch = 1: length(Children)
                        xline=[x(PR) x(Children(ch))]';
                        yline=[y(PR) y(Children(ch))]';
                        plot_arrow(x(PR),y(PR),x(Children(ch)),...
                              y(Children(ch)),'headwidth',0.08,...
                               'headheight',0.15,'color',[1 0 0],'facecolor',[1 0 0]);
                        JOIN_DATA=JOIN_DATA+1;
                        nodes(length(nodes)+1)=Children(ch);
                        nodesparent(length(nodesparent)+1)=PR;
                        hold on
                    end
                    hold on
                    JOIN_DATA;  % eliminate semicolon while observing
                end
                exParent;
                %fprintf('eliminate exParent from Children\n')
                for ep=1:length(exParent)
                    Children(find(Children==exParent(ep)))=[];
                end
                if (pr==1)
                    AllChild=Children;
                else
                    %fprintf('eliminate Children & AllChild\n');
                    for ac=1:length(AllChild)
                        Children(find(Children==AllChild(ac)))=[];
                    end
                    AllChild;
                    AllChild(length(AllChild)+1:length(AllChild)+length(Children)
                    )=Children;
                end

                PR;
                % eliminating the Parent from the AllChild
                if (length(AllChild)~=0)
                    AllChild(find(AllChild==PR))=[];
                end
                AllChild;

                % end of one set of Parent and determining the new Parent
                if (pr==length(Parent))
                    Parent=AllChild;
                    AllChild=[];
                end
                exParent;
                Parent;
                connection;
                Children=[];
                if (length(connection)==0)
                    pr;
                    if (pr==length(Parent))
                        Parent=[];
                    end
                    break
                end
            end
        end
    end
end
nodes;
nodesparent;
JOIN_DATA;
pause(0.2)

%====================================
% create JOIN_TABLE and select route
%====================================
con_mesh=connection_original
child_mesh=[];
if (length(nodesparent)>1)
    while (length(con_mesh)~=0)
        co=nodes(1)
        con_mesh(find(con_mesh==co))=[];
        child_mesh=co;
        ortu=0
        pause(0.1)
        while (find(nodesparent==co))
            cf=find(nodesparent==co);
            child=nodes(cf);
            pause(0.1)
            if (length(child)~=0)
                for cho=1:length(child)
                    if (length(child_mesh)~=0)
                        if (find(child_mesh==child(cho)))
                            child(cho)=0;
                        end
                    end
                end
                child;
                child(find(child==0))=[];
            end
            child;
            for chod=1:length(child)
                con_mesh(find(con_mesh==child(chod)))=[];
            end
            add_child=length(child)+length(child_mesh);
            child_mesh(length(child_mesh)+1:add_child)=child;
            ortu(length(ortu)+1:add_child)=co;
            co=nodesparent((cf(length(cf)))+1);
            con_mesh;
            if (length(con_mesh)==0)
                break
            end
        end
    end
    child_mesh
    ortu
    pause(0.1)
    tablenode5 = [child_mesh;ortu];
    fprintf('-------------------------------------------------\n');
    disp('A table of node number and its previous hop');
    fprintf('-------------------------------------------------\n');
    fprintf('            %g        %g         \n',tablenode5);
    fprintf('-------------------------------------------------\n');
    NumberOfJOIN_DATA=JOIN_DATA;
elseif (length(nodesparent)<=1)
    JOIN__DATA=0;
end
NumberOfJOIN_DATA=JOIN_DATA;
fprintf('The total number of JOIN_DATA message = %g\n',NumberOfJOIN_DATA);
fprintf('\n');

pause(1)

%=========================
% broadcast JOIN_REPLY
%=========================
if (length(child_mesh)~=0)
    JOIN_REPLY=0;
    child_re=child_mesh;
    ortu_re=ortu;
    c_re=child_re(length(child_re));
    adjacency_matrix;
    pause(0.1)
    while (length(child_re)~=0)
        c_re;
        if (c_re~=sen)
            plot(x(c_re),y(c_re),'o','MarkerEdgeColor','k',...
                'MarkerFaceColor','y','MarkerSize',15);
            str=num2str(c_re);
            text((x(c_re)-1),y(c_re),str);
            hold on
        else
            fprintf('Already arrive at the Source\n');
            pause(1)
            break
        end
        for k=1:n
            if (adjacency_matrix(c_re,k)==0)
                neighbo(k)=0;
                elseif (adjacency_matrix(c_re,k)==1)
                    neighbo(k)=k;
                end
            end
            neighbo;
            neighbo(find(neighbo==0))=[];
            neighbo;
            prev_hop=ortu(find(child_mesh==c_re));
            if (length(neighbo)==0)
                fprintf('Node %g has no neighbo\n',c_re);
            else
                if (length(neighbo)~=0)
                    for chr=1:length(neighbo)
                        xline=[x(c_re) x(neighbo(chr))]';
                        yline=[y(c_re) y(neighbo(chr))]';
                        if (neighbo(chr)~=prev_hop)
                            plot_arrow(x(c_re),y(c_re),x(neighbo(chr)),y(neighbo(chr)),...
                                'headwidth',0.08,'headheight',0.15,'color',[0 1 0],...
                                'facecolor',[0 1 0]);
                        elseif (neighbo(chr)==prev_hop)
                            plot_arrow(x(c_re),y(c_re),x(neighbo(chr)),y(neighbo(chr)),...
                                'linewidth',3,'headwidth',0.08,'headheight',0.15,...
                                'color',[1 1 0],'facecolor',[1 1 0]);
                        end
                        JOIN_REPLY=JOIN_REPLY+1;
                        hold on
                    end
                    hold on
                    JOIN_REPLY;
                    receiver_o(length(receiver_o)+1)=c_re;
                    pause(0.1)
                end
            end
            c_re_keep=c_re;
            c_re=ortu_re(find(child_re==c_re_keep));
            find_o=find(ortu_re==c_re);
            child_re;
            if (length(find_o)>1)
                c_re=child_re(find_o);
                c_re(find(c_re==c_re_keep))=[];
            end
            ortu_re(find(child_re==c_re_keep))=[];
            child_re(find(child_re==c_re_keep))=[];
            if ((length(find_o)==1) & (c_re~=0))
                c_re=child_re(length(child_re))
            end
            if (length(c_re)>1)
                c_re2=c_re;
                c_re=[];
                for cre=1:length(c_re2)
                    c_re=c_re2(cre)
                    break
                end
            end
            %fprintf('================\n');
            pause(0.1)
            if (c_re==0)
                break
            end
        end
        JOIN_REPLY;
    elseif (length(child_mesh)==0)
        JOIN_REPLY=0;
    end
    NumberOfJOIN_REPLY=JOIN_REPLY
    allMessages_od=NumberOfJOIN_DATA+NumberOfJOIN_REPLY;
    allMessages_o(length(allMessages_o)+1)=allMessages_od
    receiver_o
    tablenode5=[];
```

```
% PLOT_ARROW
=================================================================================
Author: Ohad Gal
=================================================================================

function handles = plot_arrow( x1,y1,x2,y2,varargin )
%
% plot_arrow - plots an arrow to the current plot
%
% format:   handles = plot_arrow( x1,y1,x2,y2 [,options...] )
%
% input:    x1,y1   - starting point
%           x2,y2   - end point
%           options - come as pairs of "property","value" as defined for "line" and "patch"
%                      controls, see matlab help for listing of these properties.
%                      note that not all properties where added, one might add them at the
%                      end of this file.
%
%                     additional options are:
%                     'headwidth':  relative to complete arrow size, default value is 0.07
%                     'headheight': relative to complete arrow size, default value is 0.15
%                     (encoded are maximal values if pixels, for the case that the arrow is
%                      very long)
%
% output:   handles - handles of the graphical elements building the arrow
%
% Example:  plot_arrow( -1,-1,15,12,'linewidth',2,'color',[0.5 0.5 0.5],'facecolor',[0.5
%            0.5 0.5] );
%           plot_arrow( 0,0,5,4,'linewidth',2,'headwidth',0.25,'headheight',0.33 );
%           plot_arrow;   % will launch demo

% =============================================
% for debug - demo - can be erased
% =============================================
if (nargin==0)
    figure;
    axis;
    set( gca,'nextplot','add' );
    for x = 0:0.3:2*pi
        color = [rand rand rand];
        h = plot_arrow( 1,1,50*rand*cos(x),50*rand*sin(x),...
            'color',color,'facecolor',color,'edgecolor',color );
        set( h,'linewidth',2 );
    end
    hold off;
    return
end
% =============================================
% end of for debug
% =============================================


% =============================================
% constants (can be edited)
% =============================================
alpha       = 0.15;   % head length
beta        = 0.07;   % head width
max_length  = 22;
max_width   = 10;

% =============================================
% check if head properties are given
% =============================================
% if ratio is always fixed, this section can be removed!
if ~isempty( varargin )
    for c = 1:floor(length(varargin)/2)
        try
            switch lower(varargin{c*2-1})
                % head properties - do nothing, since handled above already
            case 'headheight',alpha = max( min( varargin{c*2},1 ),0.01 );
            case 'headwidth', beta = max( min( varargin{c*2},1 ),0.01 );
            end
        catch
            fprintf( 'unrecognized property or value for: %s\n',varargin{c*2-1} );
        end
    end
end

% ============================================
% calculate the arrow head coordinates
% ============================================
den         = x2 - x1 + eps;                                % make sure no devision by zero
occurs
teta        = atan( (y2-y1)/den ) + pi*(x2<x1) - pi/2;      % angle of arrow
cs          = cos(teta);                                    % rotation matrix
ss          = sin(teta);
R           = [cs -ss;ss cs];
line_length = sqrt( (y2-y1)^2 + (x2-x1)^2 );                % sizes
head_length = min( line_length*alpha,max_length );
head_width  = min( line_length*beta,max_length );
x0          = x2*cs + y2*ss;                                % build head coordinats
y0          = -x2*ss + y2*cs;
coords      = R*[x0 x0+head_width/2 x0-head_width/2; y0 y0-head_length y0-head_length];

% ============================================
% plot arrow  (= line + patch of a triangle)
% ============================================
h1          = plot( [x1,x2],[y1,y2],'k' );
h2          = patch( coords(1,:),coords(2,:),[0 0 0] );

% ============================================
% return handles
% ============================================
handles = [h1 h2];

% ============================================
% check if styling is required
% ============================================
% if no styling, this section can be removed!
if ~isempty( varargin )
    for c = 1:floor(length(varargin)/2)
        try
            switch lower(varargin{c*2-1})

             % only patch properties
            case 'edgecolor',   set( h2,'EdgeColor',varargin{c*2} );
            case 'facecolor',   set( h2,'FaceColor',varargin{c*2} );
            case 'facelighting',set( h2,'FaceLighting',varargin{c*2} );
            case 'edgelighting',set( h2,'EdgeLighting',varargin{c*2} );

            % only line properties
            case 'color'    , set( h1,'Color',varargin{c*2} );

            % shared properties
            case 'linestyle', set( handles,'LineStyle',varargin{c*2} );
            case 'linewidth', set( handles,'LineWidth',varargin{c*2} );
            case 'parent',    set( handles,'parent',varargin{c*2} );

            % head properties - do nothing, since handled above already
            case 'headwidth',;
            case 'headheight',;

            end
        catch
            fprintf( 'unrecognized property or value for: %s\n',varargin{c*2-1} );
        end
    end
end

==============================================================================
(Source: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3345)
==============================================================================
% REVERSE

function b=reverse(a)
% gives the vector in reverse order.
p=length(a);
for c=1:p;
   b(c)=a(p-c+1);
end;
```

```
% SMALLERID is to find a smaller id
%
% Author: A. Radyastuti
% v6.5 19-Jun-04

nodewith_id_origin;
msm_id_origin;
finalparent2=finalparent;
finalparent=[];
for fp2=1:length(finalparent2)
    FF=find(nodewith_id_origin==finalparent2(fp2));
    id_parent(fp2)=msm_id_origin(FF);
end
finalparent2;
id_parent;
[id_parent,IND]=sort(id_parent)
finalparent2([IND]);
finalparent2=finalparent2([IND])
id_parent=[];
pause(0.5)
joinreqmany % call joinreqmany.m
```