An Optimized Polymorphic Hybrid Multicast Routing Protocol for MANET's

P-2171

A PROJECT REPORT

Submitted by

Revathi R	Reg.No:	71204205038
Saraswathi P	Reg.No:	71204205044

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2008

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"An Optimized Polymorphic Hybrid Multicast Routing Protocol for MANET's"** is the bonafide work of **"Revathi R and Saraswathi P"** who carried out the project work under my supervision.

SIGNATURE

Dr.S.ThangasamyB.E(HONS)., Phd

HEAD OF THE DEPARTMENT

Dept of Information Technology,
Kumaraguru College of Technology,
Coimbatore – 641 006.

SIGNATURE

Prof.K.R.Baskaran B.E., M.S.

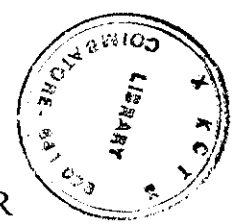
SUPERVISORS

Dept of Information Technology,
Kumaraguru College of Technology,
Coimbatore – 641 006.

The candidates with University Register No **71204205038** and **71204205044** were examined by us in the project viva-voice examination held on ...24.04.08.

INTENAL EXAMINER

EXTERNAL EXAMINER



P 2171

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc, F.I.E** and Vice Chairman **Prof. K. Arumugam B.E., M.S., M.I.E.,** for all their support and ray of strengthening hope extended. We are immensely grateful to our principal **Dr. Joseph V. Thanikal M.E., Ph.D., PDF., CEPIT.,** for his invaluable support to the outcome of this project.

We are deeply obliged to **Dr.S.Thangasamy**, Head of the department of Information Technology for his valuable guidance and useful suggestions during course of this project.

We also extend our heartfelt thanks to our project coordinator **Prof.K.R.Baskaran B.E., M.S.,** Department of Information Technology for providing us his support which really helped us.

We are indebted to our project guide **Prof.K.R.Baskaran B.E., M.S.,** Department of Information Technology for her helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support during the course of this project. We also thank all of our friends who helped us to complete this project successfully.

DECLARATION

We,

Revathi R	Reg.No:	71204205038
Saraswathi P	Reg.No:	71204205044

hereby declare that the project entitled **"An Optimized Polymorphic Hybrid Multicast Routing Protocol** for MANET's", submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) degree, is a record of original work done by us under the supervision and guidance of Department of Information Technology, Kumaraguru college of Technology, Coimbatore.

Place: Coimbatore

Date: 22.04.08

R. Revathi P. Saraswathi

[Revathi R] [Saraswathi P]

Project Guided by

[Prof.K.R.Baskaran B.E., M.S.]

ABSTRACT

We propose in this paper an optimized, polymorphic, hybrid multicast routing protocol for MANET. This new polymorphic protocol attempts to benefit from the high efficiency of proactive behavior (in terms of quicker response to transmission requests) and the limited network traffic overhead of the reactive behavior, while being power, mobility, and vicinity-density (in terms of number of neighbor nodes per specified area around a mobile node) aware. The proposed protocol is based on the principle of adaptability and multibehavioral modes of operations. It is able to change behavior in different situations in order to improve certain metrics like maximizing battery life, reducing communication delays, improving deliverability, etc. The protocol is augmented by an optimization scheme, adapted from the one proposed for the Optimized Link State Routing protocol (OLSR) in which only selected neighbor nodes propagate control packets to reduce the amount of control overhead. Extensive simulations and comparison to peer protocols demonstrated the effectiveness of the proposed protocol in improving performance and in extending battery power longevity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

OPHMR	-	An Optimized Polymorphic Hybrid Multicast Routing Protocol for MANET's
GLOMOSIM	-	GLObal MObile Information System SIMulator
PARSEC	-	PARallel Simulation Environment for Complex systems
MPR	-	MultiPoint Relay
MANET	-	Mobile Ad-hoc NETwork

1.INTRODUCTION

1.1 GENERAL

Mobile ad hoc networks (MANETs) are infrastructure free networks of mobile nodes that communicate with each other wirelessly. There are several routing schemes that have been proposed and several of these have been extensively simulated or completely implemented as well. The primary applications of such networks have been in disaster relief operations, military use, conferencing and environment sensing. Unlike conventional wireless networks one may find in offices, universities, communities or homes there is no central entity that controls how, when and where, packets are delivered to each recipient. All communication takes place in an ad hoc manner, which means on the fly and all the nodes in the network participate in relaying packets or messages to each other whenever it is possible for each node to do so.

There are several unicast routing algorithms that have been developed for MANETs that have their own unique characteristic strengths and weaknesses. Most protocols can be classified in several ways. Some are classified as reactive or on- demand while others are proactive. In general, a proactive protocol finds routes in advance while a reactive protocol finds routes to the destination only when it absolutely must. For example, Ad hoc On demand Distance Vector routing (AODV) **[AODV]** is an on-demand protocol since no protocol information is transmitted before an application decides to send data and no data is sent until a route is formed, whereas Destination Sequenced Distance Vector protocol (DSDV) **[DSDV]** is a more proactive protocol in which routes are discovered and stored even before they are needed.

Proactive protocols generally generate much more traffic than on-demand protocols. A third general category is a hybrid algorithm that effectively combines multiple characteristics in a unique and meaningful way. For example, the Zone Routing Protocol (ZRP) [ZRP] is a hybrid protocol that combines local proactive routing with a globally reactive routing strategy.

1.2 PROBLEM DEFINITION

This new polymorphic protocol attempts to benefit from the high efficiency of proactive behavior (in terms of quicker response to transmission requests) and the limited network traffic overhead of the reactive behavior, while being power, mobility, and vicinity-density (in terms of number of neighbor nodes per specified area around a mobile node) aware. The proposed protocol is based on the principle of adaptability and multibehavioral modes of operations. It is able to change behavior in different situations in order to improve certain metrics like maximizing battery life, reducing communication delays, improving deliverability, etc. The protocol is augmented by an optimization scheme, adapted from the one proposed for the Optimized Link State Routing protocol (OLSR) in which only selected neighbor nodes propagate control packets to reduce the amount of control overhead. Extensive simulations and comparison to peer protocols demonstrated the effectiveness of the proposed protocol in improving performance and in extending battery power longevity.

1.3 OBJECTIVE OF THE PROJECT

The simple hybrid routing protocol is built on the principle that the mobile node is able to behave either proactively or reactively under different conditions. As such, the resulting protocol inherits the benefits of both reactive and proactive behaviors and, hence, can expand its suitability region within the ad hoc network design space.

The distance of the destination nodes from the source nodes are calculated and based on that, the mode of transmission (either proactive or reactive) is chosen.In case of short distance transmission ,the reactive mode is used and in long distance transmission proactive mode is used.

In the OPHMR protocol, each node maintains a two-hop Neighborhood Table (2NTable). The 2NTable is used to calculate the MPR information. When a node receives an update packet, it uses the neighborhood information in the packet to calculate the two-hop neighborhood and updates the corresponding entries in the 2NTable.

MPR nodes are selected to forward broadcast messages during the flooding process. This technique substantially reduces the message overhead as compared to the classical flooding mechanism, where every node retransmits each message when it receives it for the first time.Each node has its MPR set and will broadcast its MPR information in the periodic update packets. When propagating the periodic update packets, only the MPRs forward update packets.

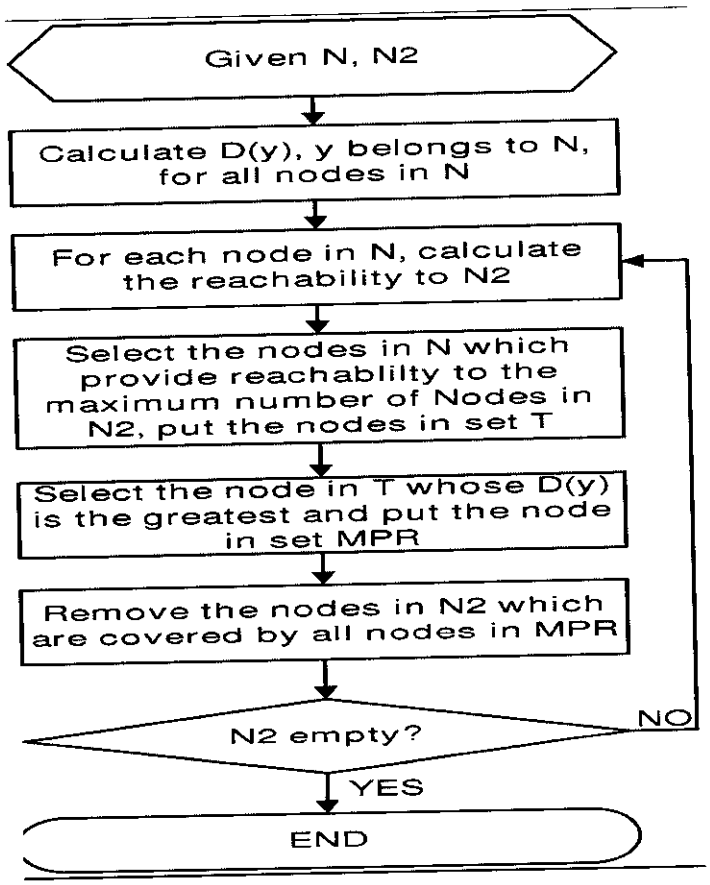


Fig 1.3.1 Flowchart for the MPR operation.

2. LITERATURE REVIEW

2.1 FEASIBILITY STUDY

2.1.1 CURRENT STATUS OF THE PROBLEM

Proactive protocols continuously exchange network topology information so as to constantly monitor topology changes and use that knowledge for efficient, low latency data transmission. While in reactive protocols nodes react only on-demand to data transmission requests and perform path finding operations only when needed.

Limitations

The network topology of ad hoc networks is unstable and changes frequently with nodes' mobility, traditional routing protocols in static networks are not efficient for ad hoc networks.

2.1.2 PROPOSED SYSTEM

The simple hybrid routing protocol is built on the principle that the mobile node is able to behave either proactively or reactively under different conditions. As such, the resulting protocol inherits the benefits of both reactive and proactive behaviors and, hence, can expand its suitability region within the ad hoc network design space.

OPHMR is built using the proactive behavior of the MZR protocol and the reactive behavior of ODMRP. In addition, the protocol was augmented with the Multipoint Relay (MPR)-based optimization mechanism of OLSR [4]. This is done with the objective of reducing the number of control packets forwarded. Next, we describe its proactive and reactive behaviors as well as the adopted optimized forwarding mechanism.

PROACTIVE BEHAVIOR. When a node is in PM1 or PM2, it periodically sends out update packets which have a TTL set to the Zone Radius. When a node receives an update packet, if it is in PM1, in PM2, or in PRM, it saves the information in the packet into the one-hop Neighborhood Table (NTable), reduces the TTL by 1, and forwards the packet.

REACTIVE BEHAVIOR. When a node has packets to send to a multicast group or wants to join a multicast group, it sends out a Join_Request packet and waits for its Join_Reply from a destination node. Only nodes that belong to the destination multicast group will send out Join_Replies, and such nodes will update their Multicast Routing Tables (MRTable) to maintain the multicast group information. If an intermediate node (including the source node) has entries in its NTable that belong to the destination multicast group, it unicasts the join_Request to the nodes registered in those entries. When the source node receives a Join_Reply, it updates its MRTable and begins data transmission.

ROUTING TABLES: Each node maintains the two routing tables, the NTable and the MRTable. The NTable acts as the neighborhood routing table we described in the algorithm and, actually, only nodes in Proactive Modes maintain it. Each neighbor in the zone has an entry in this table. Each such an entry contains the routing information to the node, including hop count and next hop address. In addition, it contains the multicast routing information of the node, such as the multicast group that the node belongs to. Each entry is assigned a lifetime and the one that exceeds its predefined lifetime is removed from the NTable. Nodes in a PM or a PRM mode maintain the NTable using the update packets they receive from others. Nodes in those modes also periodically flush the NTable to remove stale entries. Nodes in RM mode just periodically flush their NTables to remove

stale entries, but will not update or insert any entries in them. Each node also uses its MRTable to maintain both its multicast routing information and the multicast routing topology. The structure of the MRTable is the same as the routing table used for ODMRP [32].

PACKET STRUCTURE. The structure of zone UPDATE packets used in the OPHMR protocol is shown in Fig. 1. In this table, the type is set to 4 to indicate that the packet is for ZONE UPDATE. The field "Reserved" is used for future use. The field "Time To Live" has the same classical usage. When the source node generates a ZONE UPDATE packet, it sets its Time To Live to the Zone_Radius. Its value is decremented by one at each hop until it becomes zero and it is then discarded. The field "Number of Multicast Groups" indicates how many multicast groups this node belongs to. The field "Source Node IP Address" stores the IP address of the source node. Finally, each "Multicast Group IP Address" field describes information on a particular multicast group that the source node belongs to. The second type of packet we describe here is the notification packet. Its structure is very simple. The first eight bits are for the packet type and are set to 5 to indicate that it is a Notification Packet for OPHMR. The next 16 bits are for the switch type and each bit is for one type (there are 12 types of switching, so the first four bits are always set to 0). The last eight bits are reserved. Table 1 shows the meaning of each bit.

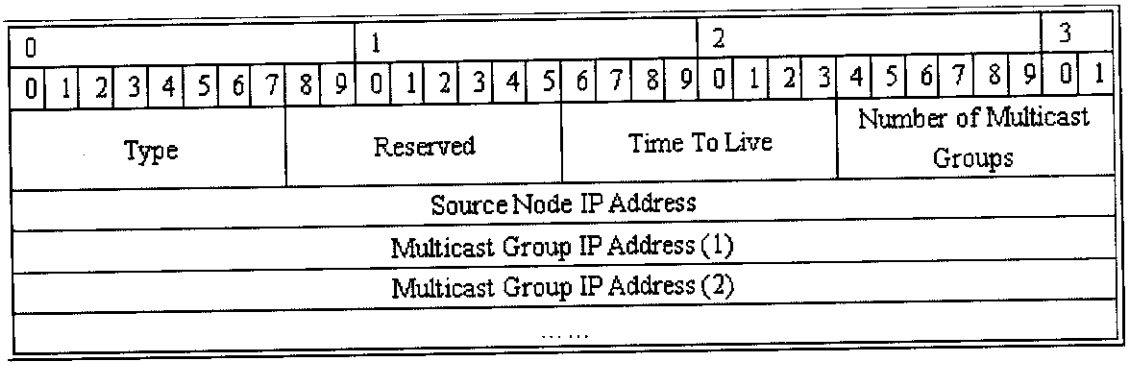


Fig 2.1.2.1: Packet Structure of OPHMR

Advantages:

The protocol attempts to benefit from the high efficiency of proactive routing in reducing response time to transmission requests and from the reduced control overhead offered by reactive routing.

2.2 HARDWARE SPECIFICATION

Processor : Intel Pentium IV 2.2 GHZ

Hard Disk Drive : 20 GB.

RAM : 256 MB RAM minimum.

2.3 SOFTWARE REQUIREMENTS

Operating System : Windows 2000/XP

Software : Glomosim Simulator

2.4 SOFTWARE OVERVIEW

2.4.1 GLOMOSIM

Our project has been done using a scalable simulation environment called GloMoSim (for Global Mobile Information System Simulator) that effectively utilizes parallel execution to reduce the simulation time of detailed high-fidelity models of large communication networks. GloMoSim has been designed to be extensible and composable: the communication protocol stack for wireless networks is divided into a set of layers, each with its own API. Models of protocols at one layer interact with those at a lower (or higher) layer only via these APIs. The modular implementation enables consistent comparison of multiple protocols at a given layer. The parallel implementation of GloMoSim can be executed using a variety of conservative synchronization protocols, which include the null message and conditional event algorithms.

2.4.2 PARSEC

PARSEC (for PARallel Simulation Environment for Complex systems) is a C-based simulation language developed by the Parallel Computing Laboratory at UCLA, for sequential and parallel execution of discrete-event simulation models. It can also be used as a parallel programming language. PARSEC runs on several platforms, including most recent UNIX variants as well as Windows. PARSEC adopts the process interaction approach to discrete-event simulation. An object (also referred to as a physical process) or set of objects in the physical system is represented by a logical process. Interactions among physical processes (events) are modeled by time-stamped message exchanges among the corresponding logical processes. One of the important distinguishing features of PARSEC

is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it. Thus, with few modifications, a PARSEC program may be executed using the traditional sequential (Global Event List) simulation protocol or one of many parallel optimistic or conservative protocols. In addition, PARSEC provides powerful message receiving constructs that result in shorter and more natural simulation programs.

2.4.3 ABOUT GLOMOSIM

GloMoSim is a mobile simulator built using C language. All message transfers and other network elements are handled by the individual layer coding built in C. To make the concepts clear, GloMoSim provides users with a Visualization Tool (VT) built using Java. The VT helps us understand the network environment, the node positions, message transfers, clustering details, etc.

2.4.4 GLOMOSIM ARCHITECTURE

The networking stack is decomposed into a number of layers as shown in Figure 1. A number of protocols have been developed at each layer and models of these protocols or layers can be developed at different levels of granularity.

In our project we deal with all these layers, but most of the coding has been implemented in the Clustering and Routing layers. The dynamic clustering algorithm explained in chapters 2 and 3 is implemented in these two layers. The clustering algorithm has been fully implemented in the

Clustering layer. The coding present in the Routing layer has been largely modified according to the cluster formation. The algorithm used for routing is the Bellman-Ford algorithm. This algorithm maintains a routing table for every node being simulated. We have modified the entries of this table based on the cluster formation which is dynamic.

The communication between the various layers is accomplished by means of the various APIs available. A common API between two layers helps in the communication between those two layers.

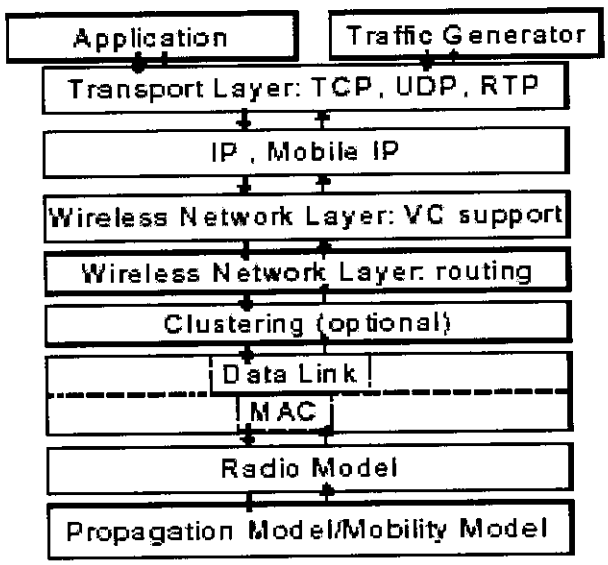


Fig.2.4.4.1. GloMoSim Architecture

2.4.5 GLOMOSIM LIBRARY

GloMoSim is a scalable simulation library for wireless network systems built using the PARSEC simulation environment. Table 1 lists the GloMoSim models currently available at each of the major layers. GloMoSim also supports two different node mobility models. Nodes can move according to a model that is generally referred to as the "random waypoint" model. A node chooses a random destination within the simulated terrain and moves to that location based on the speed specified in the configuration file. After reaching its destination, the node pauses for a duration that is also specified in the configuration file. The other mobility model in GloMoSim is referred to as the "random drunken" model. A node periodically moves to a position chosen randomly from its immediate neighboring positions. The frequency of the change in node position is based on a parameter specified in the configuration file.

Layer:	Models:
Physical (Radio propagation)	Free space, Rayleigh, Ricean, SIRCIM
Data Link (MAC)	CSMA, MACA, MACAW, FAMA, 802.11
Network (Routing)	Flooding, Bellman-Ford, OSPF, DSR, WRP
Transport	TCP, UDP
Application	Telnet, FTP

Table 2.4.5.1: Models currently in the GloMoSim library.

3. DETAILS OF METHODOLOGY EMPLOYED

3.1 ALGORITHMS

Algorithm 1

POLYMORPHIC ALGORITHM

if Power > P TH1 then

if the MN is not in PM1, it switches to PM1.

then it notifies neighbors about the mode switch.

else

if Power < P TH2 then

if the MN is not in RM, it switches to RM.

then it notifies neighbors about the mode switch.

else

Perform the mobility speed routine.

end if

end if

Algorithm 2

MOBILITY SPEED ROUTINE

if Mobility > M TH then

if Vicinity < V TH then

if the MN is not in PM2, it switches to PM2.

then it notifies neighbors about the mode switch.

else

if the MN is not in PRM switches to PRM.

then it notifies neighbors about the mode switch.

end if

else

if the MN is not in RM switches to RM.

then it notifies neighbors about the mode switch.

end if

3.2. Path Finding Procedure

When a node has packets to send to a multicast group or wants to join that group, it begins the path finding procedure. If it is in RM mode, it sends out a Join_Request the way it is done in ODMRP and waits for replies.If the node is in proactive mode or proactive ready mode, it first looks in its NTable to see whether there are nodes that belong to the destination multicast group. If so, it unicasts Join_Requests to all these nodes and waits for replies. Otherwise, the node will broadcast a Join_Request. When a node receives a Join_Request and it is a member of the multicast group, it generates a reply and sends it back to the source of the Join_Request, updating the MRTable to record the route. If the node could not send a reply, it checks its own behavior. If it is in a reactive mode, it just propagates the Join_Request and records it in the route cache. If the node is in a proactive mode or in the PRM, it looks in its own NTable to find the destination multicast group member. If there are members in its zone, it unicasts the Join_Request to all of them. If not, it just propagates the Join_Request. When the source node receives a reply, it updates its MRTable to record the route and begins data transmission.

4. PERFORMANCE EVALUATION

4.1 SIMULATION ENVIRONMENT

An extensive simulations are conducted to evaluate the performance of OPHMR. The simulations were implemented on GloMoSim 2.03, a scalable simulation environment for wireless network systems. GloMoSim uses the parallel discrete-event simulation capability provided by PARSEC.

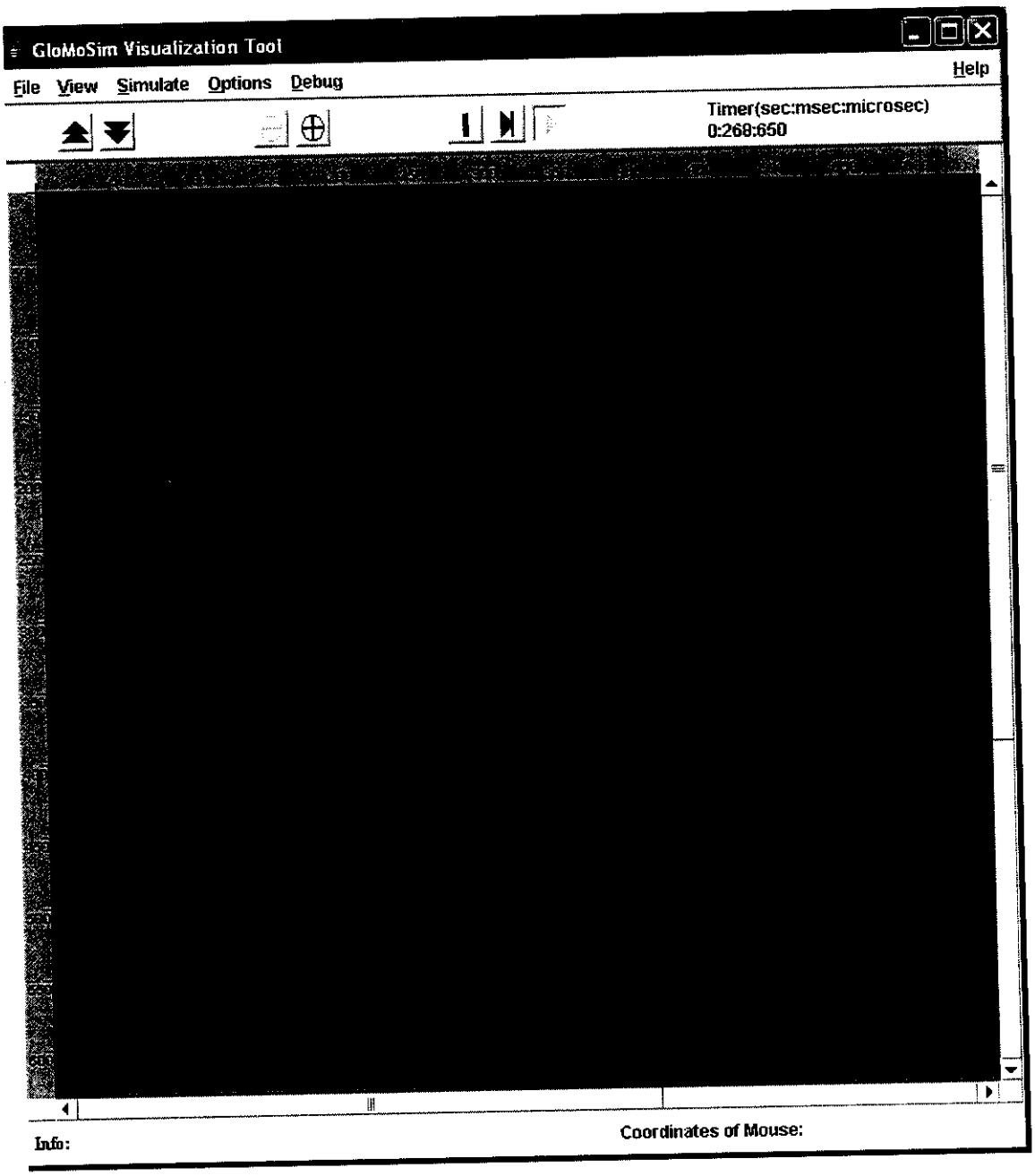
The simulation parameters used are as follows:

PARAMETERS	VALUES
Network size	49nodes
Radio Range	250m
Packet size	512bytes
Terrain-Dimensions	(500,500)
Node-Placement	Uniform
GUI Option	YES
Routing Protocols	OPHMR

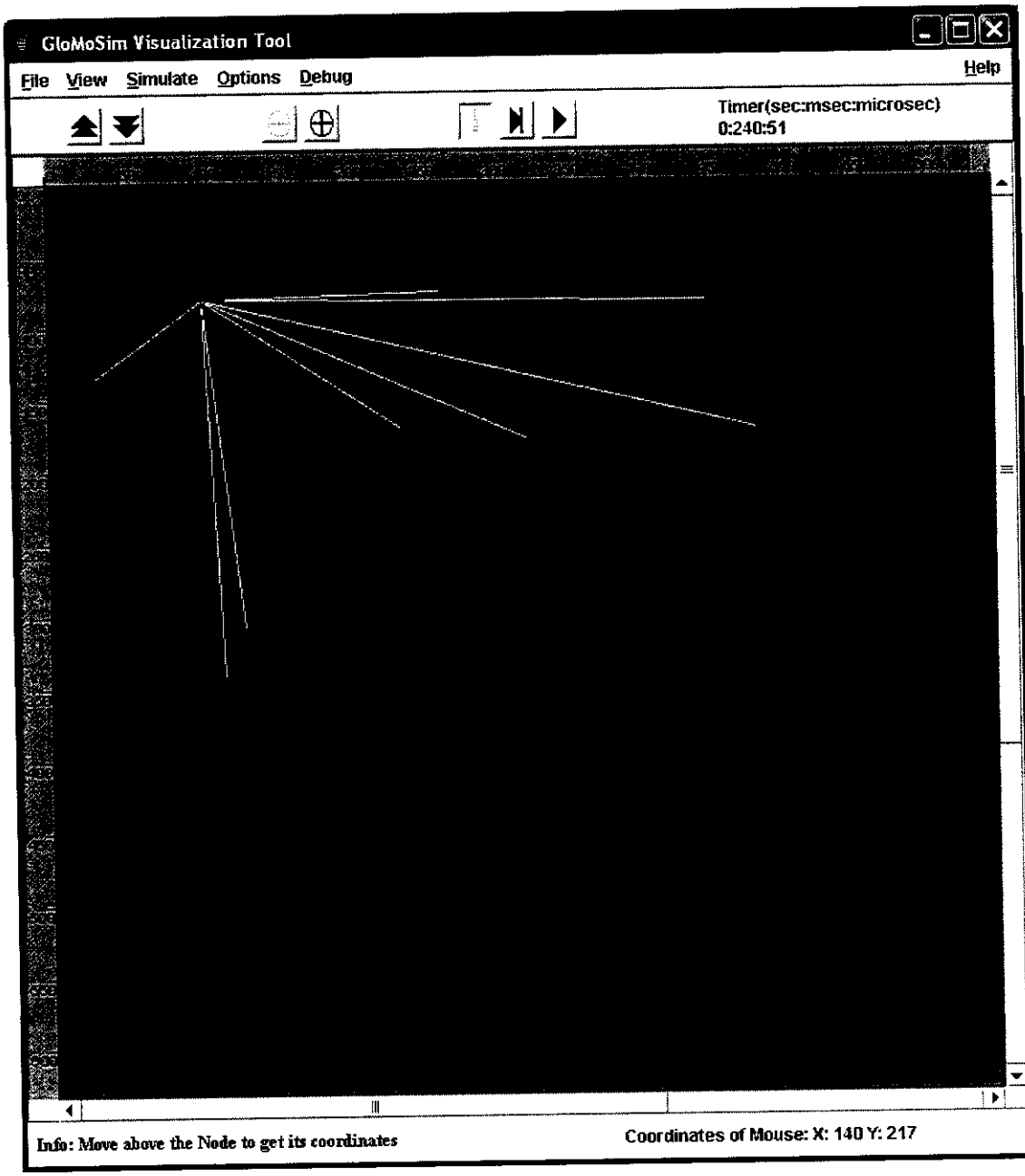
Table.4.1.1 Simulation Parameters

SCREEN SHOTS

1. NODE PLACEMENT

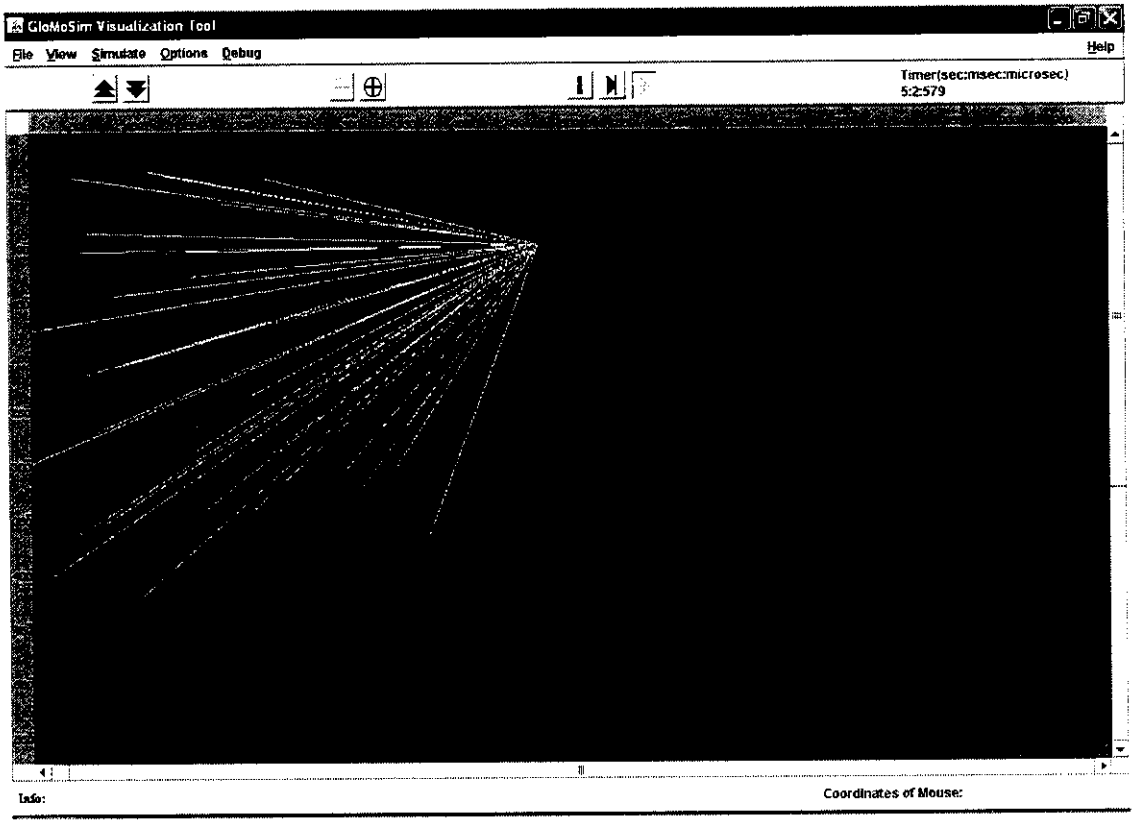


2. CONNECTION ESTABLISHMENT

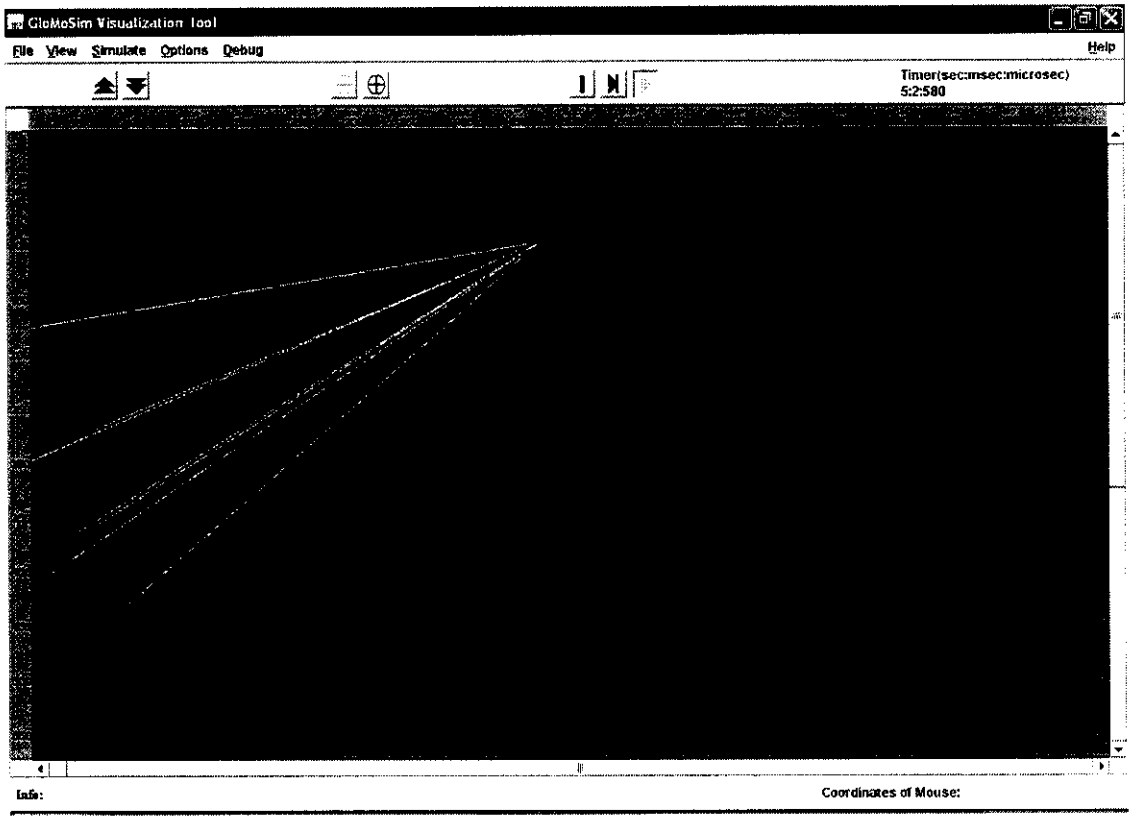


4.2 SIMULATION RESULTS

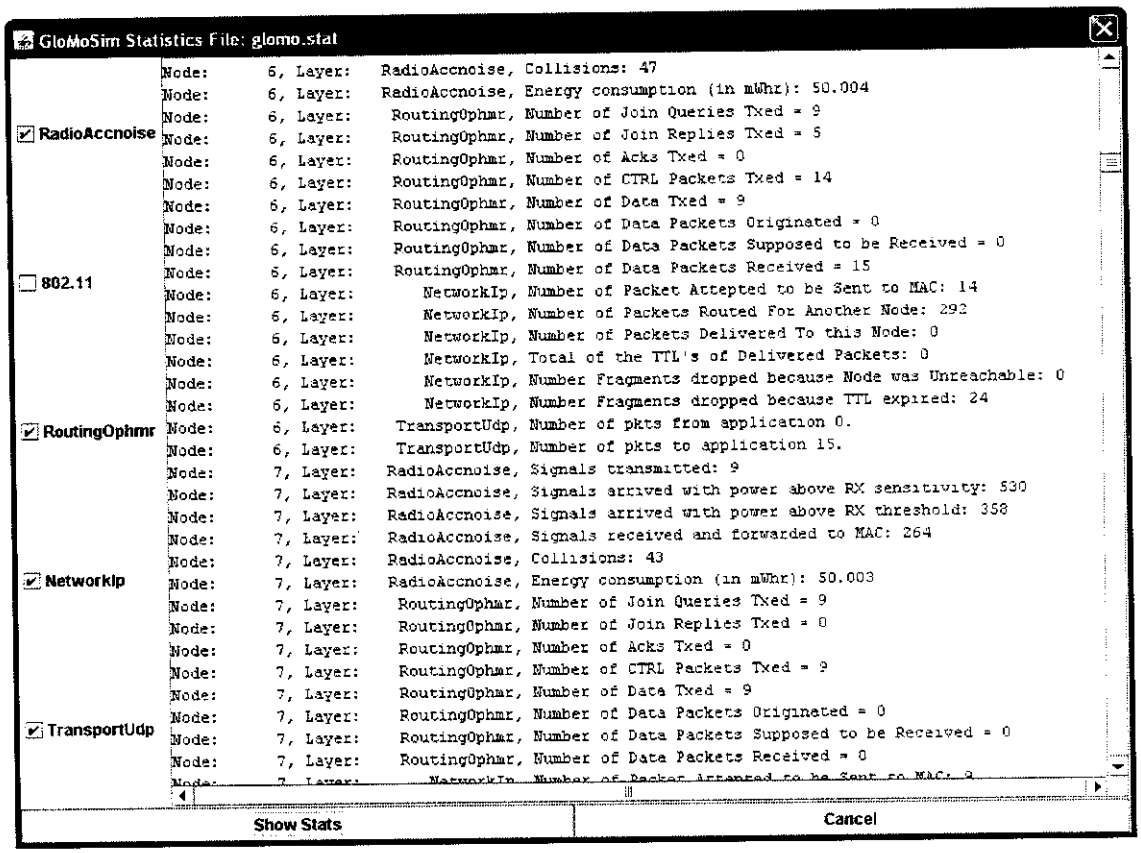
SHORT DISTANCE TRANSMISSION(REACTIVE)



LONG DISTANCE TRANSMISSION(PROACTIVE)



STATISTICS:


GloMoSim Statistics File: glomo.stat
RadioAccnoise
802.11
RoutingOphmr
NetworkIp
TransportUdp
Node: 6, Layer: RadioAccnoise, Collisions: 47
Node: 6, Layer: RadioAccnoise, Energy consumption (in mWhr): 50.004
Node: 6, Layer: RoutingOphmr, Number of Join Queries Txed = 9
Node: 6, Layer: RoutingOphmr, Number of Join Replies Txed = 5
Node: 6, Layer: RoutingOphmr, Number of Acks Txed = 0
Node: 6, Layer: RoutingOphmr, Number of CTRL Packets Txed = 14
Node: 6, Layer: RoutingOphmr, Number of Data Txed = 9
Node: 6, Layer: RoutingOphmr, Number of Data Packets Originated = 0
Node: 6, Layer: RoutingOphmr, Number of Data Packets Supposed to be Received = 0
Node: 6, Layer: RoutingOphmr, Number of Data Packets Received = 15
Node: 6, Layer: NetworkIp, Number of Packet Attepted to be Sent to MAC: 14
Node: 6, Layer: NetworkIp, Number of Packets Routed For Another Node: 292
Node: 6, Layer: NetworkIp, Number of Packets Delivered To this Node: 0
Node: 6, Layer: NetworkIp, Total of the TTL's of Delivered Packets: 0
Node: 6, Layer: NetworkIp, Number Fragments dropped because Node was Unreachable: 0
Node: 6, Layer: NetworkIp, Number Fragments dropped because TTL expired: 24
Node: 6, Layer: TransportUdp, Number of pkts from application 0.
Node: 6, Layer: TransportUdp, Number of pkts to application 15.
Node: 7, Layer: RadioAccnoise, Signals transmitted: 9
Node: 7, Layer: RadioAccnoise, Signals arrived with power above RX sensitivity: 530
Node: 7, Layer: RadioAccnoise, Signals arrived with power above RX threshold: 358
Node: 7, Layer: RadioAccnoise, Signals received and forwarded to MAC: 264
Node: 7, Layer: RadioAccnoise, Collisions: 43
Node: 7, Layer: RadioAccnoise, Energy consumption (in mWhr): 50.003
Node: 7, Layer: RoutingOphmr, Number of Join Queries Txed = 9
Node: 7, Layer: RoutingOphmr, Number of Join Replies Txed = 0
Node: 7, Layer: RoutingOphmr, Number of Acks Txed = 0
Node: 7, Layer: RoutingOphmr, Number of CTRL Packets Txed = 9
Node: 7, Layer: RoutingOphmr, Number of Data Txed = 9
Node: 7, Layer: RoutingOphmr, Number of Data Packets Originated = 0
Node: 7, Layer: RoutingOphmr, Number of Data Packets Supposed to be Received = 0
Node: 7, Layer: RoutingOphmr, Number of Data Packets Received = 0
Show Stats
Cancel

5.CONCLUSION

The protocol design is a novel way of combining three dimensions in protocol design, namely, hybridity, adaptability, and power awareness. With regard to hybridity, The protocol attempts to benefit from the high efficiency of proactive routing in reducing response time to transmission requests and from the reduced control overhead offered by reactive routing.

6.FUTURE ENHANCEMENT

The project has been further enhanced to include the mechanisms for the battery power calculation of every nodes and can change the mode of transmission .This new concept of polymorphic protocols constitutes the next trend in the design of efficient multibehavioral routing protocols for wireless, power-constrained networks such as MANETs.

APPENDICES

SOURCE CODE:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "api.h"
#include "structmsg.h"
#include "fileio.h"
#include "main.h"
#include "message.h"
#include "network.h"
#include "ip.h"
#include "nwip.h"
#include "nwcommon.h"
#include "application.h"
#include "transport.h"
#include "java_gui.h"

#include "mac.h"
#include "ophmr.h"

#define noDEBUG

void RoutingOphmrInit(GlomoNode *node,
                      GlomoRoutingOphmr **ophmrPtr,
                      const GlomoNodeInput *nodeInput)
{
    GlomoNodeInput memberInput;
    BOOL retVal;
    int i;

    char joinTimeStr[GLOMO_MAX_STRING_LENGTH];
    char leaveTimeStr[GLOMO_MAX_STRING_LENGTH];
    clocktype joinTime, leaveTime;
    NODE_ADDR srcAddr, mcastAddr;

    GlomoRoutingOphmr *ophmr =
```

```
    (GlomoRoutingOphmr *)checked_pc_malloc
  (sizeof(GlomoRoutingOphmr));

  (*ophmrPtr) = ophmr;

  if (ophmr == NULL)
  {
     fprintf(stderr, "OPHMR: Cannot alloc memory for OPHMR
     struct!\n");
    assert (FALSE);
  }

  NetworkIpSetRouterFunction(node, &RoutingOphmrRouterFunction);

  RoutingOphmrInitStats(node);
  RoutingOphmrInitMembership(&ophmr->memberFlag);
  RoutingOphmrInitFgFlag(&ophmr->fgFlag);
  RoutingOphmrInitMemberTable(&ophmr->memberTable);
  RoutingOphmrInitTempTable(&ophmr->tempTable);
  RoutingOphmrInitRouteTable(&ophmr->routeTable);
  RoutingOphmrInitMessageCache(&ophmr->messageCache);
  RoutingOphmrInitSeqTable(node);
  RoutingOphmrInitSent(&ophmr->sentTable);
  RoutingOphmrInitAckTable(&ophmr->ackTable);
  RoutingOphmrInitResponseTable(&ophmr->responseTable);

  retVal = GLOMO_ReadCachedFile(nodeInput, "MCAST-CONFIG-
FILE",
  &memberInput);

  if (retVal == FALSE)
  {
     fprintf(stderr, "OPHMR: Needs MCAST-CONFIG-FILE.\n");
     assert(FALSE);
  }

  for (i = 0; i < memberInput.numLines; i++)
  {
     retVal = sscanf(memberInput.inputStrings[i],
          "%ld %ld %s %s",&srcAddr, &mcastAddr,
```

```
        joinTimeStr, leaveTimeStr);


  if (retVal != 4)
  {
     fprintf(stderr, "Application: Wrong configuration
     format!\n");
     assert(0); abort();
  }

  joinTime = GLOMO_ConvertToClock(joinTimeStr);
  leaveTime = GLOMO_ConvertToClock(leaveTimeStr);

  if (node->nodeAddr == srcAddr)
  {
     RoutingOphmrSetTimer( node, MSG_NETWORK_JoinGroup,
     mcastAddr, joinTime);
     RoutingOphmrSetTimer(
     node, MSG_NETWORK_LeaveGroup, mcastAddr, leaveTime);
  }
}
void RoutingOphmrHandleProtocolEvent(GlomoNode *node, Message
  *msg)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node->
      networkData.networkVar;
  GlomoRoutingOphmr* ophmr= (GlomoRoutingOphmr *) ipLayer->
      routingProtocol;
  char clockStr[GLOMO_MAX_STRING_LENGTH];

  switch (msg->eventType)
    {
    case MSG_NETWORK_CheckAcked:
    {
      NODE_ADDR *mcastAddr = (NODE_ADDR
            *)GLOMO_MsgReturnInfo(msg);
      clocktype jitterTime;

      if (RoutingOphmrCheckAckTable(*mcastAddr, &ophmr->
          ackTable))
```

```
  {
    #ifdef DEBUG
    printf("Node %ld not acked. Retx!\n", node->nodeAddr);
    #endif
    RoutingOphmrRetxReply(node,*mcastAddr,&ophmr->ackTable);
  }

  GLOMO_MsgFree(node, msg);

  break;
}

case MSG_NETWORK_CheckTimeoutAlarm:
{
  NODE_ADDR *mcastAddr=(NODE_ADDR
      *)GLOMO_MsgReturnInfo(msg);
  if(RoutinOphmrLookupSentTable(*mcastAdr,&ophmr->sentTable))
  {
     if (RoutingOphmrCheckMinExpTime(
        *mcastAddr, &ophmr->sentTable) &&
        RoutingOphmrCheckCongestionTime(
        *mcastAddr, &ophmr->memberTable) &&
        !RoutingOphmrCheckSendQuery(*mcastAddr,&ophmr->
                sentTable))
     {
        #ifdef DEBUG
        printf("HERE\n");
        #endif
        RoutingOphmrSetSendQuery(*mcastAddr,&ophmr->
                  sentTable);
     }

     RoutingOphmrSetTimer(node,
            MSG_NETWORK_CheckTimeoutAlarm,
      *mcastAddr, OPHMR_TIMER_INTERVAL);
   }
   GLOMO_MsgFree(node, msg);
   break;
  }
case MSG_NETWORK_JoinGroup:
```

```
{
 NODE_ADDR *mcastAddr = (NODE_ADDR*)
                GLOMO_MsgReturnInfo(msg);
 RoutingOphmrJoinGroup(node, *mcastAddr);
 GLOMO_MsgFree(node, msg);
 break;
}

case MSG_NETWORK_LeaveGroup:
{
 NODE_ADDR *mcastAddr = (NODE_ADDR*)
                                GLOMO_MsgReturnInfo(msg);
 RoutingOphmrLeaveGroup(node, *mcastAddr);
 GLOMO_MsgFree(node, msg);
 break;
}

case MSG_NETWORK_SendReply:
{
 NODE_ADDR *mcastAddr = (NODE_ADDR*)
                                GLOMO_MsgReturnInfo(msg);
 RoutingOphmrSendReply(node, *mcastAddr,&ophmr-
       >memberTable, &ophmr->tempTable);
 GLOMO_MsgFree(node, msg);
 break;
}

case MSG_NETWORK_CheckFg:
{
 NODE_ADDR *mcastAddr = (NODE_ADDR*)
                                GLOMO_MsgReturnInfo(msg);
 if (RoutingOphmrLookupFgFlag(*mcastAddr, &ophmr->fgFlag))
 {
 if (RoutingOphmrCheckFgExpired(*mcastAddr, &ophmr->fgFlag))
  {
    RoutingOphmrResetFgFlag(*mcastAddr, &ophmr->fgFlag);
  }
  else
   {
```

```
            RoutingOphmrSetTimer(node, MSG_NETWORK_CheckFg,
                    *mcastAddr, OPHMR_FG_TIMEOUT);
          }
         }
        GLOMO_MsgFree(node, msg);
        break;
       }

       default:
        printf("Time %s: Node %ld received message of unknown type
              %d.\n",clockStr, node->nodeAddr, msg->eventType);
       assert(FALSE);
   }
}

void RoutingOphmrHandleProtocolPacket(GlomoNode *node, Message
   *msg,   NODE_ADDR srcAddr, NODE_ADDR destAddr)
{
  OPHMR_PacketType *ophmrHeader = (OPHMR_PacketType *)
   GLOMO_MsgReturnPacket(msg);

   switch (*ophmrHeader)
    {
     case OPHMR_ACK:
       RoutingOphmrHandleAck(node, msg, srcAddr, destAddr);
       break;

     case OPHMR_JOIN_REPLY:
       RoutingOphmrHandleReply(node, msg, srcAddr, destAddr);
       break;

     default:
       printf("OPHMR received packet of unknown type\n");
       assert (FALSE);
    }
 }

 void RoutingOphmrRouterFunction(GlomoNode *node,Message
                                 *msg,NODE_ADDR destAddr,BOOL
                                 *packetWasRouted)
```

```
{
 GlomoNetworkIp* ipLayer = (GlomoNetworkIp *)node->
                                networkData.networkVar;
 GlomoRoutingOphmr* ophmr =(GlomoRoutingOphmr *)ipLayer-
                                >routingProtocol;
 IpHeaderType *ipHeader = (IpHeaderType *) msg->packet;
 OphmrIpOptionType option;

 if (ipHeader->ip_p == IPPROTO_OPHMR)
 {
    if (ipHeader->ip_dst >= IP_MIN_MULTICAST_ADDRESS)
    {
       NODE_ADDR sourceAddress;
       NODE_ADDR destinationAddress;
       unsigned char IpProtocol;
       unsigned int ttl;
       NetworkQueueingPriorityType priority;

       NetworkIpRemoveIpHeader(node, msg, &sourceAddress,
         &destinationAddress, &priority, &IpProtocol, &ttl);
       RoutingOphmrHandleProtocolPacket(                    node, msg,
                sourceAddress,destinationAddress);
       *packetWasRouted = TRUE;
       return;
    }
    else
    {
       *packetWasRouted = FALSE;
       return;
    }
 }

 *packetWasRouted = TRUE;
  if (FindAnIpOptionField(ipHeader, IPOPT_OPHMR) == NULL &&
    ipHeader->ip_src == node->nodeAddr)
 {

  #ifdef DEBUG
  printf("Node %ld has data to send\n", node->nodeAddr);
  #endif
```

```
    if (RoutingOphmrLookupMembership(destAddr, &ophmr-
>memberFlag))
     {
      if (RoutingOphmrLookupSentTable(destAddr, &ophmr->sentTable)
&&
          !RoutingOphmrCheckSendQuery(destAddr, &ophmr->sentTable))
         {
          #ifdef DEBUG
          printf(" Sending Data\n ");
          #endif
          RoutingOphmrSendData(node, msg, destAddr);
         }
       else
          {
           #ifdef DEBUG
           printf("    Sending Join Query\n");
           #endif
           RoutingOphmrSendQuery(node, msg, destAddr);
          }
       }
     }
     else
      {
       RoutingOphmrHandleData(node, msg);
      }
    }

  void RoutingOphmrFinalize(GlomoNode *node)
  {
    GlomoNetworkIp *ipLayer = (GlomoNetworkIp *)node-
                                  >networkData.networkVar;
    GlomoRoutingOphmr *ophmr = (GlomoRoutingOphmr *)ipLayer-
                                  >routingProtocol;
    FILE *statOut;
    char buf[GLOMO_MAX_STRING_LENGTH];

    sprintf(buf, "Number of Join Queries Txed = %d",
            ophmr->stats.numQueryTxed);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);
    sprintf(buf, "Number of Join Replies Txed = %d",
```

```
            ophmr->stats.numReplySent);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);
    sprintf(buf, "Number of Acks Txed = %d",
            ophmr->stats.numAckSent);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);

    sprintf(buf, "Number of CTRL Packets Txed = %d",
            ophmr->stats.numQueryTxed + ophmr->stats.numReplySent
            ophmr->stats.numAckSent);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);
    sprintf(buf, "Number of Data Txed = %d",
            ophmr->stats.numDataTxed);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);
    sprintf(buf, "Number of Data Packets Originated = %d",
            ophmr->stats.numDataSent);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);

    sprintf(buf, "Number of Data Packets Supposed to be Received = %d",
            ophmr->stats.numDataToReceive);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);

    sprintf(buf, "Number of Data Packets Received = %d",
            ophmr->stats.numDataReceived);
    GLOMO_PrintStat(node, "RoutingOphmr", buf);

  }

  void RoutingOphmrHandleData(GlomoNode *node, Message *msg)
  {
    GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                                  >networkData.networkVar;
    GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                                  >routingProtocol;
    IpHeaderType *ipHdr = (IpHeaderType
*)GLOMO_MsgReturnPacket(msg);
    NODE_ADDR sourceAddress;
    NODE_ADDR destinationAddress;
    unsigned char IpProtocol;
    unsigned int ttl;
    NetworkQueueingPriorityType priority;
```

```
clocktype delay;
NODE_ADDR srcAddr = ipHdr->ip_src;
NODE_ADDR mcastAddr = ipHdr->ip_dst;
OphmrIpOptionType option = GetOphmrIpOptionField(msg);
Message *newMsg = NULL;

if (option.query == TRUE)
{
   RoutingOphmrHandleJoinQuery(node, msg);
   return;
}

if (!RoutingOphmrLookupMessageCache(srcAddr, option.seqNumber,
        &ophmr->messageCache))
{
  RoutingOphmrInsertMessageCache(node, srcAddr, option.seqNumber,
        &ophmr->messageCache);

   if (RoutingOphmrLookupMembership(mcastAddr, &ophmr-
>memberFlag))
    {
      #ifdef DEBUG
      printf("Node %ld received DATA\n", node->nodeAddr);
      #endif
      #ifdef DEBUG
      printf("    Member got it!\n");
      #endif
      ophmr->stats.numDataReceived++;
      newMsg = GLOMO_MsgCopy(node, msg);
      NetworkIpRemoveIpHeader(node, newMsg, &sourceAddress,
        &destinationAddress, &priority, &IpProtocol, &ttl);
      SendToUdp(node, newMsg, priority, sourceAddress,
            destinationAddress);
    }
    if (RoutingOphmrLookupFgFlag(mcastAddr, &ophmr->fgFlag))
     {
      #ifdef DEBUG
      printf("Node %ld received DATA\n", node->nodeAddr);
      #endif
      #ifdef DEBUG
```

```
    printf("    FG. Forwarding it\n");
    #endif
    option.lastAddr = node->nodeAddr;
    option.hopCount++;
    SetOphmrIpOptionField(msg, &option);
    delay = pc_erand(node->seed) * OPHMR_BROADCAST_JITTER;
    NetworkIpSendPacketToMacLayerWithDelay(
          node, msg, DEFAULT_INTERFACE, ANY_DEST, delay);

      ophmr->stats.numDataTxed++;
   }
  }

  else
   {
    GLOMO_MsgFree(node, msg);
   }
 }
 void RoutingOphmrHandleJoinQuery(GlomoNode *node, Message *msg)
 {
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                                >networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                                >routingProtocol;
  IpHeaderType *ipHdr = (IpHeaderType
*)GLOMO_MsgReturnPacket(msg);
  NODE_ADDR sourceAddress;
  NODE_ADDR destinationAddress;
  unsigned char IpProtocol;
  unsigned int ttl;
  NetworkQueueingPriorityType priority;
  clocktype delay, jrDelay;
  OPHMR_MT_Node *mcastEntry;
  OPHMR_RPT_Node *mEntry;
  NODE_ADDR srcAddr = ipHdr->ip_src;
  NODE_ADDR mcastAddr = ipHdr->ip_dst;
  OphmrIpOptionType option = GetOphmrIpOptionField(msg);
  Message *newMsg = NULL;
  if (!RoutingOphmrLookupMessageCache(
     srcAddr, option.seqNumber, &ophmr->messageCache))
```

```
{
 #ifdef DEBUG
printf("Node %ld received Join Query from %d\n", node->nodeAddr,
      option.lastAddr);
#endif
RoutingOphmrInsertMessageCache(
   node, srcAddr, option.seqNumber, &ophmr->messageCache);
{
   #ifdef DEBUG
   printf(" Member got it!\n");
   #endif
   ophmr->stats.numDataReceived++;
   if (option.hopCount < OPHMR_MAX_HOP)
     {
        option.lastAddr = node->nodeAddr;
        option.hopCount++;
        SetOphmrIpOptionField(msg, &option);
        delay = pc_erand(node->seed) *
          OPHMR_BROADCAST_JITTER;
        NetworkIpSendPacketToMacLayerWithDelay(
             node, msg, DEFAULT_INTERFACE, ANY_DEST, delay);

        ophmr->stats.numDataTxed++;
        ophmr->stats.numQueryTxed++;

        #ifdef DEBUG
        printf("    Relaying it\n");
        #endif

     }
   newMsg = GLOMO_MsgCopy(node, msg);
   NetworkIpRemoveIpHeader(node, newMsg, &sourceAddress,
        &destinationAddress, &priority, &IpProtocol, &ttl);

   SendToUdp(node, newMsg, priority, sourceAddress,
              destinationAddress);

   mcastEntry = RoutingOphmrGetMTEntry(mcastAddr,&ophmr-
              >memberTable);
```

```
RoutingOphmrCheckSourceExpired(mcastAddr, &ophmr-
            >memberTable);

mEntry = RoutingOphmrGetRPTEntry(mcastAddr, &ophmr-
            >responseTable);

jrDelay = pc_erand(node->seed) * OPHMR_JR_JITTER;

 RoutingOphmrSetTimer(
        node, MSG_NETWORK_SendReply, mcastAddr, jrDelay);
}
else if (option.hopCount < OPHMR_MAX_HOP)
{
   option.lastAddr = node->nodeAddr;
   option.hopCount++;
   SetOphmrIpOptionField(msg, &option);

   delay = pc_erand(node->seed) * OPHMR_BROADCAST_JITTER;

   NetworkIpSendPacketToMacLayerWithDelay(
        node, msg, DEFAULT_INTERFACE, ANY_DEST, delay);

   ophmr->stats.numDataTxed++;
   ophmr->stats.numQueryTxed++;
   #ifdef DEBUG
   printf("    Relaying it\n");
   #endif

 }
 else
 {
    GLOMO_MsgFree(node, msg);
 }

else
 {
 GLOMO_MsgFree(node, msg);
 }
```

```
void RoutingOphmrHandleReply(GlomoNode *node, Message *msg,
    NODE_ADDR  lastAddr, NODE_ADDR mcastAddr)
{
 GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
        >networkData.networkVar;
 GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
        >routingProtocol;
 Message *newMsg;
 OPHMR_Ack *ackPkt;
 OPHMR_JoinReply *replyPkt = (OPHMR_JoinReply
        *)GLOMO_MsgReturnPacket(msg);
 OPHMR_TT_Node *mcastEntry;
 OPHMR_RPT_Node *mEntry;
 clocktype delay;
 int i;
 char *pktPtr;
 int pktSize = sizeof(OPHMR_Ack);
 BOOL changed = FALSE;

 for (i = 0; i < replyPkt->count; i++)
 {
 if (replyPkt->ackReq[i] && replyPkt->nextAddr[i] == node->nodeAddr)
   {
    newMsg = GLOMO_MsgAlloc(
        node, GLOMO_MAC_LAYER, 0, MSG_MAC_FromNetwork);
    GLOMO_MsgPacketAlloc(node, newMsg, pktSize);

    pktPtr = (char *) GLOMO_MsgReturnPacket(newMsg);
    ackPkt = (OPHMR_Ack *) pktPtr;
    ackPkt->pktType = OPHMR_ACK;
    ackPkt->mcastAddr = mcastAddr;
    ackPkt->srcAddr = replyPkt->srcAddr[i];
        NetworkIpSendRawGlomoMessageToMacLayer(node, newMsg,
        lastAddr, CONTROL, IPPROTO_OPHMR, 1,
        DEFAULT_INTERFACE, lastAddr);
    ophmr->stats.numAckSent++;
    #ifdef DEBUG
    printf("Node %ld received a Join Reply from node %ld\n", node-
        >nodeAddr, lastAddr);
    #endif
```

```
#ifdef DEBUG
printf("    sending Exp Ack to node %ld\n", lastAddr);
printf("    mcast = %u, src = %d\n", ackPkt->mcastAddr,        ackPkt-
    >srcAddr);
#endif
}
if (replyPkt->IAmFG)
{
  RoutingOphmrDeleteAckTable(mcastAddr,
       replyPkt->srcAddr[i], lastAddr, &ophmr->ackTable);
}
if (replyPkt->nextAddr[i] == node->nodeAddr &&
  replyPkt->srcAddr[i] != node->nodeAddr)
{
  #ifdef DEBUG
  printf("Node %ld received a Join Reply from node %ld\n", node-
       >nodeAddr, lastAddr);
  #endif
  #ifdef DEBUG
  printf("    I'm a FG!\n");
  #endif
  if (RoutingOphmrLookupFgFlag(mcastAddr, &ophmr->fgFlag))
  {
    RoutingOphmrUpdateFgFlag(mcastAddr, &ophmr->fgFlag);
  }
  else
  {
    RoutingOphmrSetFgFlag(mcastAddr, &ophmr->fgFlag);
  }

  mcastEntry = RoutingOphmrGetTTEntry(
                  mcastAddr, &ophmr->tempTable);
  if (mcastEntry == NULL)
  {
    #ifdef DEBUG
    #endif
    RoutingOphmrInsertTempTable(
      mcastAddr, replyPkt->srcAddr[i], &ophmr->tempTable);
  }
  else
```

```
    {
      #ifdef DEBUG
      #endif
      RoutingOphmrInsertTempSource(replyPkt->srcAddr[i],
              mcastEntry);
    }

    if (RoutingOphmrCheckTempChanged(
                    mcastAddr, &ophmr->tempTable))
    {
      #ifdef DEBUG
      printf("     Temp changed!\n");
      #endif
      mEntry = RoutingOphmrGetRPTEntry(
                    mcastAddr, &ophmr->responseTable);
      changed = TRUE;
     }

     RoutingOphmrCheckTempExpired(
                    mcastAddr, &ophmr->tempTable);
   }
}

if (changed && !RoutingOphmrCheckTempSent(
              mcastAddr, &ophmr->tempTable))
 {
  #ifdef DEBUG
  printf("  Changed and temp sent!\n");
  #endif
  RoutingOphmrSetTempSent(mcastAddr, &ophmr->tempTable);
        delay = pc_erand(node->seed) * OPHMR_JR_JITTER +
       OPHMR_JR_PAUSE_TIME;
   RoutingOphmrSetTimer(
    node, MSG_NETWORK_SendReply, mcastAddr, delay);
}

if (RoutingOphmrLookupFgFlag(mcastAddr, &ophmr->fgFlag))
{
   RoutingOphmrSetTimer(
```

```
        node, MSG_NETWORK_CheckFg, mcastAddr,
OPHMR_FG_TIMEOUT);
  }

  GLOMO_MsgFree(node, msg);

}

void RoutingOphmrHandleAck(GlomoNode *node, Message *msg,
 NODE_ADDR   lastAddr, NODE_ADDR targetAddr)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                            >networkData.networkVar;
                              GlomoRoutingOphmr* ophmr =
                            (GlomoRoutingOphmr *) ipLayer-
                            >routingProtocol;
  OPHMR_Ack *ackPkt = (OPHMR_Ack
*)GLOMO_MsgReturnPacket(msg);

  if (targetAddr == node->nodeAddr)
  {
    #ifdef DEBUG
    printf("Node %ld got Ack from node %ld\n", node->nodeAddr,
            lastAddr);
    #endif
    RoutingOphmrDeleteAckTable(ackPkt->mcastAddr, ackPkt->srcAddr,
            lastAddr, &ophmr->ackTable);
  }

  GLOMO_MsgFree(node, msg);
 }

 void RoutingOphmrInitMessageCache(OPHMR_MC *messageCache)
 {
   messageCache->front = NULL;
   messageCache->rear = NULL;
   messageCache->size = 0;
 }
 void RoutingOphmrInitSent(OPHMR_SS *sentTable)
 {
```

```
  sentTable->head = NULL;
  sentTable->size = 0;
}
void RoutingOphmrInitStats(GlomoNode *node)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                            >networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                            >routingProtocol;
  ophmr->stats.numQueryTxed = 0;

  ophmr->stats.numReplySent = 0;

  ophmr->stats.numAckSent = 0;
  ophmr->stats.numDataSent = 0;
  ophmr->stats.numDataReceived = 0;
  ophmr->stats.numDataToReceive = 0;
  ophmr->stats.numDataTxed = 0;

}

void RoutingOphmrDeleteMsgCache(OPHMR_MC *messageCache)
{
  OPHMR_MC_Node *toFree;
  toFree = messageCache->front;
  messageCache->front = toFree->next;
  pc_free(toFree);
  --(messageCache->size);

  if (messageCache->size == 0)
  {
     messageCache->rear = NULL;
  }
}

BOOL RoutingOphmrLookupFgFlag(NODE_ADDR mcastAddr,
OPHMR_FgFlag  *fgFlag)
{
  OPHMR_FF_Node *current;
  if (fgFlag->size == 0)
```

```
{
   return (FALSE);
}

for (current = fgFlag->head;
    current != NULL;current = current->next)
{
  if (current->mcastAddr == mcastAddr)
   {
      return (TRUE);
   }
}
return (FALSE);

}

BOOL RoutingOphmrLookupSentTable(NODE_ADDR mcastAddr,
        OPHMR_SS   *sentTable)
{
 OPHMR_SS_Node *current;

 if (sentTable->size == 0)
 {
    return (FALSE);
 }

 for (current = sentTable->head;
     current != NULL;current = current->next)
 {
    if (current->mcastAddr == mcastAddr)
    {
       return (TRUE);
    }
 }

 return (FALSE);
}

BOOL RoutingOphmrCheckFgExpired(NODE_ADDR mcastAddr,
                    OPHMR_FgFlag *fgFlag)
```

```
OPHMR_FF_Node *current;

if (fgFlag->size == 0 || fgFlag->head == NULL)
{
  return (FALSE);
}

for (current = fgFlag->head;
   current != NULL;current = current->next)
{
  if (current->mcastAddr == mcastAddr &&
     simclock() - current->timestamp >= OPHMR_FG_TIMEOUT)
  {
     return (TRUE);
  }
}

return (FALSE);

void RoutingOphmrInsertAckTable(NODE_ADDR mcastAddr,
 OPHMR_AT *ackTable, OPHMR_JoinReply *reply)
{
   ++(ackTable->size);
    ackTable->head = RoutingOphmrInsertATInOrder(
        mcastAddr, ackTable->head, reply);

}

BOOL RoutingOphmrCheckASExist(NODE_ADDR srcAddr,
OPHMR_AT_Node *mcast)
{
 OPHMR_AT_Snode *current;
 if (mcast->size == 0 || mcast->head == NULL)
 {
   return (FALSE);
 }
 for (current = mcast->head;
    current != NULL && current->srcAddr <= srcAddr;
    current = current->next)
```

```
  {
    if (current->srcAddr == srcAddr)
    {
      return (TRUE);
    }
  }
  return (FALSE);
}

OPHMR_AT_Snode *RoutingOphmrInsertAckSource(
      NODE_ADDR srcAddr, NODE_ADDR nextAddr,
OPHMR_AT_Node *mcast)
{
  OPHMR_AT_Snode *current;
  if (!RoutingOphmrCheckASExist(srcAddr, mcast))
  {
    ++(mcast->size);
    mcast->lastSent = simclock();
    mcast->head = RoutingOphmrInsertASInOrder(
                srcAddr, nextAddr, mcast->head, NULL);
    return (mcast->head);
  }
  else
  {
    for (current = mcast->head;
        current != NULL;
        current = current->next)
    {
      if (current->srcAddr == srcAddr)
      {
        current->nextAddr = nextAddr;
        current->numTx++;
        break;
      }
    }
    return (NULL);
  }

}
```

```
int RoutingOphmrGetAckCount(NODE_ADDR mcastAddr, OPHMR_AT
*ackTable)
{
  OPHMR_AT_Node *current;

  if (ackTable->size == 0 || ackTable->head == NULL)
  {
    return (0);
  }

   for (current = ackTable->head;
     current != NULL && current->mcastAddr <= mcastAddr;
     current = current->next)
  {
    if (current->mcastAddr == mcastAddr)
    {
      return (current->size);
    }
  }

  return (0);
}

void RoutingOphmrJoinGroup(GlomoNode *node, NODE_ADDR
mcastAddr)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *)
                            node>networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                            >routingProtocol;
  if (!RoutingOphmrLookupMembership(mcastAddr, &ophmr-
>memberFlag))
  {
    #ifdef DEBUG
    printf("Node %ld joined group %u\n", node->nodeAddr, mcastAddr);
    #endif
    RoutingOphmrSetMemberFlag(mcastAddr, &ophmr->memberFlag);

  }
}
```

```
void RoutingOphmrLeaveGroup(GlomoNode *node, NODE_ADDR
mcastAddr)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                            >networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                            >routingProtocol;
  if (RoutingOphmrLookupMembership(mcastAddr, &ophmr-
>memberFlag))
  {
     #ifdef DEBUG
     printf("Node %ld left group %u\n", node->nodeAddr, mcastAddr);
     #endif
     RoutingOphmrResetMemberFlag(mcastAddr, &ophmr->memberFlag);
     RoutingOphmrDeleteSourceSent(mcastAddr, &ophmr->sentTable);
     RoutingOphmrDeleteMemberTable(mcastAddr, &ophmr-
>memberTable);
  }

}

BOOL RoutingOphmrCheckAckTable(NODE_ADDR mcastAddr,
OPHMR_AT *ackTable)
{
  OPHMR_AT_Node *current;
  OPHMR_AT_Snode *curSrc;
  if (ackTable->size == 0 || ackTable->head == NULL)
  {
     return (FALSE);
  }
  for (current = ackTable->head; current != NULL; current = current->next)
  {
     if (current->mcastAddr == mcastAddr)
     {
        if (current->size == 0 || current->head == NULL)
        {
           return (FALSE);
        }

        for (curSrc = current->head;
```

```
            curSrc != NULL;
            curSrc = curSrc->next)
        {
          if (curSrc->numTx < OPHMR_MAX_NUM_TX)
          {
            return (TRUE);
          }
        }
      }
    }
    return (FALSE);
}

void RoutingOphmrDeleteAckTable(NODE_ADDR mcastAddr,
NODE_ADDR srcAddr,
                    NODE_ADDR lastAddr, OPHMR_AT *ackTable)
{
    OPHMR_AT_Node *current;
    OPHMR_AT_Snode *curSrc;
    OPHMR_AT_Snode *toFree;
    for (current = ackTable->head; current != NULL; current = current-
                    >next)
     {
      if (current->mcastAddr == mcastAddr)
      {
        if (current->size == 0 || current->head == NULL)
        {
          return;
        }
              for (curSrc = current->head;
            curSrc != NULL;
            curSrc = curSrc->next)
        {
          if (curSrc->srcAddr == srcAddr &&
            curSrc->nextAddr == lastAddr)
          {
            toFree = curSrc;

            if (curSrc->prev == NULL && curSrc->next == NULL)
            {
```

```
      current->head = curSrc->next;
    }
    else if (curSrc->prev == NULL)
    {
      curSrc->next->prev = curSrc->prev;
      current->head = curSrc->next;
    }
    else if (curSrc->next == NULL)
    {
      curSrc->prev->next = curSrc->next;
    }
    else
    {
      curSrc->prev->next = curSrc->next;
      curSrc->next->prev = curSrc->prev;
    }

    pc_free(toFree);
    --(current->size);
  }

  else if (curSrc->numTx >= OPHMR_MAX_NUM_TX)
  {
    toFree = curSrc;

    if (curSrc->prev == NULL && curSrc->next == NULL)
    {
      current->head = curSrc->next;
    }
    else if (curSrc->prev == NULL)
    {
      curSrc->next->prev = curSrc->prev;
      current->head = curSrc->next;
    }
    else if (curSrc->next == NULL)
    {
      curSrc->prev->next = curSrc->next;
    }
    else
    {
```

```
                curSrc->prev->next = curSrc->next;
                curSrc->next->prev = curSrc->prev;
            }

            pc_free(toFree);
            --(current->size);
          }
        }
      }
    }
}


void RoutingOphmrInitAckTable(OPHMR_AT *ackTable)
{
  ackTable->head = NULL;
  ackTable->size = 0;
}

 if (current->next != NULL && current->next->mcastAddr == mcastAddr)
     {
         toFree = current->next;
         current->next = toFree->next;
         pc_free(toFree);
         --(tempTable->size);
       }
    }
 }

 void RoutingOphmrSetTimer(GlomoNode *node, long eventType,
             NODE_ADDR  mcastAddr, clocktype delay)
 {
    Message *msg;
    NODE_ADDR *info;
    msg = GLOMO_MsgAlloc(node, GLOMO_NETWORK_LAYER,
              ROUTING_PROTOCOL_OPHMR, eventType);
    GLOMO_MsgInfoAlloc(node, msg, sizeof(NODE_ADDR));
    info = (NODE_ADDR *) GLOMO_MsgReturnInfo(msg);
    *info = mcastAddr;
    GLOMO_MsgSend(node, msg, delay);
```

```
}

void RoutingOphmrSendReply(GlomoNode *node, NODE_ADDR
mcastAddr,
                    OPHMR_MT *memberTable, OPHMR_TT *tempTable)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                                >networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                                >routingProtocol;
   Message *msg;
   OPHMR_JoinReply *replyPkt;
   OPHMR_AT_Node *mcastEntry;
   OPHMR_MT_Node *current;
   OPHMR_MT_Snode *curSrc;
   OPHMR_TT_Node *curNode;
   OPHMR_TT_Snode *curSrcNode;
   BOOL match;
   BOOL done;
   int i, j, k, l, m, n;
   char *pktPtr;
   int pktSize = sizeof(OPHMR_JoinReply);
   clocktype delay;
   BOOL needAck = FALSE;
  GLOMO_MsgPacketAlloc(node, msg, pktSize);
   pktPtr = (char *) GLOMO_MsgReturnPacket(msg);
   replyPkt = (OPHMR_JoinReply *) pktPtr;
   replyPkt->pktType = OPHMR_JOIN_REPLY;
   RoutingOphmrSetTempSent(mcastAddr, &ophmr->tempTable);
   if (RoutingOphmrLookupMemberTable(mcastAddr, &ophmr-
      >memberTable) &&RoutingOphmrCheckLastSent(mcastAddr,
      &ophmr->memberTable))
     {
         #ifdef DEBUG
         printf("      There are members!\n");
         #endif
         replyPkt->count = RoutingOphmrGetMemberCount(
              mcastAddr, &ophmr->memberTable);
         #ifdef DEBUG
         #endif
```

```
replyPkt->IAmFG = RoutingOphmrLookupFgFlag(
            mcastAddr, &ophmr->fgFlag);
for (current = memberTable->head;
    current != NULL && current->mcastAddr <= mcastAddr;
    current = current->next)
{
   if (current->mcastAddr == mcastAddr)
   {
      for (i = 0, curSrc = current->head;
          i < current->size && curSrc != NULL;
          i++, curSrc = curSrc->next)
      {
         replyPkt->srcAddr[i] = curSrc->srcAddr;
         #ifdef DEBUG
         #endif
      }
   }
}

for (j = 0; j < replyPkt->count; j++)
{
   replyPkt->nextAddr[j] =RoutingOphmrGetNextNode(
      replyPkt->srcAddr[j], &ophmr->routeTable);
   #ifdef DEBUG
   #endif
   if (replyPkt->nextAddr[j] == replyPkt->srcAddr[j])
   {
     replyPkt->ackReq[j] = TRUE
     needAck = TRUE;
   }
   else
   {
      replyPkt->ackReq[j] =  FALSE;
      needAck = TRUE;
   }
}

if (tempTable->size == 0 || tempTable->head == NULL)
{
}
```

```
else
{
   for (curNode = tempTable->head;
     curNode != NULL && curNode->mcastAddr <=
              mcastAddr;curNode = curNode->next)
   {
     if (curNode->mcastAddr == mcastAddr)
     {
       if (curNode->size == 0 || curNode->head == NULL)
        {
        }
        else
        {
          for (curSrcNode = curNode->head;
           curSrcNode != NULL;
               curSrcNode = curSrcNode->next)
           {
              done = FALSE;
              for (k = 0; k < replyPkt->count; k++)
              {
                 if (replyPkt->srcAddr[k] ==
                    curSrcNode->srcAddr)
                 {
                    done = TRUE;
                    break;
                 }
              }

             if (done == FALSE)
             {
                replyPkt->srcAddr[replyPkt->count] =
                   curSrcNode->srcAddr;
              }
             else
               {
                replyPkt->ackReq[replyPkt->count] =

                                          FALSE;
               }
             replyPkt->count++;
```

```
                }
            }
        }
    }
}

#ifdef DEBUG
#endif
while (l < replyPkt->count)
 {
   match = FALSE;
   match = RoutingOphmrCheckResponseMatch(
   mcastAddr, replyPkt->srcAddr[l], &ophmr->responseTable);
   if (!match)
     {
      for (m = l; m < replyPkt->count; m++)
        {
       replyPkt->srcAddr[m] = replyPkt->srcAddr[m+1];
       replyPkt->nextAddr[m] = replyPkt->nextAddr[m+1];
       replyPkt->ackReq[m] = replyPkt->ackReq[m+1];
      }

      replyPkt->count--;
    }
    if (match)
     {
      l++;
     }
   }

 #ifdef DEBUG
 #endif
 if (replyPkt->count > 0)
    {
     RoutingOphmrDeleteResponseTable(
       mcastAddr, &ophmr->responseTable);

    #ifdef DEBUG
    for (l = 0; l < replyPkt->count; l++)
```

```
    printf("src[%d] = %d, nxt[%d] = %d\n", l, replyPkt-
                                >srcAddr[l], l, replyPkt->nextAddr[l]);
    #endif
    NetworkIpSendRawGlomoMessageToMacLayer(
        node, msg, mcastAddr, CONTROL, IPPROTO_OPHMR, 1,
            DEFAULT_INTERFACE, ANY_DEST);
    ophmr->stats.numReplySent++;
    RoutingOphmrSetMemberSent(mcastAddr, &ophmr-
                        >memberTable);
    RoutingOphmrUpdateLastSent(mcastAddr, &ophmr-
                        >memberTable);
    if (needAck){
      mcastEntry = RoutingOphmrGetATEntry(
        mcastAddr, &ophmr->ackTable);
      if (mcastEntry == NULL)
      {
        RoutingOphmrInsertAckTable(
          mcastAddr, &ophmr->ackTable, replyPkt);
      }
      else
      {
        for (n = 0; n < replyPkt->count; n++)
        {
          RoutingOphmrInsertAckSource(replyPkt->srcAddr[n],
           replyPkt->nextAddr[n], mcastEntry);
        }
      }
      RoutingOphmrSetTimer(
        node, MSG_NETWORK_CheckAcked, mcastAddr,
        OPHMR_CHECKACK_INTERVAL);
    }
   }
}
else if (RoutingOphmrLookupMemberTable(mcastAddr, &ophmr-
                >memberTable))
{
#ifdef DEBUG
printf("Node %ld send Join Reply recently. Sending DELAYED\n",
            node->nodeAddr);
#endif
```

```
  RoutingOphmrSetTimer(node, MSG_NETWORK_SendReply,
                              mcastAddr,  OPHMR_JR_PAUSE_TIME);
  }

else if (RoutingOphmrLookupFgFlag(mcastAddr, &ophmr->fgFlag))
{
  #ifdef DEBUG
  printf("FG node %ld sending a Join Reply\n", node->nodeAddr);
  #endif
  replyPkt->count = RoutingOphmrGetTempCount(
    mcastAddr, &ophmr->tempTable);
  replyPkt->IAmFG =
                              RoutingOphmrLookupFgFlag(mcastAddr,&oph
                              mr->fgFlag);
  for (curNode = tempTable->head;
      curNode != NULL && curNode->mcastAddr <= mcastAddr;
      curNode = curNode->next)
  {
     if (curNode->mcastAddr == mcastAddr)
     {
        for (i = 0, curSrcNode = curNode->head;
            i < curNode->size && curSrcNode != NULL;
            i++, curSrcNode = curSrcNode->next)
        {
           replyPkt->srcAddr[i] = curSrcNode->srcAddr;
        }
     }
  }
  for (j = 0; j < replyPkt->count; j++)
  {
     replyPkt->nextAddr[j] = RoutingOphmrGetNextNode(
     replyPkt->srcAddr[j], &ophmr->routeTable);
     if (replyPkt->nextAddr[j] == replyPkt->srcAddr[j])
     {
        replyPkt->ackReq[j] = TRUE needAck = TRUE;
     }
     else
     {
      replyPkt->ackReq[j] =  FALSE;
      needAck = TRUE;
```

```
    }
  }
  k = 0;
  while (k < replyPkt->count)
  {
    match = FALSE;
    match = RoutingOphmrCheckResponseMatch(
    mcastAddr, replyPkt->srcAddr[k], &ophmr->responseTable);
    if (!match)
    {
     for (l = k; l < replyPkt->count; l++)
      {
        replyPkt->srcAddr[l] = replyPkt->srcAddr[l+1];
        replyPkt->nextAddr[l] = replyPkt->nextAddr[l+1];
         replyPkt->ackReq[l] = replyPkt->ackReq[l+1];
      }
      replyPkt->count--;
    }
    if (match)
    {
      k++;
    }
  }
  NetworkIpSendRawGlomoMessageToMacLayer(
    node, msg, mcastAddr, CONTROL, IPPROTO_OPHMR, 1,
     DEFAULT_INTERFACE, ANY_DEST);
     ophmr->stats.numReplySent++;
     if (needAck)
     {
       mcastEntry = RoutingOphmrGetATEntry(mcastAddr, &ophmr-
                                >ackTable);
      if (mcastEntry == NULL)
      {
        RoutingOphmrInsertAckTable(
               mcastAddr, &ophmr->ackTable, replyPkt);
      }
      else
      {
        for (m = 0; m < replyPkt->count; m++)
        {
```

```
            RoutingOphmrInsertAckSource(
            replyPkt->srcAddr[m], replyPkt->nextAddr[m], mcastEntry);
          }
        }

      RoutingOphmrSetTimer(
         node, MSG_NETWORK_CheckAcked, mcastAddr,
        OPHMR_CHECKACK_INTERVAL);
    }
  }
 }
}

void RoutingOphmrRetxReply(
 GlomoNode *node, NODE_ADDR mcastAddr, OPHMR_AT *ackTable)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                            >networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                            >routingProtocol;
  Message *msg;
  OPHMR_JoinReply *replyPkt;
  OPHMR_AT_Node *mcastEntry;
  OPHMR_AT_Node *current;
  OPHMR_AT_Snode *curSrc;
  int i, j;
  char *pktPtr;
  int pktSize = sizeof(OPHMR_JoinReply);
  clocktype delay;

  msg = GLOMO_MsgAlloc(node, GLOMO_MAC_LAYER, 0,
                       MSG_MAC_FromNetwork);
  GLOMO_MsgPacketAlloc(node, msg, pktSize);
  pktPtr = (char *) GLOMO_MsgReturnPacket(msg);
  replyPkt = (OPHMR_JoinReply *) pktPtr;
  replyPkt->pktType = OPHMR_JOIN_REPLY;
  replyPkt->count = RoutingOphmrGetAckCount(mcastAddr, &ophmr-
                    >ackTable);
  replyPkt->IAmFG = RoutingOphmrLookupFgFlag(mcastAddr, &ophmr-
                    >fgFlag);
```

```
for (current = ackTable->head;
    current != NULL && current->mcastAddr <= mcastAddr;
    current = current->next)
{
   if (current->mcastAddr == mcastAddr)
   {
      for (i = 0, curSrc = current->head;
          i < current->size && curSrc != NULL;
          i++, curSrc = curSrc->next)
      {
         replyPkt->srcAddr[i] = curSrc->srcAddr;
         replyPkt->nextAddr[i] = curSrc->nextAddr;
         if (curSrc->numTx > 0 || curSrc->srcAddr == curSrc->nextAddr)
         {
            replyPkt->ackReq[i] = TRUE;
         }
         else
         {
            replyPkt->ackReq[i] = FALSE;
         }
   #ifdef DEBUG
   printf("src[%d] = %d, next[%d] = %d, ack[%d] = %d\n", i,
      replyPkt->srcAddr[i], i, replyPkt->nextAddr[i], i, replyPkt-
      >ackReq[i]);
   #endif
   }
 }
}
if (replyPkt->count > 0)
 {
   delay = pc_erand(node->seed) * OPHMR_JR_RETX_JITTER;
   NetworkIpSendRawGlomoMessageToMacLayerWithDelay(
    node, msg, mcastAddr, CONTROL, IPPROTO_OPHMR, 1,
    DEFAULT_INTERFACE, ANY_DEST, delay);
    ophmr->stats.numReplySent++;
   #ifdef DEBUG
   printf("Node %ld retransmitting a Join Reply\n", node->nodeAddr);
   #endif
   RoutingOphmrSetTimer(
```

```
    node, MSG_NETWORK_CheckAcked, mcastAddr,
            OPHMR_CHECKACK_INTERVAL);
   mcastEntry = RoutingOphmrGetATEntry(mcastAddr, &ophmr-
            >ackTable);
    if (mcastEntry == NULL)
    {
     RoutingOphmrInsertAckTable(mcastAddr, &ophmr->ackTable,
          replyPkt);
    }
    else
    {
       for (j = 0; j < replyPkt->count; j++)
       {
          RoutingOphmrInsertAckSource(replyPkt->srcAddr[j], replyPkt-
               >nextAddr[j], mcastEntry);
       }
    }
  }

}
void RoutingOphmrSendData(GlomoNode *node, Message *msg,
NODE_ADDR mcastAddr)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                            >networkData.networkVar;
  GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                            >routingProtocol;
  OphmrIpOptionType option;
  GLOMO_MsgSetLayer(msg, GLOMO_MAC_LAYER, 0);
  GLOMO_MsgSetEvent(msg, MSG_MAC_FromNetwork);
  AddCustomOphmrIpOptionFields(node, msg);
  option.query = FALSE;
  option.lastAddr = node->nodeAddr;
  option.seqNumber = RoutingOphmrGetSeq(node);
  option.hopCount = 1;
  NetworkIpSendPacketToMacLayer(node, msg,DEFAULT_INTERFACE,
   ANY_DEST);
   void RoutingOphmrSendQuery(GlomoNode *node, Message *msg,
    NODE_ADDR mcastAddr){
```

```
GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node-
                          >networkData.networkVar;
GlomoRoutingOphmr* ophmr = (GlomoRoutingOphmr *) ipLayer-
                          >routingProtocol;
OphmrIpOptionType option;
GLOMO_MsgSetLayer(msg, GLOMO_MAC_LAYER, 0);
GLOMO_MsgSetEvent(msg, MSG_MAC_FromNetwork);
AddCustomOphmrIpOptionFields(node, msg);
option.query = TRUE;
option.lastAddr = node->nodeAddr;
option.seqNumber = RoutingOphmrGetSeq(node);
option.hopCount = 1;
SetOphmrIpOptionField(msg, &option);
NetworkIpSendPacketToMacLayer(node, msg,
                              DEFAULT_INTERFACE,
                              ANY_DEST);

ophmr->stats.numDataTxed++;
ophmr->stats.numQueryTxed++;
RoutingOphmrInsertMessageCache(node, node->nodeAddr,
        option.seqNumber, &ophmr->messageCache);
if (RoutingOphmrCheckSendQuery(mcastAddr, &ophmr->sentTable))
 {
  RoutingOphmrUnsetSendQuery(mcastAddr, &ophmr->sentTable);
 }
if (!RoutingOphmrLookupSentTable(mcastAddr, &ophmr->sentTable))
 {
   RoutingOphmrSetSent(mcastAddr, &ophmr->sentTable);
 }
 else
 {
   RoutingOphmrSetMinExpireTime(mcastAddr, &ophmr->sentTable);
 }

RoutingOphmrSetTimer(node, MSG_NETWORK_CheckTimeoutAlarm,
                     mcastAddr, OPHMR_TIMER_INTERVAL);
}
```

8.REFERENCES

[1] B. An and S. Papavassiliou, "A Mobility-Based Hybrid Multicast Routing in Mobile Ad-Hoc Wireless Networks," Proc. IEEE Military Comm. Conf. (MILCOM '01), Oct. 2001.

[2] P. Bergamo, A. Giovanardi, A. Travasoni, D. Maniezzo, G. Mazzini, and M. Zorzi, "Distributed Power Control for Energy Efficient Routing in Ad Hoc Networks," Wireless Networks, vol. 10, no. 1, pp. 29-42, 2004.

[3] J. Broch, D.B. Johnson, and D.A. Maltz, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks," Internet Draft, draft-ietf-manet-dsr-03.txt, Oct. 1999.

[4] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," Internet Draft, draft-ietf-manet-olsr-11.txt, Jan. 2003.

[5] V. Devarapalli and D. Sidhu, "MZR: A Multicast Protocol for Mobile Ad Hoc Networks," Proc. IEEE Int'l Conf. Comm., June 2001.

[6]http://www.terminodes.org/