ENSC 835: HIGH-PERFORMANCE NETWORK
CMPT 885: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

# More Efficient Routing Algorithm for Ad Hoc Routing Algorithm

Fall 2003

## Final Report

Mark Wang mrw@sfu.ca
Carl Qian chunq@sfu.ca
http://www.ensc.sfu.ca/people/grad/chunq/personal/

## Table of Contents

**ABSTRACT……………………………………………………………………………………..3**
**I. Introduction……………………………………………………………………………….4**
**II. Overview of AODV ……………………………………………………………………….6**
**III. Efficient Flooding in Ad Hoc Networks and related works………………………………7**
**IV. Multipoint Relays Technique……………………………………………………………..8**
IV.1 Terminology…………………………………………………………………………………9
Node………………………………………………………………………………………………9
Neighbor node…………………………………………………………………………………....9
2-hop neighbor……………………………………………………………………………….....9
Multipoint relay (MPR)………………………………………………………………………….9
Multipoint relay selector (MPR selector, MS)…………………………………………………...9
Symmetric link……………………………………………………………………………………9
Asymmetric link………………………………………………………………………………….9
IV.2 Multipoint Relay in Detail……………………………………………………………………...9
IV.3 Multipoint Relay Enhancement…………………………………………………………….10
IV.4 Implementation of MPR…………………………………………………………………......10
IV.4.1 HELLO message broadcast………………………………………………………………10
IV.4.2 HELLO message processing………………………………………………………………...11
IV.4.3 Multipoint Relay Selection Algorithm……………………………………………………12
**V. Simulation results comparison and analysis……………………………………………..14**
**VI. Discussion and Future work……………………………………………………………..16**
**VII. Conclusion………………………………………………………………………….....16**
**VIII. References……………………………………………………………………….. .17**
**APPENDIX I: MPR Selector Pseudo Code…………………………………………………....18**
**APPENDIX II: Code Listing……………………………………………………………….....20**
aodv_queue.cc ..……………………….……………………………………………………….20
aodv_queue.h ………………………………….…………………………………………….....25
OTCL CODES ………………………………….………………………………………….......26

# ABSTRACT

Network wide broadcasting [3] in Mobile Ad Hoc Networks provides important control and route establishment functionality for a number of unicast and multicast protocols. Nodes in an ad hoc network must cooperate and carry out a distributed routing protocol in order to make multi-hop communications possible. On Demand Routing is one of the most popular routing styles in ad hoc networks. In On Demand Routing, "flooding" is used to find a feasible route from source to destination. The blind flooding can become very inefficient because of redundant, "superfluous" forwarding. In fact, superfluous flooding increases link overhead and wireless medium congestion. In this project, we implement and modify the mechanism of multipoint relays (MPR) [4] to efficiently flood the broadcast message [3] in the mobile wireless networks. We demonstrate the efficiency of the proposed scheme in the AODV (Ad Hoc on Demand Distance Vector) [5] routing scheme.

Mark Wang
Carl Qian

# I. Introduction

Mobile ad hoc networks are emerging as a very hot issue in telecommunication. It is a collection of wireless nodes dynamically forming a temporary network without the use of any existing network infrastructure or centralized administration, characterized by node mobility, dynamic topology structure, scarce bandwidth, unreliable media and limited power supply. They have numerous applications in several fields, including both military and civilian use. Since the effective range of high capacity radio link is limited (IEEE 802.11a, g), the capacity of packet relaying inside the network is a key component of such network. Mobile ad hoc routing has received tremendous quantity of intention since the opening of the IETF working group MANET. There have been many recent proposals of unicast routing protocols for ad hoc mobile networks. We have done a complete research on the current routing protocols and summarized the existing ad hoc routing protocols as shown in Figure 1. The common classifications for these protocols are in the following three main categories:

1. Proactive protocols (routing table-driven): based on periodic exchanges that proactively update the routing tables to all possible destination, even if no traffic goes through. OLSR, TBRPF, and DSDV are proactive protocols.

2. Reactive protocols (route on-demand): based on on-demand route discoveries that update routing table only for the destination that have traffic going through. The two reactive protocols are AODV and DSR.

3. Hierarchical and Hybrids Protocol: based on selected cluster head and gateway to find routes from source to destination.

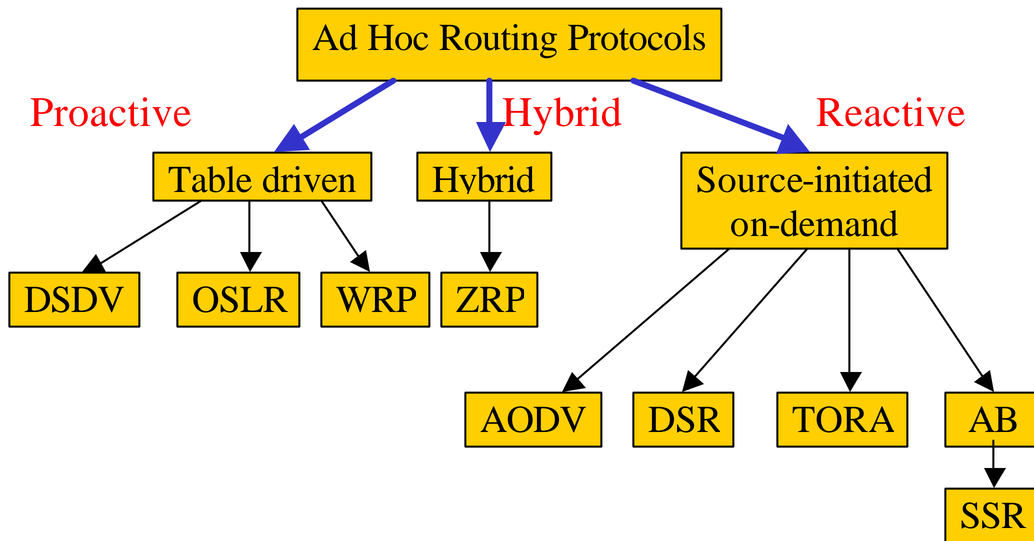


Figure 1: Existing Ad Hoc Routing Protocols

On Demand Routing [5] is one of the most popular routing styles in ad hoc networks. It discovers routes based on needs, thus reduces the broadcast overhead. Flooding is the basic mechanism to propagate control messages in finding the route from source to destination. In flooding, a node transmits a message to all of its neighbors. The neighbors in turn transmit to their neighbors and so on until the message has been propagated to the entire network**.** Conventional flooding can be very costly in On Demand networks in terms of network throughput efficiency as well as node energy consumption. The main reason is that the same packet is rebroadcast unnecessarily several times (redundant rebroadcast). Indeed, the penalty of redundant rebroadcast increases when the size of network grows and the density of network increases. AODV protocol uses two control mechanisms to reduce the cost of flooding. One of the mechanisms for prohibiting infinite duplication is tracking flooding packets. Duplicates are detected (from a unique source identifier and a sequence number, for example) by each receiving node and are immediately discarded in order to avoid endless looping. Another control mechanism is Time-to-Live (TTL). A flooding packet carries a TTL field which represents the maximum hop that the packet can traverse. Upon reception of a flooding packet, the receiving node checks the TTL field and determines whether the packet will be re-broadcasted (after decreasing TTL) or dropped.

In this project we implement and optimize an alternative flooding control mechanism, called *Multipoint Relay* (MPR), which was first introduced in the Optimized Link State Routing Protocol (OLSR) [4], a proactive routing protocol. In order to use this optimized mechanism, the nodes must perform a proactive control in order to know their two-hop neighborhood. This can be done via the reception of hello messages generating by the nodes and containing their neighbors list. Since two nodes are neighbors when they can see each other address in their respective hellos, this is a very straightforward procedure. The cost of the proactive neighbors control is not negligible but is far less important than the cost of classical flooding, in particular when the condition of traffic and mobility causes too many route discovery procedures. In this project, we will do some comparisons between AODV and AODV with modified MPR under different situations.

## II. Overview of AODV

The Ad hoc On Demand Distance Vector (AODV) [5] routing algorithm is a routing protocol designed for ad hoc mobile networks. AODV is capable of both unicast and multicast routing. It is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Additionally, AODV forms trees which connect multicast group members. The trees are composed of the group members and the nodes needed to connect the members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes.

AODV builds routes using a route request / route reply query cycle. When a source node desires a route to a destination for which it does not already have a route, it broadcasts a route request (RREQ) packet across the network. Nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the route tables. In addition to the source node's IP address, current sequence number, and broadcast ID, the RREQ also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the RREQ may send a route reply (RREP) if it is either the destination or if it has a route to the destination with corresponding sequence number greater than or equal to that contained in the RREQ. If this is the case, it unicasts a RREP back to the source. Otherwise, it rebroadcasts the RREQ. Nodes keep track of the RREQ's source IP address and broadcast ID. If they receive a RREQ which they have already processed, they discard the RREQ and do not forward it.

As the RREP propagates back to the source, nodes set up forward pointers to the destination. Once the source node receives the RREP, it may begin to forward data packets to the destination. If the source later receives a RREP containing a greater sequence number or contains the same sequence number with a smaller hop count, it may update its routing information for that destination and begin using the better route.

As long as the route remains active, it will continue to be maintained. A route is considered active as long as there are data packets periodically traveling from the source to the destination along that path. Once the source stops sending data packets, the links will time out and eventually be deleted from the intermediate node routing tables. If a link break occurs while the route is active, the node upstream of the break propagates a route error (RERR) message to the source node to inform it of the now unreachable destination(s). After receiving the RERR, if the source node still desires the route, it can reinitiate route discovery.

## III. Efficient Flooding in Ad Hoc Networks and Related Works

Flooding is a packet dissemination procedure by which every incoming packet at a node is sent out on every outgoing link except the one it arrived on. It is used to find a feasible route to a destination or to advertise routing information. If the network is dense, it is not necessary for every node to relay the flood search packet. In fact, it may suffice to use only a subset of nodes as relays. There are many ways to reduce the number of forwarding participants. All of the approaches concern selecting the dominant set, i.e., a minimal subset of forwarding nodes which is sufficient to deliver the flooding packet to every other node in the system. One is called Probability Based Methods [1] [2] in which nodes only rebroadcast with a predetermined probability. In dense networks multiple nodes share similar transmission coverage. Thus, randomly having some nodes not rebroadcast saves node and network resources without harming delivery effectiveness. The other one is called Location-Based Scheme which uses a more precise estimation of expected additional coverage area in the decision to rebroadcast. In this method, each node must have the means to determine its own location, e.g., a Global Positioning System (GPS). Our project is focused on the Neighbor Knowledge Methods witch employs multipoint technique to choose the dominated set to forward the routing packages. Figure2 and Figure 3 [4] show that applying MPR technique can reduce the flood significantly.

## IV. Multipoint Relays Technique (Our Modification)

In this section, we present the multipoint relay technique by introducing the related terminologies followed by detailed description of MPR technique.

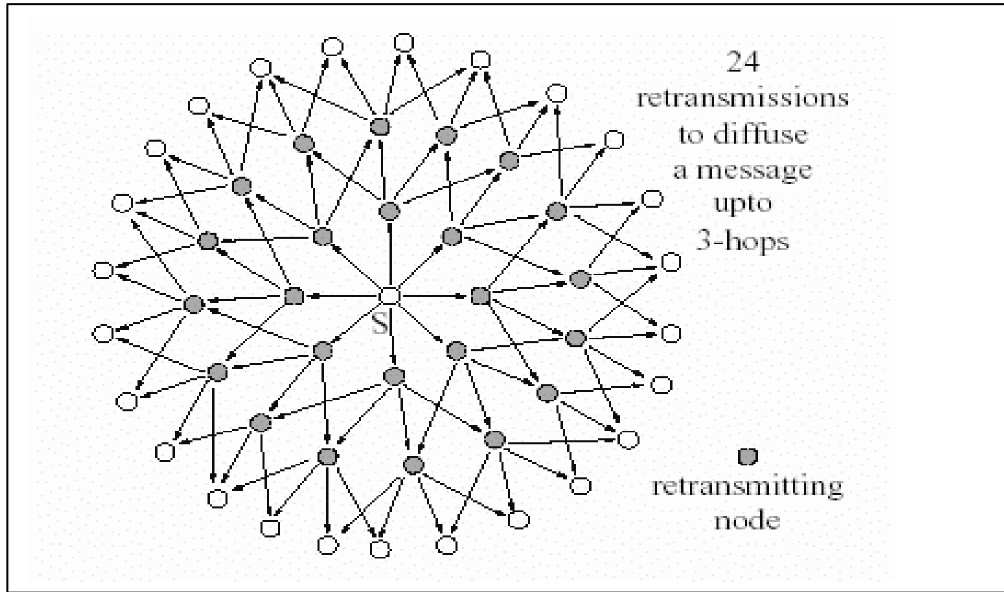


Figure 2: Diffusion of broadcast message using pure flooding

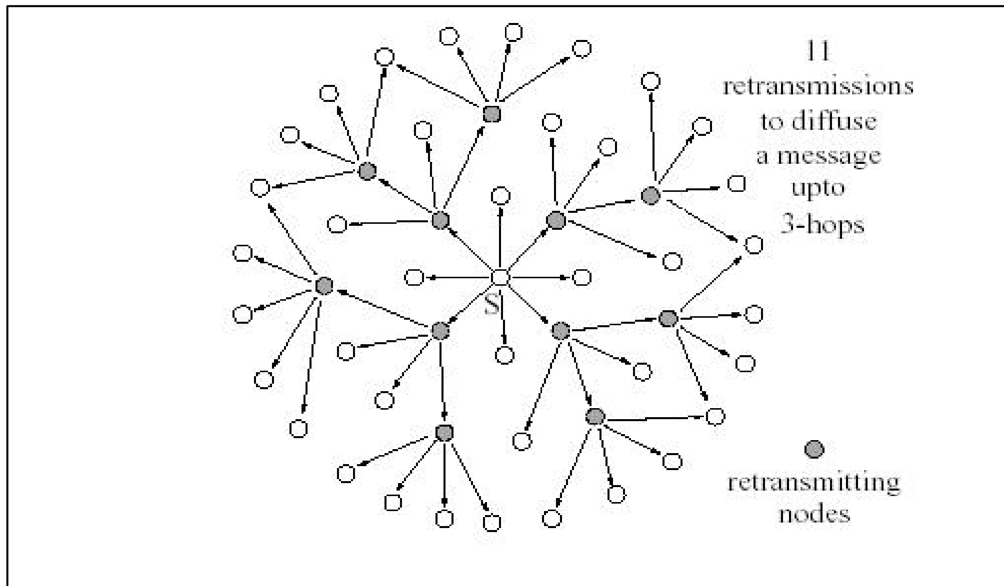


Figure 3: Diffusion of broadcast message using Multipoint Relays

## IV.1. Terminology

✍✍ ***Node***
A MANET router which implements AODV Routing protocol.

✍✍ ***Neighbor node***
A node X is a neighbor node of node Y if node Y can hear node X
.

✍✍ ***2-hop neighbor***
A node heard by a neighbor.

✍✍ ***Multipoint relay (MPR)***
A node which is selected by its 1-hop neighbor, node X, "re-transmit" all the broadcast messages that it receives from X, provided that the message is not a duplicate, and that the time to live field of the message is greater than one.

✍✍ ***Multipoint relay selector (MPR selector, MS)***
A node which has selected its 1-hop neighbor, node X, as its multipoint relay, will be called a multipoint relay selector of node X.

✍✍ ***Symmetric link***
A verified bi-directional link between two AODV interfaces.

✍✍ ***Asymmetric link***
A link between two AODV interfaces, verified in only one direction.

## IV.2. Multipoint Relay in Detail

The idea of multipoint relays is to minimize the overhead of flooding messages in the network by reducing duplicate retransmissions in the same region. Each node in the network selects a set of nodes in its neighborhood which may retransmit its messages. This set of selected neighbor nodes is called the "Multipoint Relay" (MPR) set of that node. The neighbors of node N which are *NOT* in its MPR set, receive and process broadcast messages but do not retransmit broadcast messages received from node N. Each node selects its MPR set among its one hop neighbors. This set is selected such that it covers (in terms of radio range) all nodes that are two hops away. The neighborhood of any node N can be defined as the set of nodes which have a symmetric link to N. The 2-hop neighborhood of N can be defined as the set of nodes which don't have a symmetric link to N but have a symmetric link to the neighborhood of N. The MPR set of N, denoted as MPR (N), is then an arbitrary subset of the neighborhood of N which satisfies the following condition: every node in the 2-hop neighborhood of N must have a symmetric link toward MPR (N). The smaller the MPR set is, the more optimal is the routing protocol. Each node maintains information about a set of its neighbors. This is the set of neighbors, called the "Multipoint Relay Selector set" (MPR selector set), which have selected the node as a MPR. A node obtains this information from the periodic HELLO messages received from the neighbors. A broadcast message, intended to be

diffused in the whole network, coming from these MPR selector neighbor nodes is assumed to be retransmitted by the node. This set can change over time (i.e. when a node selects another MPR-set) and is indicated by the selector nodes in their HELLO messages. Each node has a specific "Multipoint relay Selector Sequence Number" (MSSN) associated with this set. Whenever its MPR selector set is updated, the node also increments it's MSSN.

## IV.3. Multipoint Relay Enhancement

In our modified MRP, when a node receives a broadcast packet and is listed as a MRP, the node uses 2-hop neighbor knowledge to determine which neighbors also received the broadcast packet in the same transmission. These neighbors are considered already "covered" and are removed from the neighbor graph used to choose next hop MRPs. This modification will further reduce the flood overhead because it eliminated the duplication of choose MRPs.

## IV.4. Implementation of MRP

The implementation of MPR technique requires up to 2-hop neighbor information. This information is obtained by HELLO message broadcast and HELLO message processing. A node maintains information about its one hop neighbors, the status of the link with these neighbors, a list of 2-hop neighbors that these one hop neighbors give access to, and an associated holding time. Based on the one hop and 2-hop neighbor information, the MPR dominate set can be calculated.

### *IV.4.1. HELLO message broadcast*

Each node should detect the neighbor nodes with which it has a direct and symmetric link. The uncertainties over radio propagation may make some links asymmetric. Consequently, all links MUST be checked in both directions in order to be considered valid. To accomplish this, each node broadcasts HELLO messages, containing information about neighbors and their link status. The link status may be "symmetric", " asymmetric" or "MPR". "Symmetric" indicates that the link has been verified to be bi-directional, i.e. it is possible to transmit data in both directions. "Asymmetric" indicates that the node can hear HELLO messages from a neighbor, but it is not confirmed that this neighbor is also able to receive messages from the node. "MPR" indicates that a node is selected by the sender as a MPR. A status of MPR further implies that the link is symmetric. These HELLO messages are broadcast to all one-hop neighbors, but are *Not Relayed* to further nodes.

A HELLO-message contains:

- A list of addresses of neighbors, to which there exists a symmetric link;
- A list of addresses of neighbors, which have been " Asymmetric ";
- A list of neighbors, which have been selected as MPRs.

*IV.4.2. HELLO message processing*

Upon receiving a HELLO message, the node should update the neighbor information corresponding to the sender node address. Suppose the "Originator Address" will be used for the address of the node which sent the HELLO-message.

1. If there exists a neighbor tuple with N_addr = Originator Address:

   1.1 if for that tuple N_status == ASYM_LINK:

   1.1.1 if the node finds its own address among the addresses listed in the HELLO message (with Link Type ASYM_LINK, SYM_LINK or MPR_LINK), it updates the N_status of the tuple to SYM_LINK and sets N_time = current time + NEIGHB_HOLDING_TIME.

   1.1.2 otherwise, if the node does not find its own address among the addresses listed in the HELLO message, it sets N_time = current time + NEIGHB_HOLDING_TIME.

   1.2 otherwise, if for that tuple:

   N_status == SYM_LINK OR
   N_status == MPR_LINK

   then:

   1.2.1 if the node finds its own address among the addresses listed in the HELLO message (with Link Type ASYM_LINK, SYM_LINK or MPR_LINK), it sets N_time = current time NEIGHB_HOLDING_TIME.

2. Otherwise, a new neighbor tuple is created with:

   N_addr = Originator Address
   N_status with the value of SYM_LINK if the node finds its own address (with Link Type ASYM_LINK, SYM_LINK or MPR_LINK) among the addresses listed in the HELLO message, and to the value of ASYM_LINK otherwise
   N_time = current time + NEIGHB_HOLD_TIME

The 2-hop neighbor set is updated as follows: for each 2-hop neighbor address listed in the HELLO message with Link Type
SYM_LINK or MPR_LINK:

1. if a 2-hop tuple exists with:

   N_addr == Originator Address AND
   N_2hop_address == the address of the 2-hop neighbor

   then the N_time of that tuple is set to:

   N_time = current time + NEIGHB_HOLD_TIME

2. otherwise a new 2-hop tuple is created with:
   N_addr = Originator Address,
   N_2hop_address = the address of the 2-hop neighbor,
   N_time = current time + NEIGHB_HOLD_TIME.

Based on the information obtained from the HELLO messages, each node construct its MPR selector set. Thus, upon receiving a HELLO message, if a node finds its own address in the address list with a link type of "MPR", it MUST update the MPR selector set to contain updated information about the sender of the HELLO message:

1. If a MPR selector tuple exists with:
   MS_addr == Originator Address
   then the expiration time of that tuple is set to:
   MS_time = current time + NEIGHB_HOLD_TIME.

2. Otherwise, a new MPR selector tuple is created with:
   MS_addr = Originator Address
   MS_time = current time + NEIGHB_HOLD_TIME
   2.1 MSSN is incremented by one to indicate that the MPR selector table has been changed.

### *IV.4.3. Multipoint Relay Selection Algorithm*

Each node in the network selects independently its own set of MPRs. MPRs are used to flood control messages from that node into the network while reducing the number of retransmissions that will occur in a region. Thus, the concept of MPRs is an optimization of a pure flooding mechanism.

The MPR set must be calculated by a node in a way such that it, through the neighbors in the MPR set, can reach all 2-hop neighbors. This means that the union of the neighbor sets the MPR nodes contains the entire 2-hop neighbor set. While it is not essential that the MPR set is minimal, it is essential that all 2-hop neighbors can be reached through the selected MPR nodes. The smaller a MPR-set, however, the more optimizations are achieved. By default, the MPR set can coincide with the entire neighbor set. This will be the case at network initialization.

The Figure 4 and following specifies a proposed heuristic for selection of MPRs. The following terminology will be used in describing this algorithm:

1. Find all 2-hop neighbors that can only be reached by one 1-hop neighbor. Assign those 1-hop neighbors as MPRs.
2. Determine the resultant cover set (i.e., the set of 2- hop neighbors that will receive the packet from the current MPR set).
3. From the remaining 1-hop neighbors not yet in the MPR set, find the one that would cover the most 2-hop neighbors not in the cover set.
4. Repeat from step 2 until all 2-hop neighbors are covered.

After selecting the MPRs among the neighbors, the link status of the corresponding one hop neighbors is changed from SYM_LINK to MPR_LINK in the neighbor table. The MPR set is re-calculated when:

- a change in the neighborhood is detected, i.e. either a symmetric link with a neighbor is failed, or a new neighbor with a symmetric link is added; or
- a change is detected in the 2-hop neighborhood such that a symmetric link is either detected or broken between a 2-hop neighbor and a neighbor.

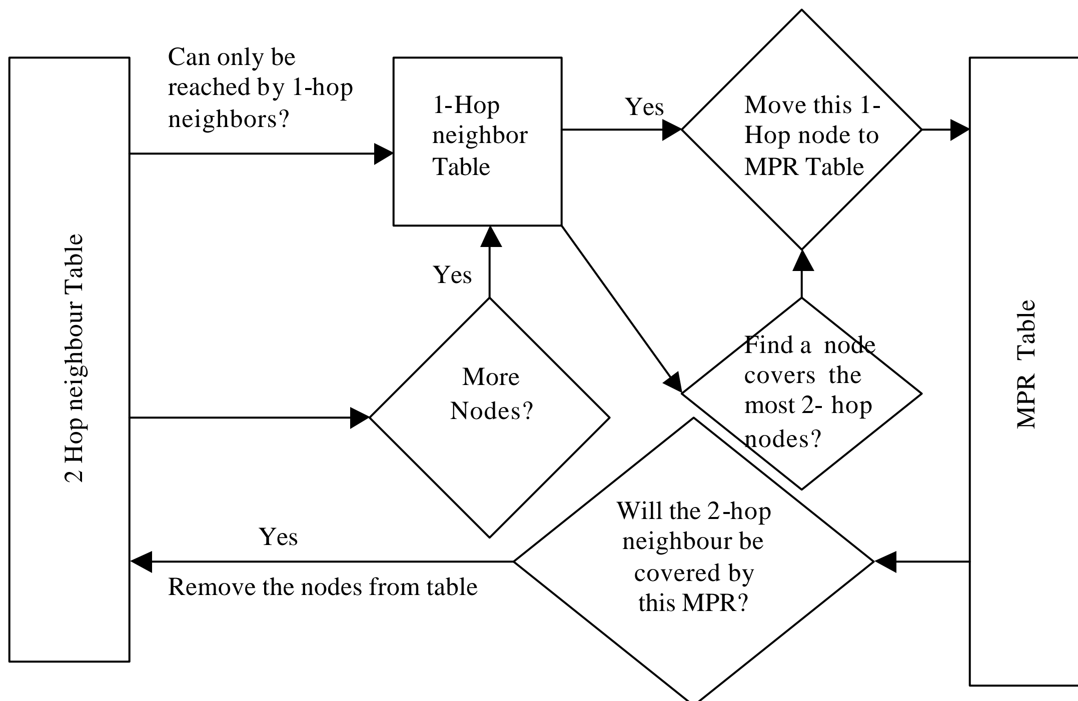


Figure 4: Flowchart of Multipoint Relay Selection

## V. Simulation results comparison and analysis

Our simulation is done in the ns-2.26 simulator installed on Linux Operating System. We use the AODV implementation in ns-2 downloaded from one of the author's web site, using IEEE 802.11 as the MAC layer protocol. The radio model simulates Lucent's WaveLAN with a nominal bit rate of 2Mb/sec and a nominal range of 250 meters. The radio propagation model is the two-ray ground model. Our application traffic is CBR (constant bit rate). The source destination pairs (connections) are chosen randomly. The application packets are all 512 bytes. We assumed a sending rate of 2 packets/second and 15 connections. The network area we set is 900 * 900 meters. The maximum node movement is 20 m/s, and maximum pause time is 10 seconds. We have simulated with 10, 50, 100, and 150 nodes using original AODV protocol and our implementation AODV+MPR protocol. Each data point represents an average of four runs using the identical traffic model, but with different randomly generated mobility scenarios. To preserve fairness, identical mobility and traffic scenarios are used for AODV and AODV+MPR. We study the performance of the following two metrics:

1. The packet delivery ratio represents the ratio of the data packets delivered to the destination to those generated by the CBR sources.
2. The average end-to-end delay of data packets includes all possible delays caused by buffering during routing discovery, queuing at the interface queue, retransmission at MAC layer, propagation, and transfer time.

Because of the long real-time simulation run for large network experiments, only four runs for each scenario were performed. The results of these runs were averaged together to produce the resulting graphs. From the trace file, we extracted the related data and calculated the packet delivery ratio and average end-to-end delay. The results are shown in Figure 4 and Figure 5. We can see that AODV+MPR achieves much better results in the dense network. There is less end-to-end delay and the package delivery ratio is much higher than the original AODV protocol. However from the results, we can see that the multipoint relay technique does not achieve better results, especially for the end-to-end delay, in the less dense network. This is because AODV+MPR takes longer time to discover the route from the source to destination.

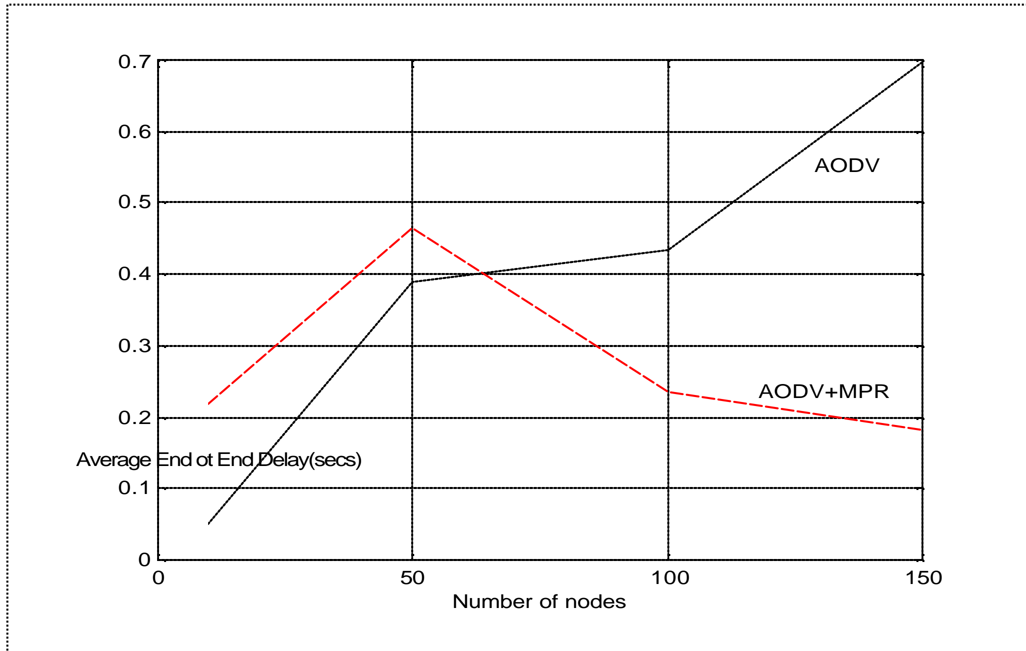


Figure 5: Average End to End Delay

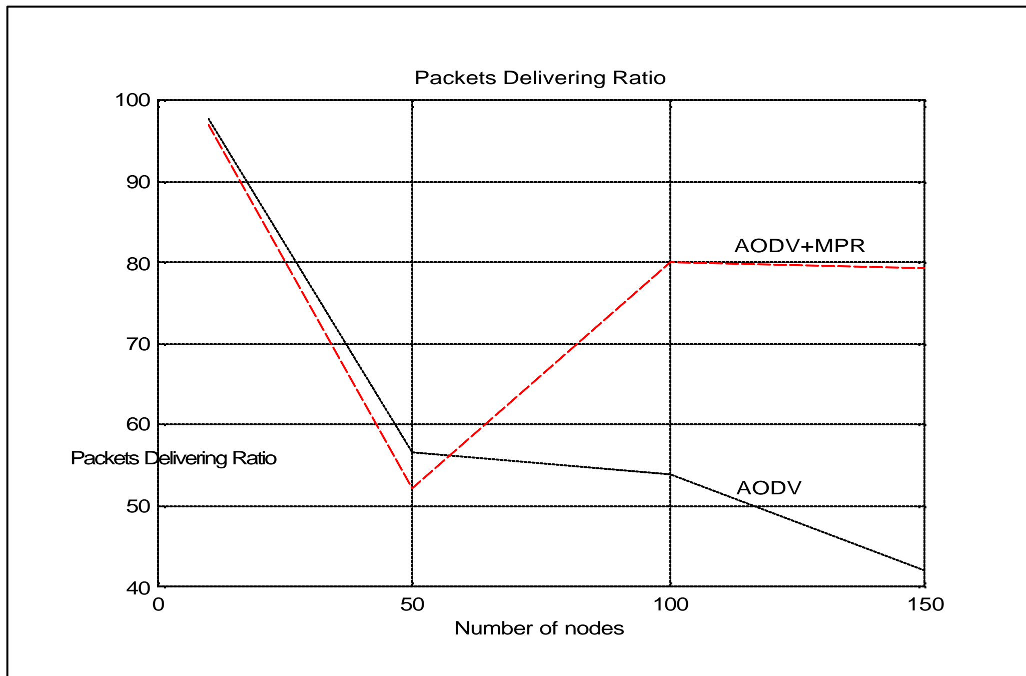


Figure 6: Package Delivery Ratio

## VI. Discussion and Future work

Due to the natures of mobile ad hoc networks - no infrastructure or centralized administration, characterized by node mobility and dynamic topology structure, scarce bandwidth, unreliable media and limited power supply, our implementation has to use the proactive HELLO message to obtain the neighbor information regularly, thus it causes background traffic overhead to exchange information between neighbors. Secondly, as the node mobility increase, accurate neighbor information collection is very hard. Our implementation works effectively only with complete neighbor topology information. As a result, the effectiveness of MPR technique is severely impaired by the increase of neighbor degree or node mobility.

The algorithm for calculating the MPR set is very complicated in this implementation; as a result, it takes huge amount of CPU power. It will be worth while to come up some more optimized and intelligent algorithms. The AODV protocol can also be further optimized by applying other techniques such as probability based methods or location based methods.

We would also like to do more simulation with different parameters such as more nodes in the network, higher mobility, increasing pause time. We believe that under different situations, the effectiveness of this technique will be different.

## VII. Conclusion

In this project, we first studied and simulated the AODV routing protocol. By analyzing the simulation results, we found that the current AODV protocol has major control overhead which is caused by "Route Query" flood packets. We have improved the AODV routing protocol by reducing routing overhead using an efficient flooding technique – multipoint relay. This technique selects the dominated nodes through out the entire network to forward route query flood packets. From the results of our simulations, we can see that MPR technique optimized original AODV protocol.

# VIII. References


[1] Yoav Sasson, David Cavin, André Schiper, "Probabilistic Broadcast for Flooding in Wireless Mobile Ad hoc Networks". IEEE Wireless Communications and Networking Conference (WCNC) - March 2003

[2] Zygmunt J. Haas, Joseph Y. Halpern, and Li Li, "Gossip-based ad hoc routing", In IEEE INFOCOM, Jun 2002.

[3] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu, "The broadcast storm problem in a mobile ad hoc network", In Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, (pages 151–162), Aug 1999.

[4] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, a. Qayyum et L. Viennot, "Optimized Link State Routing Protocol", IEEE INMIC Pakistan 2001.

[5] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir Das, "Ad Hoc On Demand Distance Vector (AODV) Routing", IETF Internet draft, draft-ietf-manet-aodv-12.txt, Nov. 2002 (Work in Progress).

# Appendix I

## Pseudo Code

✍✍ ***Hello message processing***

```
1.  IF  There exists a neighbor tuple with N_addr = Originator Address:
    THEN

1.1     IF (For that tuple) N_status == ASYM_LINK
        THEN

            IF  The node finds its own address among the
                addresses listed in the HELLO message (with Link
                Type ASYM_LINK, SYM_LINK or MPR_LINK),
            THEN
                It updates the N_status of the tuple to SYM_LINK and sets
                N_time = current time + NEIGHB_HOLDING_TIME.

            ELSE IF
                The node does not find its own address among the
                addresses listed in the HELLO message;
            THEN
                N_time = current time + NEIGHB_HOLDING_TIME.
            ENDIF

1.2     ELSE IF  (For that tuple)
            N_status == SYM_LINK OR
            N_status == MPR_LINK
        THEN
            IF The node finds its own address among the addresses
                listed in the HELLO message (with Link Type
                ASYM_LINK, SYM_LINK or MPR_LINK),
            THEN
                N_time = current time NEIGHB_HOLDING_TIME.
            ENDIF
        ENDIF
    ENDIF


2. ELSE
    A new neighbor tuple is created with:
    N_addr = Originator Address
    N_status with the value of SYM_LINK if the node finds its own address
    (with Link Type ASYM_LINK, SYM_LINK or MPR_LINK) among the addresses
    listed in the HELLO message, and to the value of ASYM_LINK otherwise
    N_time = current time + NEIGHB_HOLD_TIME
```

2-hop neighbor set updating:
For each 2-hop neighbor address listed in the HELLO message with Link Type
SYM_LINK or MPR_LINK:

**IF** A 2-hop tuple exists with:
N_addr == Originator Address AND
N_2hop_address == the address of the 2-hop neighbor
**THEN** The N_time of that tuple is set to:
N_time = current time + NEIGHB_HOLD_TIME

**ELSE**
A new 2-hop tuple is created with:
N_addr = Originator Address,
N_2hop_address = the address of the 2-hop neighbor,
N_time = current time + NEIGHB_HOLD_TIME
**ENDIF**

Updating MPR selector set:
IF A MPR selector tuple exists with:
MS_addr == Originator Address

**THEN**
The expiration time of that tuple is set to:
MS_time = current time + NEIGHB_HOLD_TIME.

**ELSE** A new MPR selector tuple is created with:
MS_addr = Originator Address
MS_time = current time + NEIGHB_HOLD_TIME
MSSN is incremented by one to indicate that the MPR
selector table has been changed.
**ENDIF**

# Appendex II

## Program Codes

Our program code files include aodv.cc, aodv.h, aodv_queue.cc, aodv_queue.h, which are C++ codes, totally about 3200 lines and 4 Otcl files, the only difference between those Otcl files is the number of nodes for simulation, we only attach one file-ensc835_50.octl as an example, this one is used for 50 nodes simulation. Since the codes file is too long, we list only part of them, and will submit the electrical copy.

### Aodv_queue.cc

```
#include <string.h>
#include <math.h>

#include "aodv_queue.h"
#include "aodv.h"


/*********************************************************************
 *  MessageQueue implementation
 */
int aodvQueue::numlinks=0;

aodvQueue::aodvQueue()
  : head(NULL), tail(NULL), peek_ptr(NULL)
{
  head=NULL;
  tail=NULL;
  peek_ptr=NULL;
  backup_peek=NULL;
  print_peek_ptr=NULL;
  temp_ptr=NULL;
}


void aodvQueue::QueueObject(AodvTuple *theObject)
{
    struct linknode *prev = tail;
    struct linknode *newnode=new struct linknode;
    newnode->object=theObject;
    newnode->N_time=CURRENT_TIME + AODV::Neighb_Hold_Time;
    theObject->parent=newnode;
    while (prev)
    {
        if (theObject->N_time < prev->object->N_time)
        {
            prev = prev->prev;  // Go up the queue
        }
        else
        {
```

```
            newnode->prev=prev;
          if ((newnode->next = prev->next))
            newnode->next->prev = newnode;
          else
            tail = newnode;
          prev->next = newnode;
          return;
        }
    }

    // theObject goes to top of the queue
    if ((newnode->next = head))
      head->prev = newnode;
    else
      tail = newnode;
    newnode->prev = NULL;
    head = newnode;
}  // end aodvQueue::QueueObject()

//added later to sort by sort value instead of time value
void aodvQueue::QueueObject(AodvTuple *theObject,float sortvalue)
{
  struct linknode *prev = tail;
  struct linknode *newnode=new struct linknode;
  //fprintf(stderr,"  %d \n",numlinks++);
  newnode->object=theObject;
  newnode->N_time=CURRENT_TIME + AODV::Neighb_Hold_Time;
  theObject->parent=newnode;
  while (prev)
    {
     if (theObject->N_sort < prev->object->N_sort)
       {
            prev = prev->prev;  // Go up the queue
       }
     else
       {
            newnode->prev=prev;
            if ((newnode->next = prev->next))
              newnode->next->prev = newnode;
            else
              tail = newnode;
            prev->next = newnode;
            return;
       }
    }

  // theObject goes to top of the queue
  if ((newnode->next = head))
    head->prev = newnode;
  else
    tail = newnode;
  newnode->prev = NULL;
  head = newnode;
```

```
}  // end aodvQueue::QueueObject()

AodvTuple* aodvQueue::FindObject(nsaddr_t id1, nsaddr_t id2)
{
  peek_ptr=head;
  while (peek_ptr) {
    if (peek_ptr->object->N_addr==id2){
      //    if (peek_ptr->object->N_2hop_addr==id2){
      if(((peek_ptr->object)->parents).FindObject(id1)){
             return peek_ptr->object;
      }
      if(((peek_ptr->object)->stepparents).FindObject(id1)){
             return peek_ptr->object;
      }
    }
    peek_ptr=peek_ptr->next;
  }
  return NULL;
}

AodvTuple* aodvQueue::FindObject(nsaddr_t id)
{
  int errorfix=0;
  peek_ptr=head;
  while (peek_ptr) {
    if(peek_ptr->object!=NULL) { // shouldn't need this at all remove me
      if (peek_ptr->object->N_addr==id){
             return peek_ptr->object;
      }
    }
    else {
      fprintf(stderr,"FindObject: Got a null object");
    }
    temp_ptr=peek_ptr;
    peek_ptr=peek_ptr->next;
  }
  return NULL;
}

AodvTuple* aodvQueue::FindNextObject(nsaddr_t id)
{
  if(peek_ptr)
    peek_ptr=peek_ptr->next;
  while(peek_ptr) {
    if (peek_ptr->object->N_addr==id) {
      return peek_ptr->object;
    }
    peek_ptr=peek_ptr->next;
  }
  return NULL;
}
```

```
void aodvQueue::printpeek()
{
  fprintf(stderr, "%p is peek ptr \n",peek_ptr);
}
void aodvQueue::RestoreBackupPeek()
{
  peek_ptr=backup_peek;
}
void aodvQueue::SetBackupPeek()
{
  backup_peek=peek_ptr;
}
int aodvQueue::checkCurrent()
{
  if(peek_ptr!=NULL)
   // fprintf(stderr, "time is %f and %f is experiation time \n",CURRENT_TIME,peek_ptr->N_time);
    if(peek_ptr->N_time<CURRENT_TIME){
      temp_ptr=peek_ptr;
      //RemoveCurrent();
      return 1;
    }
  return 0;
}
void aodvQueue::Clear()
{
  peek_ptr=head;
  while(peek_ptr!=NULL){
    temp_ptr=peek_ptr->next;
    //fprintf(stderr,"removing %d's pointer ",peek_ptr->object->N_addr);
    //fflush(stdout);
    free(peek_ptr);
    peek_ptr=temp_ptr;
  }
  head=NULL;
  tail=NULL;
}

void aodvQueue::RemoveCurrent()
{
  if(peek_ptr==NULL){
    peek_ptr=head;
  }
  temp_ptr=NULL;
  //fprintf(stderr,"%p peek pointer in RemoveCurrent\n",peek_ptr->next);
  if(peek_ptr){
    if (peek_ptr->prev){
      temp_ptr=peek_ptr->prev;
      peek_ptr->prev->next = peek_ptr->next;
    }
    else {
      head = peek_ptr->next;
      temp_ptr = NULL;
    }
```

```
    if (peek_ptr->next)
      peek_ptr->next->prev = peek_ptr->prev;
    else
      tail = peek_ptr->prev;
    free(peek_ptr);
    peek_ptr=temp_ptr;
  }
}  // end aodvQueue::Remove()
AodvTuple* aodvQueue::PeekInit()
{
  peek_ptr=head;
  if(peek_ptr!=NULL)
    return peek_ptr->object;
  return NULL;
}

AodvTuple* aodvQueue::PeekNext()
{
  if (peek_ptr!=NULL){
    peek_ptr=peek_ptr->next;
    if(peek_ptr!=NULL)
      return peek_ptr->object;
    return NULL;   // list has been traversed
  }
  else {
    peek_ptr=head;
    if(peek_ptr!=NULL)  //object was removed and peek_ptr was pointing at null
      return peek_ptr->object;
    else
      return NULL;     //last object was removed
  }
}

AodvTuple* aodvQueue::PrintPeekInit()
{
  print_peek_ptr=head;
  if(print_peek_ptr!=NULL)
    return print_peek_ptr->object;
  return NULL;
}

AodvTuple* aodvQueue::PrintPeekNext()
{
  if (print_peek_ptr!=NULL){
    print_peek_ptr=print_peek_ptr->next;
    if(print_peek_ptr!=NULL)
      return print_peek_ptr->object;
    return NULL;   // list has been traversed
  }
  else {
    return NULL;
  }
}
```

/*******************************************************************/

**aodv_queue.h**

```
#ifndef __aodv_queue_h__
#define __aodv_queue_h__

#include <aodv/aodv_rtable.h>
//#include <aodv/aodv_rqueue.h>

struct linknode {
  struct linknode *next;
  struct linknode *prev;
  class AodvTuple *object;
  double N_time;
};

class aodvQueue
{
  friend class  AODV;
    private:
      linknode *head, *tail;
                  linknode *peek_ptr,*temp_ptr, *print_peek_ptr,*backup_peek;
                  static int numlinks;

    public:
      aodvQueue();
      bool IsEmpty() {return (head == NULL);}
      void QueueObject(AodvTuple *theObject);
      void QueueObject(AodvTuple *theObject,float sortvalue);
      AodvTuple *FindObject(nsaddr_t id1,nsaddr_t id2);
                  AodvTuple *FindObject(nsaddr_t id);
                  AodvTuple *FindNextObject(nsaddr_t id);
      AodvTuple *Head() {return head->object;}
      void RemoveCurrent();
                  void Clear();
                  int checkCurrent();
                  void printpeek();
                  void RestoreBackupPeek();
                  void SetBackupPeek();
                  AodvTuple * PeekInit();
                  AodvTuple * PeekNext();
                  AodvTuple * PrintPeekInit();
                  AodvTuple * PrintPeekNext();
};



class AodvTuple
{
    friend class aodvQueue;

    public:
```

```
        nsaddr_t   N_addr;
        nsaddr_t   N_2hop_addr; //address fo neighbor
                u_int16_t   N_delay;  //used in routing
                int    N_status;  // ASYM_LINK SYM_LINK or MRP_LINK possible LOST_LINK
only used in 1 hop list
                double  N_sort;      // planed to be used in SOME list sorting
                double    N_time;    // Node id of message source (list sorted by this value)
                double              N_time2;
                int                hop;       // number of hop neighbor
                int                cdegree;    // used to find mprs current
                int                tdegree;   // used to find mprs total
                int                seq_num;  // used by dupicate and topology tables
                double              konectivity;  //used by historisis function
                int                recievedHello; //set to 0 on sending hello 1 on recieving


                aodvQueue            parents;  //queue of links
                aodvQueue            children; //queue of links
                aodvQueue            stepparents; //queue of links
        private:
        struct linknode *parent;
};


#endif
```

## ensc835_50.tcl

```
# ============================================================
# Define options
# ============================================================

set opt(chan)           Channel/WirelessChannel
set opt(prop)           Propagation/TwoRayGround
set opt(netif)          Phy/WirelessPhy
set opt(mac)            Mac/802_11
set opt(ifq)            Queue/DropTail/PriQueue
set opt(ll)             LL
set opt(ant)        Antenna/OmniAntenna

set opt(x)              900         ;# X dimension of the topography
set opt(y)              900             ;# Y dimension of the topography
set opt(cp)             "cbr-50-test"
set opt(sc)             "scen-50-test"

set opt(progress)     4          ;# progress markers

set opt(ifqlen)         50              ;# max packet in ifq
set opt(nn)             50          ;# number of nodes
set opt(seed)           1.0
set opt(cn)             15.0            ;# number of connections
set opt(rate)           4               ;# packet generating rate
```

```
;# Traffic pattern parameter
set opt(stop)              275              ;# simulation time
set opt(tr)                aodv50.tr        ;# trace file
set opt(nam)               aodvg.nam
set opt(rp)          AODV           ;# routing protocol script
set opt(lm)          "off"          ;# log movement
set opt(energymodel)    EnergyModel     ;
set opt(initialenergy)  100            ;# Initial energy in Joules


LL set mindelay_                 50us
LL set delay_                    25us
LL set bandwidth_                0         ;# not used
LL set off_prune_                0         ;# not used
LL set off_CtrMcast_             0         ;# not used

Agent/Null set sport_            0
Agent/Null set dport_            0

Agent/CBR set sport_             0
Agent/CBR set dport_             0
Agent/AODV set aodvDebugValue_  4     ;#debug level
Agent/AODV set Hello_Interval_ 1      ;#hello interal
Agent/AODV set Hello_Jitter_    0.5     ;#hello interal



Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 11*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0
Phy/WirelessPhy set bandwidth_ 11e6

# Initialize the 802.11 MAC
Mac set bandwidth_ 11e6

# ==========================================================
# Main Program
```

```
# ======================================================================

#
# Initialize Global Variables
#
set ns_          [new Simulator]
set topo [new Topography]
set god_      [new God]

#set namtrace  [open $opt(nam) w]
set traceall       [open $opt(tr) w]

$topo load_flatgrid $opt(x) $opt(y)
$ns_ use-newtrace
$ns_ trace-all $traceall
#$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

#$ns_ use-newtrace

#
# Create God
#
create-god $opt(nn)

#
#  Create the specified number of nodes $opt(nn) and "attach" them
#  the channel.
#  Each routing protocol script is expected to have defined a proc
#  create-mobile-node that builds a mobile node and inserts it into the
#  array global $node_($i)
#

# Create channel #1
set chan_1_ [new $opt(chan)]

#global node setting

$ns_ node-config -adhocRouting $opt(rp) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channel $chan_1_ \
                 -topoInstance $topo \
                 -energyModel $opt(energymodel) \
                 -macTrace OFF \
                 -agentTrace ON \
                 -routerTrace ON \
                 -movementTrace OFF \
                 -rxPower 0.3 \
```

```
                    -txPower 0.6 \
                    -initialEnergy $opt(initialenergy)

for {set i 0} {$i < $opt(nn) } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 1          ;# enable random motion
    $god_ new_node $node_($i)
}

#
# Source the Connection and Movement scripts
#

puts "Loading connection pattern..."
source $opt(cp)

puts "Loading scenario file..."
source $opt(sc)
puts "Load complete."

#
# stopping the simulation
#

for {set i 1} {$i <= $opt(progress)} {incr i} {
    set t [expr $i * $opt(stop) / ($opt(progress) + 1)]
    $ns_ at $t "puts \"completed through $t secs...\""
}

proc stop {} {
    global ns_ traceall namtrace
    close $traceall
    $ns_ flush-trace
    exit 0
}

for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}

$ns_ at $opt(stop).0002 "stop"

#
# Define node initial position in nam
#

for {set i 0} {$i < $opt(nn)} {incr i} {

    # positions will be set by scenario
    # The function must be called after mobility model is defined

    $ns_ initial_node_pos $node_($i) 20          ;# whatever
}
```

```
puts "Starting Simulation..."
$ns_ run
```