# MAODV Implementation for NS-2.26

Yufang Zhu and Thomas Kunz
Systems and Computer Engineering
Carleton University

## 1. Introduction

MAODV [6] [7] is the multicast extension of AODV [5]. Both AODV and MAODV are routing protocols for ad-hoc networks, with AODV for unicast traffic and MAODV for multicast traffic. NS2 [3], a widely used simulation tool, includes a standard implementation of the AODV protocol to analyse its performance [1], upon which we developed our initial MAODV implementations [2] [8]. But those MAODV implementations have two key limitations: 1) Only group members can send multicast traffic to the multicast group; 2) The multicast data packets are unicast, resulting in wasted bandwidth. This new version of MAODV allows each node in the network to send out multicast data packets, and the multicast data packets are broadcast when propagating along the multicast group tree. In addition, several functions are improved as well in this new version.

## 2. The Core of MAODV

Each multicast group has a unique multicast group address. According to the MAODV specification [6] [7], each multicast group is organized by using a tree structure, composed of the group members and several routers, which are not group member but must exist in the tree to connect the group members. We say the group members and the routers are all tree members and belong to the group tree. Associated with each multicast tree, the group member that first constructs the tree is the group leader for that tree, responsible for maintaining the group tree by periodically broadcasting Group-Hello (GRPH) messages in the whole network. The group leader also maintains the group sequence number, which is propagated in the network through the GRPH.

Each node in the network may maintain three tables. The first one is the *Unicast Route Table*, recording the next hop for routes to other destinations for unicast traffic. Usually, the destination is one of the other nodes in the network. A special case is when the destination is a multicast address, which happens when the node is not a multicast tree member but has multicast data packets to send to that multicast group. The second one is the *Multicast Route Table*, listing the next hops for the tree structure of each multicast group. Each entry represents one group tree structure. Every node that belongs to that group tree should maintain such entries, with its own identity as group leader, group member, or router (non-multicast member that is in the tree to provide connectivity). Every next hop is associated with direction either downstream or upstream. If the next hop is one-hop nearer to the group leader, the direction is upstream; otherwise, the direction is downstream. The group leader has no upstream, while other nodes in the tree should have one and only one upstream. The third table is the *Group Leader Table*. It records the currently-known multicast group address with its group leader address and the next hop towards that group leader when a node receives a periodic GRPH message. It includes the function of the *Request Table* described in [6].

## 3. Route Discovery and Maintenance for Reaching a Specific Node

This is the main task of AODV protocol and there is a standard implementation in NS2 [1] [3]. Two things we should mention again are: 1) only MAC layer detection is used for detecting broken links on the active routes, either the route to a specific node or the route to a multicast tree. We only use one-hop Neighbor-Hello to detect the link breakage in multicast group tree. 2) Although the AODV implementation in the latest version of NS2 (2.26) does implement the local repair for link breakage, in our implementation and simulation, we ignore local repair and still let the data source node initiate discovery for a new route instead.

## 4. Route Discovery and Maintenance for Reaching a Multicast Tree

As this new version of MAODV implementation supports that every node in the network can send out multicast traffic, we must consider how these data packets reach each multicast group member if the data source node is not a tree member. We choose a two-step approach: first step, there is a route from that data source node to a tree member; then after the tree member receives the multicast data packets, it propagates the data through the whole tree, reaching every group member.

We borrow the mechanism used for route discovery and maintenance for reaching a specific node in AODV to accomplish the first step. The data source node initiates a RREQ to ask for a route to that multicast address. Usually, this RREQ is the same as the RREQ used in AODV, without any flags and broadcasted in the network. But with the *Group Leader Table*, the source node may already know a route to reach the group leader. By using the information recorded in the *Group Leader Table*, the RREQ can be sent unicastly towards the group leader if this is the first time the node sends RREQ. During RREQ propagation, the reverse route towards the source node is constructed as described in the AODV protocol. Any node not in the multicast tree but with fresh enough route to that multicast address, or any tree member with known group leader can respond to this RREQ with a RREP. While the RREP is sent back to the source node along the reverse route, every intermediate node and the source node updates the route to that tree member with the destination address set to the multicast group address, thus the forwarding route is established in their *Unicast Route Tables*. For this first step, the end node is a tree member. The second step is accomplished under multicast tree construction, which is described in the “Multicast Tree Construction” section.

During multicast data packet forwarding, each node first checks if itself is in the multicast tree. If it is not a tree member, it will check its *Unicast Route Table* to find the next hop for the multicast address. If it has the information, the data packets are forwarded towards the next hop; otherwise, it will send an unsolicited RREP back to the source node, in order to let the source node initiate a new route discovery if it still needs a route to that multicast address. If the node itself is a tree member, it will follow its *Multicast Route Table* to forward the packets.

When using the *Unicast Route Table* to forward multicast data packets, we use the MAC layer detection to detect link breakage on the route and use the unsolicited RREP to inform the data source node to discover another route if necessary.

## 5. Multicast Tree Construction

The same RREQ and RREP messages used in AODV are adapted to be used for tree construction in MAODV. In addition, MACT message is introduced to finish the last step of tree construction.

A node initiates a RREQ with a join flag (RREQ-J) when it is not a tree member and wants to join that multicast group. Before sending out RREQ-J, the node creates an entry in its *Multicast Route Table*, and identifies itself as a group member, but with an unknown group leader address, and without any upstream and downstream next hop. If a node in the tree but not a group member wants to become a group member, it simply changes its identity recorded in its *Multicast Route Table*, from a router to a group member.

Usually, RREQ-J is broadcast in the network, but if the node can get information about the group leader and how to reach that group leader through checking its own *Group Leader Table*, and this is the first time that it sends out RREQ-J, RREQ-J can be sent unicastly towards the group leader. During RREQ-J propagation, the reverse route back to the request source node is established in the *Unicast Route Table*.

When a node receives a non-duplicate RREQ-J, only the tree members with same or larger multicast group sequence number can respond to the RREQ-J with a RREP-J (RREP with a join flag). While RREP-J travels back along the reverse route, it means this route may be a potential branch to the tree. So in the *Multicast Route Table*, we cache the information about the upstream next hop towards the tree without adding any new valid next hop in the entry. This cache can only cache one potential upstream. If later the node receives another RREP-J indicating a better branch (i.e. larger multicast group sequence number or smaller hop count with the same multicast group sequence number), it will cache the new one and propagate RREP-J towards the request source node; otherwise, discard the later RREP-Js.

A new message, Multicast Route Activation (MACT), is used for grafting a branch to the tree. When the request source node sends out RREQ-J and waits for a specific time RREP_WAIT_TIME, it checks if it already received any RREP-J and cached the corresponding information. If it has the information, it sends out a MACT with a join flag (MACT-J) towards the cached upstream node and adds the new next hop in its *Multicast Route Table*. Every node that receives MACT-J should add a new next hop indicating the downstream from where it receives the MACT-J in its *Multicast Route Table*. Then if it is a tree member, the branch is finally grafted; otherwise, it will check its own cache to find the potential upstream next hop towards the tree, add this next hop in its *Multicast Route Table*, and send out a MACT-J towards that node as well.

If the request source node tries several times (RREQ_RETRIES) to join a group tree, but has not received any RREP-J, which means in the network there is no such group, or the request source node cannot reach that group due to network partition. Then, the request source node becomes the first node in that group and acts as the group leader to maintain the group sequence number and tree structure.

We use broadcast to accomplish the multicast data packet forwarding in the tree in order to save

bandwidth. But we do not know if all the neighbors in the tree receive the data correctly, because we cannot reserve the channel to get feedback about reception. In order to provide more chance for data reception, every node in the tree, except for leaf nodes.

## 6. Multicast Tree Maintenance

Multicast tree maintenance is much more complicated than unicast route maintenance, as the maintenance includes Periodic Group-Hello Propagation, Neighbor Connectivity Maintenance, Group Leader Selection, Membership Revocation, and Tree Merge.

### 6.1 Periodic Group-Hello Propagation

The group leader must periodically broadcasts a Group-Hello message (GRPH) throughout the whole network, to indicate the existence of that group and its current status. When receiving a GRPH message, each node updates its *Group Leader Table*, to indicate the group leader and the route towards the group leader. Then the node that is not a tree member retransmits the first-time received GRPH.

A node that is a tree member and receives GRPH from its own upstream can use the GRPH to update their current group sequence number, current group leader and the current distance from the group leader, which requires the GRPH to be propagated from upstream to downstream step by step in its own tree structure. If a tree member receives a GRPH not from its own upstream, it first checks its group leader information recorded in its *Multicast Route Table*. If it is the same group leader as indicated in the GRPH, this GRPH is discarded and the node waits for next GRPH from its own upstream. If the group leader information recorded in its *Multicast Route Table* is not the same as indicated in the GRPH, there exists another tree with the same multicast group address but not the same group leader, and these two trees can be connected. So tree merge is initiated, which is described in the "Tree Merge" Section. The tree merge is initiated by the tree member with a group leader whose address is smaller than the leader address indicated in the GRPH. If its leader address is larger than that indicated in the GRPH, just discard this GRPH.

### 6.2 Neighbor Connectivity Maintenance

Because multicast traffic is broadcast through the tree, we cannot use the MAC layer detection to find link breakage in the tree. So we have to use periodic one-hop Neighbor-Hello messages. To reduce this one-hop Neighbor-Hello overhead, if a node already sent a broadcast message (including data packets), it can delay the transmission of the Neighbor-Hello.

Tree neighbor connectivity is maintained by local repair when the downstream node of a link in the tree realizes that the link is broken by not receiving any broadcast messages from that neighbor in a specific time (usually three times of Neighbor-Hello interval). After detecting a link breakage, the downstream node first deletes that next hop (its only upstream) in its *Multicast Route Table*, and then becomes the request source node sending out RREQ-J to discover a new branch. This RREQ-J is different from the RREQ-J used for the node that is not a tree member but wants to join the multicast group, in that this RREQ-J must be broadcast and attached with an extension including additional information about the node's hop count to the group leader, in order to avoid the old branch and its own downstream nodes responding to the RREQ-J. Besides the criteria for

responding to a RREQ-J (only tree member with same or larger group sequence number can respond), when a tree member receives a RREQ-J with Extension, it must check its own hop count to the group leader, only the tree member with equal or smaller hop count can respond with a RREP-J. The remaining process after sending out RREP-J is the same as the process for a non-member node wanting to join the multicast group. Other than that, if the local repair makes the request source node change its group information such as group leader, group sequence number or hop count to group leader, it will send out a GRPH with a update flag (GRPH-U) to its downstream nodes one by one unicastly to update their group information.

If the request source node tries several times (RREQ_RETRIES) to repair that branch, but has not received any RREP-J, tree partition happened due to network partition. Then, a new group leader must be selected for this partitioned tree. The group leader selection is described in the "Group Leader Selection" Section.

If it is not the downstream node but the upstream node that realizes link breakage, the upstream node deletes that next hop (one of its downstream links) in its *Multicast Route Table*. If this upstream node is not a group member and a leaf node without any downstream, it stays for a while in the tree and after that if it still is a leaf node, it begins self-prune as described in the "Membership Revocation" Section.

### 6.3 Group Leader Selection

A new group leader must be selected for the partitioned tree or when the group leader revokes its group membership. When tree partition happens, and the current node is a group member, it will become the new group leader. Otherwise, it will force one of its tree neighbors to be the leader. All its neighbors are downstream nodes, because the partitioned node has no upstream and the previous group leader has no upstream either.

If the current node only has one downstream node, it cancels the entry for that group in its *Multicast Route Table*, indicating it is no longer belongs to the tree, and sends out a MACT with a prune flag (MACT-P) to this downstream node, indicating that it will leave the tree and the tree needs a leader. If the current node has more than one downstream node, it selects one downstream, change its direction from downstream to upstream, and sends a MACT with a group-leader flag (MACT-GL) towards that node, indicating that it has other branch(es) in the tree and the tree needs a leader.

So the downstream node can receive either a MACT-P or a MACT-GL from its upstream node. When receiving MACT-P from its upstream, the node removes its upstream link from its *Multicast Route Table*. When receiving MACT-GL from upstream, the node changes the upstream direction into downstream. Then if the node is a group member, it will be the new group leader. Otherwise, it continues the above procedure till a group member is reached and becomes the new group leader.

Once the new group leader is selected, it begins to maintain the tree and broadcast GRPH periodically in the whole network. If it has any downstream nodes, it will also send GRPH-U to

every downstream node unicastly, to indicate the new leader and update the group information in their *Multicast Route Tables*.

### 6.4 Membership Revocation

A group member including the group leader can revoke its group membership at any time. If this node is the group leader, it changes its own identity to a router, and a new group leader must be selected. The group leader selection is described in the "Group Leader Selection" Section.

If the node is not the group leader, it first discards its membership by changing its own identity to a router, and then checks if it has any downstream node. If it has downstream nodes, it must stay in the tree to connect group member. Otherwise, it can self-prune from the multicast tree.

Self-pruning causes the node to leave the multicast tree, so it must update its *Multicast Route Table* to cancel the entry for that multicast address. Then it sends a MACT with a prune flag (MACT-P) to its upstream node to inform the upstream that it no longer belongs to the tree. If receiving a MACT-P makes the upstream node a leaf and it is also not a group member, it can similarly prune itself from the tree with the same action. The procedure terminates when a group member or non-leaf tree member is met.

Although this MACT-P is the same as the MACT-P used for group leader selection, this MACT-P is sent from downstream to upstream, while the MACT-P used for group leader selection is sent from upstream to downstream.

### 6.5 Tree Merge

Tree merge can be detected when a tree member with a smaller group leader address receives a GRPH generated by another group leader with a larger address for the same group. Different from the previous implementation for MAODV, this implementation mainly follows the tree merge procedure described in [6]. But different from [6], where only group membesr in the tree with the smaller group leader address can initiate tree merge, in our implementation, any tree member in the tree with the smaller group leader address can initiate a merge.

The tree member initiates the merge by unicastly sending a RREQ with a repair flag (RREQ-R) to the group leader, asking for the leader's permission to rebuild the tree. This RREQ-R propagates from downstream to upstream till the leader is reached. If the leader has not permitted other nodes to rebuild the tree, it can send back a RREP with a repair flag (RREP-R) to that request node. When receiving RREQ-R, the reverse route to the request node is formed, so the RREP-R follows this reverse route to the request node. One special case is that the leader itself realizes there is another tree for that group with a group leader having a larger address. In this case, the RREQ-R and RREP-R cycle is omitted and if the leader has not permitted any other tree member to rebuild the tree, it will start to rebuild the tree.

Tree-rebuilding starts when the request node unicastly sends a RREQ with a join-and-repair flag (RREQ-JR) to the group leader with larger address. Because when the request node receives the GRPH from that group leader, it already records the route to that group leader in its *Group Leader*

*Table*, so we use the information in the *Group Leader Table* to reach the other tree. Once the tree member in the other tree is reached, the RREQ-JR is sent unicastly from downstream node to upstream node till the group leader is reached. During the travel of RREQ-JR, the reverse route back to the request node is established as well. When the group leader with the larger address receives the RREQ-JR, it sends back RREP-JR to the request node along the reverse route. During the travel of RREP-JR, the group information, such as group leader address, group sequence number, and hop count to group leader, is updated. When the RREP-JR travels in the tree with the larger group leader address, it simply propagates from upstream to downstream. If a node that is not a tree member is reached, this node becomes a router for the new tree, adding two next hops in its *Multicast Route Table*, indicating the corresponding upstream and downstream node. When the tree member in the tree with the smaller group leader address is reached, it adds a next hop from which it receives the RREP-JR, indicating the new upstream node, changes the old upstream node to downstream direction, and relays the RREP-JR to the old upstream node. Then the RREP-JR propagates from downstream to upstream in the tree with the smaller group leader address. At every step, the current node changes the downstream direction to upstream for the next hop from which it receives RREP-JR, changes the old upstream to downstream, and forwards the RREP-JR towards the old upstream node. So when RREP-JR reaches the group leader with smaller address, this group leader updates one of its own downstream nodes to upstream node, and changes its identity from group leader to group member, thus the new tree is completely built. After that, the old group leader should unicast GRPH-U to its downstream, indicating the change of group information.

## 7. Control Messages in Summary

There are four types of *Route Requests*: RREQ, RREQ-J, RREQ-R and RREQ-JR. RREQ is used under the following two situations: 1) unicast route discovery and maintenance for reaching a specific node; 2) unicast route discovery and maintenance for reaching a multicast group, when a node is not a multicast tree member but has multicast data packet(s) to send to that multicast group without knowing how to reach that tree. RREQ-J is used under the following two situations: 1) when a node is not a multicast tree member but wants to join the multicast group; 2) link breakage in the tree. RREQ-R and RREQ-JR are used for tree merge.

Corresponding to different *Route Requests*, there are four different *Route Replies*: RREP, RREP-J, RREP-R and RREP-JR.

There are three types of MACT: MACT-J, MACT-P and MACT-GL. MACT-J is used for tree construction when a non-member node wants to join the multicast group or when a link breakage is repaired in the tree. MACT-P is used for pruning a node from the tree if received from downstream. If received from upstream, MACT-P indicates not only pruning but also selecting a new group leader. MACT-GL is used for new group leader selection.

There are two types of GRPH: GRPH, GRPH-U. GRPH is periodically sent out from the group leader in the whole network. GRPH-U is sent out from an upstream node to downstream nodes in the tree to change the group information.

The last one is the one hop Neighbor-Hello, used for detecting link breakage in the tree.

## 8. Proactive Approach

We add the proactive connection maintenance feature to the tree maintenance in MAODV implementation. The basic idea is to predict the link breakage time of an active link in the tree before the breakage actually happens, then a new connection, excluding that soon-to-be-broken link, is pro-actively constructed before the old one actually becomes unavailable, in order to avoid the loss of data packets on that link. The details of this idea are discussed in [8].

## 9. MAODV Software Package

### 9.1 Implementation Environment

Operating System: Linux Red Hat 9.0
NS2 version: ns-allinone-2.26

### 9.2 MAODV Implementation Results

We run several simulations with the proactive approach feature and below are the results that can be compared with the results in [4].

The simulation environment is:

1) Area: 1500 x 300 meters;
2) Number of nodes: 50;
3) Simulation duration: 910 seconds;
4) Number of repetitions: 7;
5) Physical/Mac Layer: IEEE 802.11 at 2Mbps, 250 meter transmission range;
6) Mobility model: random waypoint model with no pause time, and node movement speed 0m/s, 1m/s or 20m/s.
7) Each sender sends 2 multicast data packets per second with each packet 256 bytes long;
8) Each receiver is a multicast group member, but each send is not group member (except for the case with 50 receivers, where all nodes are group members);
9) All receivers join a single multicast group at the beginning of the simulation, and the senders start sending data 30 seconds later. After 900 seconds, all senders stop transmitting data;
10) Only multicast traffic exists in the simulation.

PDR is the Packet Delivery Ratio, simply calculated as total received packets / (total send packets * the number of receivers). Latency is the average delay for data transfer from a sender to a receiver.

PDR and latency for MAODV with proactive approach, no movement

| | 1 Sender | | 2 Senders | | 5 Senders | | 10 Senders | |
|---|---|---|---|---|---|---|---|---|
| | PDR | Latency | PDR | Latency | PDR | Latency | PDR | Latency |
| 10 Receivers | 0.9977 | 0.0337 | 0.9740 | 0.0380 | 0.9301 | 0.0358 | 0.8707 | 0.0394 |
| 20 Receivers | 0.9950 | 0.0273 | 0.9355 | 0.0370 | 0.9098 | 0.0342 | 0.8051 | 0.0362 |
| 30 Receivers | 0.9910 | 0.0277 | 0.9534 | 0.0428 | 0.9084 | 0.0341 | 0.8223 | 0.0420 |
| 40 Receivers | 0.9928 | 0.0215 | 0.9607 | 0.0271 | 0.9020 | 0.0293 | 0.8190 | 0.0320 |
| 50 Receivers | 0.9889 | 0.0334 | 0.9629 | 0.0408 | 0.9022 | 0.0366 | 0.8367 | 0.0392 |

PDR and latency for MAODV with proactive approach, 1m/s maximum speed

| | 1 Sender | | 2 Senders | | 5 Senders | | 10 Senders | |
|---|---|---|---|---|---|---|---|---|
| | PDR | Latency | PDR | Latency | PDR | Latency | PDR | Latency |
| 10 Receivers | 0.9886 | 0.0251 | 0.9688 | 0.0274 | 0.0922 | 0.0296 | 0.8448 | 0.0357 |
| 20 Receivers | 0.9897 | 0.0265 | 0.9586 | 0.0310 | 0.8966 | 0.0344 | 0.7969 | 0.0379 |
| 30 Receivers | 0.9599 | 0.0328 | 0.9543 | 0.0338 | 0.8552 | 0.0352 | 0.7699 | 0.0393 |
| 40 Receivers | 0.9502 | 0.0262 | 0.9369 | 0.0318 | 0.8695 | 0.0327 | 0.7600 | 0.0363 |
| 50 Receivers | 0.9695 | 0.0293 | 0.9467 | 0.0327 | 0.8602 | 0.0322 | 0.7885 | 0.0366 |

PDR and latency for MAODV with proactive approach, 20m/s maximum speed

| | 1 Sender | | 2 Senders | | 5 Senders | | 10 Senders | |
|---|---|---|---|---|---|---|---|---|
| | PDR | Latency | PDR | Latency | PDR | Latency | PDR | Latency |
| 10 Receivers | 0.9114 | 0.0253 | 0.9076 | 0.0311 | 0.8724 | 0.0314 | 0.7913 | 0.0341 |
| 20 Receivers | 0.9201 | 0.0277 | 0.9148 | 0.0311 | 0.8573 | 0.0325 | 0.7619 | 0.0367 |
| 30 Receivers | 0.9173 | 0.0282 | 0.8947 | 0.0299 | 0.8503 | 0.0338 | 0.7483 | 0.0382 |
| 40 Receivers | 0.9239 | 0.0282 | 0.8869 | 0.0311 | 0.8316 | 0.0342 | 0.7265 | 0.0419 |
| 50 Receivers | 0.8926 | 0.0261 | 0.8903 | 0.0282 | 0.8106 | 0.0297 | 0.7252 | 0.0323 |

While the results for low mobility and few multicast senders are rather encouraging, the protocol performance deteriorates for increased mobility and higher traffic (in the form of more multicast senders). On the plus side, the average packet latency is remarkably consistent across all scenarios.

### 9.3 MAODV Implementation Installation

First, the NS2 simulation must be installed. Download the NS2 package from http://www.isi.edu/nsnam/ns/, and type "./install" in the directory of extraction folder: ns-allinone-2.26.

In our MAODV Package (as ZIP file), there are a total of 18 files:

1) aodv.h;
2) aodv.cc;
3) aodv_mcast.cc;
4) aodv_mtable_aux.cc;
5) aodv_mtable_aux.h;

6) aodv_mtable.cc
7) aodv_mtable.h;
8) aodv_packet.h;
9) aodv_rqueue.cc;
10) aodv_rqueue.h;
11) aodv_rtable.cc;
12) aodv_rtable.h;
13) cmu-trace.cc;
14) wireless-phy.cc;
15) wireless-phy.h;
16) node.cc;
17) node.h;
18) ns-mcast.tcl.

Copy files 1) – 12) to directory ./ns-2.26/aodv/; copy file 13) to directory ./ns-2.26/trace/; copy files 14) and 15) to directory ./ns-2.26/mac/; copy files 16) and 17) to directory ./ns-2.26/common; and finally copy file 18) to directory ./ns-2.26/tcl/mcast/.

The default configuration in aodv.h includes:

```
#define AODV_LOCAL_REPAIR
#define AODV_LINK_LAYER_DETECTION
#define IMPROVEMENT
#define MULTICAST
#define PREDICTION
```

"**`#define AODV_LOCAL_REPAIR`**" and "**`#define AODV_LINK_LAYER_DETECTION`**" are the original configuration options in aodv.h. If we compile the protocol with only these two options, only unicast traffic can be simulated, and the simulation results should be the same as the results generated by the original protocol implementation distributed in version 2.26

"**`#define IMPROVEMENT`**" is used only for unicast packets to be transmitted for the second time when the first-time transmission failed. If the second-time transmission still fails, the packets will be dropped. We give the packets one more chance to be sent out, because we observe that in version 2.26, the packet delivery ratio will be improved dramatically by trying this.

We explicitly add the multicast configuration as "**`#define MULTICAST`**".

The proactive approach feature can be controlled by "**`#define PREDICTION`**". So if the proactive approach is included without multicast traffic, the proactive route maintenance [8] for unicast traffic will be invoked.

There is another feature, added as "**`#define UPPER_LEVEL_RECEIVE`**" (by default, it is inactive). It allows the upper level agent such as a UDP agent to receive the multicast data packets. The agent port number must be known in advance and defined in aodv.h, (for example, "**`#define`**

**`UPPER_LEVEL_PORT 100`**). This feature is only used for multicast traffic, used for multicast group member receiving multicast data packets at the same port number. In order to make it work, the traffic file must be carefully organized (please see the example below).

The top-level `Makefile` (in directory ns-2.26) needs to be changed, adding `aodv_mcast.o`, `aodv_mtable.o`, and `aodv_mtable_aux.o` to the compilation list.

Under directory ./ns-2.26/, type "`make clean`", to remove all *.obj files, the recompile NS2 with "`make`" to get the new MAODV implementation.

### 9.4 Creating Mobile Node Movement Scenario Files

Under directory: ns-allinone-2.26/ns-2.26/indep-utils/cmu-scen-gen/setdest, run:

```
./setdest [-n num_of_nodes] [-p pausetime] [-s maxspeed] [-t simtime]
[-x maxx] [-y maxy] > [output-file]
```

### 9.5 Creating CBR Traffic Pattern Scenario Files

For unicast traffic, under directory: ns-allinone-2.26/ns-2.26/indep-utils/cmu-scen-gen, run:

```
ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-seed seed] [-mc connections]
[-rate packet/second for one connection]>[output-file]
```

For multicast, the traffic pattern scenario is like: (0xE000000 is a multicast address)

```
set udp_(0) [new Agent/UDP]
$udp_(0) set dst_addr_ 0xE000000

#this is for upper level agent receiving multicast traffic
#udp_(0) set dst_port_ 100
##########################################################

$ns_ attach-agent $node_(1) $udp_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 64
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$cbr_(0) set dst_ 0xE000000
$ns_ at 30.00000000000000 "$cbr_(0) start"

$ns_ at 0.010000000000000 "$node_(2) aodv-join-group 0xE000000"

#this is for upper level agent receiving multicast traffic at group member
#set null_(0) [new Agent/Null]
#$node_(2) attach $null_(0) 100
##########################################################
```

### 9.6 Simulation Script

Below is a sample for simulation script (.tcl file):

```
set opt(stop) 910.0
set nodes  50
set mobility  1
set scenario  1
set pausetime 0
set traffic   cbr
set senders   1
set receivers   10

set ns_ [new Simulator]
set topo [new Topography]
$topo load_flatgrid 1500 300

set tracefd
[open ./trace-$pausetime-$mobility-$scenario-$senders-$receivers w]
$ns_ trace-all $tracefd

set god_ [create-god $nodes]
$ns_ node-config -adhocRouting AODV \
           -llType LL \
           -macType Mac/802_11 \
           -ifqLen 50 \
           -ifqType Queue/DropTail/PriQueue \
           -antType Antenna/OmniAntenna \
           -propType Propagation/TwoRayGround \
           -phyType Phy/WirelessPhy \
           -channel [new Channel/WirelessChannel] \
           -topoInstance $topo \
           -agentTrace ON \
           -routerTrace ON \
           -macTrace OFF \
           -movementTrace OFF

for {set i 0} {$i < $nodes} {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0;
}

puts "Loading connection pattern ..."
```

```
source "../$traffic/$traffic-$senders-$receivers-2-256.aodv.$nodes"

puts "Loading scenarios file..."
source
"../scen-1500x300/scen-1500x300-$nodes-$pausetime-$mobility-$scenario
"

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

$ns_ at $opt(stop) "$ns_ halt"

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

### 9.7 Statistic Report

Below is a sample file (.perl) for simulation result analysis:

```
#!/usr/local/bin/perl
$totalReceivers = 10;
$totalNodes = 50;
$totalPid = 20000;

($traceFile, $outFile) = @ARGV;
$outFile = ">".$outFile;
open (MYFILE, $traceFile) || die "cannot open the trace file\n";
open (PACKET, $outFile) || die "cannot open the output file\n";

$i = 0;
while($i < $totalPid){
  $send[$i] = 0;
  $sendTime[$i] = 0;
  $j = 0;
  while($j<$totalNodes){
    $recv[$i][$j] = 0;
    $latency[$i][$j] = 0;
    $j++;
  }
  $i++;
}

while (<MYFILE>)
{
```

```
  $current_time = findTime($_);
  if (m/cbr/ && m/^s/ && m/AGT/) {
    $id = findPacketId($_);
    $send[$id] = 1;
    $sendTime[$id] = $current_time;
  }
  elsif(m/cbr/ && m/^r/ && m/RTR/){
    $id = findPacketId($_);
    $node = findNodeId($_);
    if ($recv[$id][$node] == 0){
      $recv[$id][$node] = 1;
      $latency[$id][$node] = $current_time - $sendTime[$id];
    }
  }
}

print "almost done!\n";
close (MYFILE);

$totalSend = 0;
$totalRecv = 0;
$totalLatency = 0;
$i = 0;
while ($i < $totalPid){
  if ($send[$i] == 1){
    $totalSend++;
    $j = $totalNodes - $totalReceivers;
    if ($j == 0 ) {$j = 1;}
    while ($j < $totalNodes){
      if ($recv[$i][$j] == 1){
        $totalRecv++;
        $totalLatency += $latency[$i][$j];
      }
      $j++;
    }
  }
  $i++;
}

print PACKET "The number of receivers is : ", $totalReceivers, "\n";
print PACKET "The number of sent-out packets is: ", $totalSend, "\n";
print PACKET "The number of received packets is: ", $totalRecv, "\n";
if ($totalReceivers != $totalNodes) {
  print PACKET "Ratio is ", $totalRecv/($totalSend*$totalReceivers),
```

```
"\n";
}
else {
  print PACKET "Ratio is ", $totalRecv/($totalSend*($totalReceivers - 1)),
"\n";
}

print PACKET "Latency is ", $totalLatency/$totalRecv, "\n";

close (PACKET);

sub findPacketId {
    @fields = split(/ /, $_[0]);
    $pid = $fields[6];
    if ($fields[4] != "") {$pid = $fileds[5];}
    return $pid;
}

sub findTime {
    @fields = split(/ /, $_[0]);
    $theTime = $fields[1];
    return $theTime;
}

sub findNodeId {
  $startPo = index($_[0], "_")+1;
  $endPo = index($_[0], "_", $startPo);
  $len = $endPo - $startPo;
  $node = substr($_[0], $startPo, $len);
  return $node;
}
```

## 10. References

[1] Broch, J.; Maltz, D. A.; Johnson, D. B.; Hu, Y.-C. and Jetcheva. J.; "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98), Dallas, TX, USA, October 1998, pages 85-97.

[2] Cheng. E.; "On-Demand Multicast Routing in Mobile Ad Hoc Networks", M. Eng. Thesis, School of Computer Science, Carleton University, January 2001.

[3] http://www.isi.edu/nsnam/ns/

[4] Kunz, T.; "Reliable multicasting in MANETs", Contractor Report, Communications Research Centre, Ottawa, Canada, July 2003

[5] Perkins, C. E. and Royer, E. M.; "Ad-hoc On-Demand Distance Vector Routing", Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99), New Orleans, LA, USA, February 1999, pages 90-100.

[6] Royer, E. M. and Perkins, C. E.; "Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol", Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99), Seattle, WA, USA, August 1999, pages 207-218.

[7] Royer, E. M. and Perkins, C. E.; "Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing", IETF, Intemet Draft: draft- ietf-manet-maodv-00.txt, 2000.

[8] Zhu, Y.; "Pro-active Connection Maintenance in AODV and MAODV", M. Sc. Thesis, Department of Systems and Computer Engineering, Carleton University, August 2002.