A Tutorial on the Implementation of Ad-hoc On Demand Distance Vector (AODV) Protocol in Network Simulator (NS-2)

Mubashir Husain Rehmani,
Sidney Doria, and Mustapha Reda Senouci *

Version 1

28th June 2009

*M. H. Rehmani is with INRIA, France, e-mail: mubashir.rehmani@inria.fr; S. Doria is with UFCG, Brazil, e-mail: sidney@dsc.ufcg.edu.br; M. R. Senouci is with Laboratory of Research in Artificial Intelligence, Algeria, email: mrsenouci@gmail.com; Special thanks to Hajer Ferjani, she has a Masters Degree in Networking from the National School of Computer Science of Tunisia, CRISTAL Laboratory in 2006, Tunisia, e-mail: f.hajer@gmail.com. The author would like to thanks Aline Carneiro Viana, who is with INRIA, France, e-mail: aline.viana@inria.fr;

Contents

Abstract

The Network Simulator (NS-2) is a most widely used network simulator. It has the capabilities to simulate a range of networks including wired and wireless networks. In this tutorial, we present the implementation of Ad Hoc On-Demand Distance Vector (AODV) Protocol in NS-2. This tutorial is targeted to the novice user who wants to understand the implementation of AODV Protocol in NS-2.

1 Introduction

The Network Simulator (NS-2) [1] is a most widely used network simulator. This tutorial presents the implementation of Ad Hoc On-Demand Distance Vector (AODV) Protocol [2] in NS-2. The expected audience are students who want to understand the code of AODV and researchers who want to extend the AODV protocol or create new routing protocols in NS-2. The version considered is NS-2.32 and 2.33, but it might be useful to other versions as well. Throughout the rest of this tutorial, the under considered files are aodv.cc, aodv.h, aodv_logs.cc, aodv_packet.h, aodv_rqueue.cc, aodv_rqueue.h, aodv_rtable.cc, aodv_rtable.h which can be found in AODV folder in the NS-2 base directory.

2 File Dependency of AODV Protocol

Fig. 1 and 2 shows the file dependency of AODV Protocol [3]. As AODV is a routing protocol, so it is derived from the class *Agent*, see agent.h.

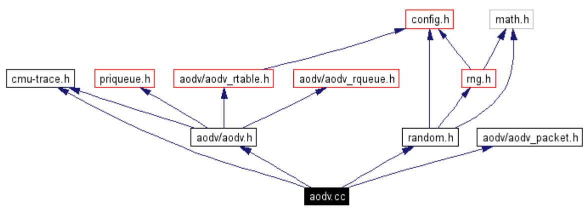

Figure 1: File Reference of 'AODV.CC'.

3 Flow of AODV

In this section, we describes the general flow of AODV protocol through a simple example:

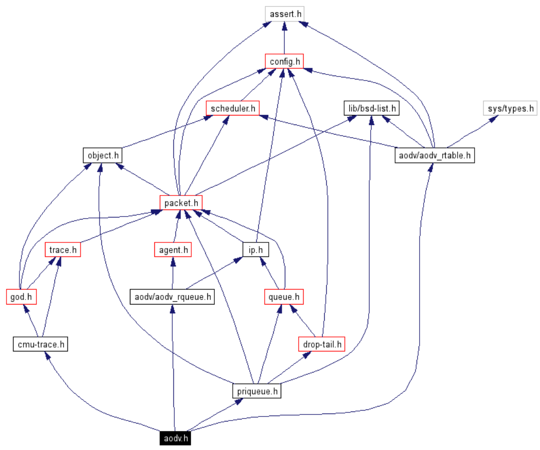

Figure 2: File Reference of 'AODV.H'.

1. In the TCL script, when the user configures AODV as a routing protocol by using the command,
   $ns node-config -adhocRouting AODV

   the pointer moves to the "start" and this "start" moves the pointer to the Command function of AODV protocol.

2. In the Command function, the user can find two timers in the "start"
   * btimer.handle((Event*) 0);
   * htimer.handle((Event*) 0);

3. Let's consider the case of htimer, the flow points to HelloTimer::handle(Event*) function and the user can see the following lines:

   agent -> sendHello();
   double interval = MinHelloInterval + ((MaxHelloInterval - MinHelloInterval) * Random::uniform());
   assert(interval -> = 0);
   Scheduler::instance().schedule(this, &intr, interval);

   These lines are calling the sendHello() function by setting the appropriate interval of Hello Packets.

4. Now, the pointer is in AODV::sendHello() function and the user can see Scheduler::instance().schedule(target_, p, 0.0) which will schedule the packets.

5. In the destination node AODV::recv(Packet*p, Handler*) is called, but actually this is done after the node is receiving a packet.

6. AODV::recv(Packet*p, Handler*) function then calls the recvAODV(p) function.

7. Hence, the flow goes to the AODV::recvAODV(Packet *p) function, which will check different packets types and call the respective function.

8. In this example, flow can go to
   case AODVTYPE_HELLO:
   recvHello(p);
   break;

9. Finally, in the recvHello() function, the packet is received.

4 Trace Format of AODV

In NS-2, the general trace format is given as below:

s 0.000000000 _0_ RTR — 0 AODV 44 [0 0 0 0] ——- [0:255 -1:255 1 0] [0x1 1 [0 2] 4.000000] (HELLO)

s 10.000000000 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)

s 21.500000000 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 4 [1 0] [0 12]] (REQUEST)

r 21.501260809 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 4 [1 0] [0 12]] (REQUEST)

The interpretation of the following trace format is as follows:

r 21.501260809 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 4 [1 0] [0 12]] (REQUEST)

Node ID 2, receives a packet type REQUEST (AODV protocol), at layer RTR (routing), at time 21.501260809. This packet have sequence number 0.

A generalized explanation of trace format [4] would be as follows :

Column Number	What Happened?	Values for instance...
1	It shows the occured event	's' SEND, 'r' RECEIVED, 'D' DROPPED
2	Time at which the event occured?	10.000000000
3	Node at which the event occured?	Node id like 0
4	Layer at which the event occured?	'AGT' application layer, 'RTR' routing layer, 'LL' link layer, 'IFQ' Interface queue, 'MAC' mac layer, 'PHY' physical layer
5	show flags	—
6	shows the sequence number of packets	0
7	shows the packet type	'cbr' CBR packet, 'DSR' DSR packet, 'RTS' RTS packet generated by MAC layer, 'ARP' link layer ARP packet
8	shows size of the packet	Packet size increases when a packet moves from an upper layer to a lower layer and decreases when a packet moves from a lower layer to an upper layer
9	[....]	It shows information about packet duration, mac address of destination, the mac address of source, and the mac type of the packet body.
10	show flags	—
11	[....]	It shows information about source node ip : port number, destination node ip (-1 means broadcast) : port number, ip header ttl, and ip of next hop (0 means node 0 or broadcast).

5 Main Implementation Files aodv.cc and aodv.h

5.1 How to Enable Hello Packets

By default HELLO packets are disabled in the aodv protocol. To enable broadcasting of Hello packets, comment the following two lines present in aodv.cc
#ifndef AODV_LINK_LAYER_DETECTION
#endif LINK LAYER DETECTION and recompile ns2 by using the following commands on the terminal:
make clean
make
sudo make install

5.2 Timers Used

In ns2, timers are used to delay actions or can also be used for the repetition of a particular action like broadcasting of Hello packets after fixed time interval. Following are the timers that are used in AODV protocol implementation:

- Broadcast Timer: This timer is responsible for purging the ID's of Nodes and schedule after every BCAST_ID_SAVE.
- Hello Timer: It is responsible for sending of Hello Packets with a delay value equal to interval, where
  double interval = MinHelloInterval + ((MaxHelloInterval - MinHelloInterval) * Random::uniform());
- Neighbor Timer: Purges all timed-out neighbor entries and schedule after every HELLO_INTERVAL .
- RouteCache Timer: This timer is responsible for purging the route from the routing table and schedule after every FREQUENCY.
- Local Repair Timer: This timer is responsible for repairing the routes.

5.3 Functions

5.3.1 General Functions

- void recv(Packet *p, Handler *): At the network layer, the Packet is first received at the recv() function, sended by the MAC layer in up direction. The recv() function will check the packet type. If the packet type is AODV type, it will decrease the TTL and call the recvAODV() function.
  If the node itself generating the packet then add the IP header to handle broadcasting, otherwise check the routing loop, if routing loop is present then drop the packet, otherwise forward the packet.
- int command(int, const char *const *): Every object created in NS-2 establishes an instance procedure, cmd{} as a hook to executing methods through the compiled shadow object. This procedure cmd invokes the method command() of the shadow object automatically, passes the arguments to cmd{} as an argument vector to the command() method [5].

5.3.2 Functions for Routing Table Management

- void rt_resolve(Packet *p): This function first set the transmit failure callback and then forward the packet if the route is up else check if I am the source of the packet and then do a Route Request, else if the local repair is in progress then buffer the packet.

  If this function founds that it has to forward a packet for someone else to which it does not have a route then drop the packet and send error upstream. Now after this, the route errors are broadcasted to the upstream neighbors.

- void rt_update(aodv_rt_entry *rt, u_int32_t seqnum,u_int16_t metric, nsaddr_t nexthop,double expire_time): This function is responsible for updating the route.

- void rt_down(aodv_rt_entry *rt): This function first confirms that the route should not be down more than once and after that down the route.

- void local_rt_repair(aodv_rt_entry *rt, Packet *p): This function first buffer the packet and mark the route as under repair and send a RREQ packet by calling the sendRequest() function.

- void rt_ll_failed(Packet *p): Basically this function is invoked whenever the link layer reports a route failure. This function drops the packet if link layer is not detected. Otherwise, if link layer is detected, drop the non-data packets and broadcast packets. If this function founds that the broken link is closer to the destination than source then It will try to attempt a local repair, else brings down the route.

- void handle_link_failure(nsaddr_t id): This function is responsible for handling the link failure. It first checks the DestCount, if It is equal to 0 then remove the lost neighbor. Otherwise, if DestCount > 0 then send the error by calling sendError() function, else frees the packet up.

- void rt_purge(void): This function is responsible for purging the routing table entries from the routing table. For each route, this function will check whether the route has expired or not. If It founds that the valid route is expired, It will purge all the packets from send buffer and invalidate the route, by dropping the packets and tracing DROP_RTR_NO_ROUTE "NRTE" in the trace file. If It founds that the valid route is not expired and there are packets in the sendbuffer waiting, It will forward them. Finally, if It founds that the route is down and if there is a packet for this destination waiting in the sendbuffer, It will call sendRequest() function.

- void enque(aodv_rt_entry *rt, Packet *p): Use to enqueue the packet.
- Packet* deque(aodv_rt_entry *rt): Use to dequeue the packet.

5.3.3 Functions for Neighbors Management

- void nb_insert(nsaddr_t id): This function is used to insert the neighbor.
- AODV_Neighbor* nb_lookup(nsaddr_t id): This function is used to lookup the neighbor.
- void nb_delete(nsaddr_t id): This function is used to delete the neighbor and It is called when a neighbor is no longer reachable.
- void nb_purge(void): This function purges all timed-out neighbor entries and It runs every
  HELLO_INTERVAL * 1.5 seconds.

5.3.4 Functions for Broadcast ID Management

- void id_insert(nsaddr_t id, u_int32_t bid): This function is used to insert the broadcast ID of the node.
- bool id_lookup(nsaddr_t id, u_int32_t bid): This function is used to lookup the broadcast ID.
- void id_purge(void): This function is used to purge the broadcast ID.

5.3.5 Functions for Packet Transmission Management

- void forward(aodv_rt_entry *rt, Packet *p, double delay): This function is used to forward the packets.
- void sendHello(void): This function is responsible for sending the Hello messages in a broadcast fashion.
- void sendRequest(nsaddr_t dst): This function is used to send Request messages.
- void sendReply(nsaddr_t ipdst, u_int32_t hop_count,nsaddr_t rpdst, u_int32_t rpseq,u_int32_t lifetime, double timestamp): This function is used to send Reply messages.

- void sendError(Packet *p, bool jitter = true): This function is used to send Error messages.

5.3.6 Functions for Packet Reception Management

- AODV::recvAODV(Packet *p): This function classify the incoming AODV packets. If the incoming packet is of type RREQ, RREP, RERR, HELLO, It will call recvRequest(p), recvReply(p), recvError(p), and recvHello(p) functions respectively.
- AODV::recvRequest(Packet *p): When a node receives a packet of type REQUEST, it calls this function.
- AODV::recvReply(Packet *p): When a node receives a packet of type REPLY, it calls this function.
- AODV::recvError(Packet *p): This function is called when a node receives an ERROR message.
- AODV::recvHello(Packet *p): This function receives the HELLO packets and look into the neighbor list, if the node is not present in the neighbor list, It inserts the neighbor, otherwise if the neighbor is present in the neighbor list, set its expiry time to: CURRENT_TIME + (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL), where ALLOWED_HELLO_LOSS = 3 packets and HELLO_INTERVAL = 1000 ms.

References

[1] [Online]. Available: http://www.isi.edu/nsnam/ns/

[2] C. E. Perkins and E. M. Royer, "The ad hoc on-demand distance vector protocol," *in Ad hoc Networking, Addison-Wesley*, pp. 173–219, 2000.

[3] [Online]. Available: http://www-rp.lip6.fr/ns-doc/ns226-doc/html/aodv_8cc-source.htm

[4] [Online]. Available: http://www.cs.binghamton.edu/~kliu/research/ns2code/

[5] K. Fall, "The ns manual (formerly ns notes and documentation)," 2008.

6 Appendex : A Simple TCL Script to Run the AODV Protocol

```
# wireless-aodv.tcl
    # A 3 nodes example for ad hoc simulation with AODV
    # Define options
    set val(chan)       Channel/WirelessChannel;# channel type
    set val(prop)       Propagation/TwoRayGround;# radio-propagation model
    set val(netif)      Phy/WirelessPhy ;# network interface type
    set val(mac)        Mac/802_11 ;# MAC type
    set val(ifq)        Queue/DropTail/PriQueue ;# interface queue type
    set val(ll)         LL ;# link layer type
    set val(ant)        Antenna/OmniAntenna ;# antenna model
    set val(ifqlen)     50 ;# max packet in ifq
    set val(nn)         3 ;# number of mobilenodes
    set val(rp)         AODV ;# routing protocol
    set val(x)          500 ;# X dimension of topography
    set val(y)          400 ;# Y dimension of topography
    set val(stop)       150 ;# time of simulation end
    set ns              [new Simulator]
    set tracefd         [open simple.tr w]
    set namtrace        [open simwrls.nam w]
    $ns trace-all $tracefd
    $ns namtrace-all-wireless $namtrace $val(x) $val(y)
    # set up topography object
    set topo [new Topography]
    $topo load_flatgrid $val(x) $val(y)
    create-god $val(nn)
    # Create nn mobilenodes [$val(nn)] and attach them to the channel.
    set chan_1_ [new $val(chan)]
    # configure the nodes
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-channel $chan_1_ \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
```

```
-movementTrace ON \

for {set i 0} {$i <$val(nn) } { incr i } {
set node_($i) [$ns node]
}

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0

# Generation of movements
$ns at 10.0 “$node_(0) setdest 250.0 250.0 3.0”
$ns at 15.0 “$node_(1) setdest 45.0 285.0 5.0”
$ns at 110.0 “$node_(0) setdest 480.0 300.0 5.0”

# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 “$ftp start”

# Define node initial position in nam
for {set i 0} {$i <$val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i <$val(nn) } { incr i } {
$ns at $val(stop) “$node_($i) reset”;
}

# ending nam and the simulation
$ns at $val(stop) “$ns nam-end-wireless $val(stop)”
```

```
$ns at $val(stop) "stop"
$ns at 150.01"puts "end simulation" ; $ns halt"
proc stop {} {
global ns tracefd namtrace
$ns flush-trace
close $tracefd
close $namtrace
}

    $ns run
```