NS2 WIRELESS NETWORK SIMULATION

A PROJECT REPORT

Submitted by

BL.EN.U4AIE19007 APOORVA.M

BL.EN.U4AIE19041 TANUJ.M

BL.EN.U4AIE19068 AISHWARYA.V

for the course

19AIE302- Advanced Computer Networks

Guided and Evaluated by

Mr.Rajesh M

IN

ARTIFICIAL INTELLIGENCE ENGINEERING

AMRITA SCHOOL OF ENGINEERING, BANGALORE

BANGALORE 560 035

PROBLEM STATEMENT

Question 2:

Implement the new routing protocol which modifies the existing AODV protocol.

The change required is in the packet format: An extra information: Packet number – a value assigned when a packet is send need to be included in the packet. Follow the steps mentioned in the document provided modifying AODV protocol.

To verify the working of modified protocol, simulate a wireless network consisting of 6 mobile nodes (n0 – n5). Use the modified routing protocol.

Note:

1. Develop the codes, test it.
2. Submit the code, a report consisting of the problem statement, details and results snapshots.
3. Prepare a PPT consisting of the details.

THEORY

AODV Routing Protocol in NS2 – Ad Hoc On-Demand Distance Vector is a routing protocol for ad hoc mobile networks with large numbers of mobile nodes. ADHOC creates routes between nodes only when the routes are requested by the source nodes. Ad Hoc Network gives the network flexibility to enter or leave the network at their own will.

The AODV protocol builds routes between nodes only if they are requested by source nodes. AODV is therefore considered an on-demand algorithm and does not create any extra traffic for communication along links. The routes are maintained as long as they are required by the sources. They also form trees to connect multicast group members. AODV makes use of sequence numbers to ensure route freshness. They are self-starting and loop-free besides scaling to numerous mobile nodes.

In AODV, networks are silent until connections are established. Network nodes that need connections broadcast a request for connection. The remaining AODV nodes forward the message and record the node that requested a connection. Thus, they create a series of temporary routes back to the requesting node. A node that receives such messages and holds a route to a desired node sends a backward message through temporary routes to the requesting node. The node that initiated the request uses the route containing the least number of hops through other nodes. The entries that are not used in routing tables are recycled after some time. If a link fails, the routing error is passed back to the transmitting node and the process is repeated.

The AODV convention creates connections between hubs only if source hubs mention them. As a result, AODV is considered an on-request calculation, and no additional traffic is created for correspondence along joins. The experts keep the courses up to date as required. They also build tree structures to link multicast bunch members. To ensure course freshness, AODV employs grouping numbers. Apart from scaling to various portable hubs, they are self-starting and circle open. Networks in AODV are silent until associations are formed. Organization hubs that use associations send out a request for membership. The extra AODV hubs are directed to the information and document hub that listed a link. They create a sequence of impermanent courses that lead back to the mentioning hub in this way.

SCRIPT AND RESULTS:

Added changes in (aodv.h) inbuilt code:

aodv.h

```
198   */
199
200         friend class aodv_rt_entry;
201         friend class BroadcastTimer;
202         friend class HelloTimer;
203         friend class NeighborTimer;
204         friend class RouteCacheTimer;
205         friend class LocalRepairTimer;
206
207  public:
208         AODV(nsaddr_t id);
209
210         void            recv(Packet *p, Handler *);
211
212  protected:
213         int             command(int, const char *const *);
214         int             initialized() { return 1 && target_; }
215
216         int pkt_count; /*ADDED CODE */
217
218
219         void            rt_resolve(Packet *p);
220         void            rt_update(aodv_rt_entry *rt, u_int32_t seqnum,
221                               u_int16_t metric, nsaddr_t nexthop,
222                               double expire_time);
223         void            rt_down(aodv_rt_entry *rt);
224         void            local_rt_repair(aodv_rt_entry *rt, Packet *p);
225  public:
226         void            rt_ll_failed(Packet *p);
227         void            handle_link_failure(nsaddr_t id);
228  protected:
229         void            rt_purge(void);
230
231         void            enque(aodv_rt_entry *rt, Packet *p);
232         Packet*         deque(aodv_rt_entry *rt);
233
```

Added changes in (aodv.cc) inbuilt code:

aodv.cc

```
134 /*
135    Constructor
136 */
137
138 AODV::AODV(nsaddr_t id) : Agent(PT_AODV),
139                           btimer(this), htimer(this), ntimer(this),
140                           rtimer(this), lrtimer(this), rqueue() {
141
142   pkt_count=0; /*ADDED CODE */
143
144   index = id;
145   seqno = 2;
146   bid = 1;
147
148   LIST_INIT(&nbhead);
149   LIST_INIT(&bihead);
150
151   logtarget = 0;
152   ifqueue = 0;
153 }
154
155 /*
156   Timers
157 */
```

aodv.cc

```
 998    Packet Transmission Routines
 999 */
1000
1001 void
1002 AODV::forward(aodv_rt_entry *rt, Packet *p, double delay) {
1003 struct hdr_cmn *ch = HDR_CMN(p);
1004 struct hdr_ip *ih = HDR_IP(p);
1005
1006 printf("Packet number %d and from node number %d\n",pkt_count++,index); /*ADDED CODE */
1007
1008  if(ih->ttl_ == 0) {
1009
1010 #ifdef DEBUG
1011   fprintf(stderr, "%s: calling drop()\n", __PRETTY_FUNCTION__);
1012 #endif // DEBUG
1013
1014   drop(p, DROP_RTR_TTL);
1015   return;
1016  }
1017
1018  if ((( ch->ptype() != PT_AODV && ch->direction() == hdr_cmn::UP ) &&
1019        ((u_int32_t)ih->daddr() == IP_BROADCAST))
1020               || (ih->daddr() == here_.addr_)) {
1021        dmux_->recv(p,0);
1022        return;
```

TCL SCRIPT:

```
# 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) 6 ;# number of mobilenodes
set val(rp) AODV ;# routing protocol
set val(x) 400 ;# 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 project.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)
```

```
# configure the nodes
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON
for {set i 0} {$i < $val(nn) } { incr i } {
set n($i) [$ns node]
}
# Provide initial location of mobilenodes
$n(0) set X_ 40.0
$n(0) set Y_ 280.0
$n(0) set Z_ 0.0

$n(1) set X_ 80.0
$n(1) set Y_ 340.0
$n(1) set Z_ 0.0

$n(2) set X_ 90.0
$n(2) set Y_ 230.0
```

```
$n(2) set Z_ 0.0

$n(3) set X_ 170.0
$n(3) set Y_ 340.0
$n(3) set Z_ 0.0

$n(4) set X_ 200.0
$n(4) set Y_ 200.0
$n(4) set Z_ 0.0

$n(5) set X_ 260.0
$n(5) set Y_ 350.0
$n(5) set Z_ 0.0

# Set a TCP connection between n(1) and n(31)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
$tcp set packetSize_ 1000
set sink [new Agent/TCPSink]
$ns attach-agent $n(2) $tcp
$ns attach-agent $n(3) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"
$ns at 125.0 "$ftp stop"

set tcp2 [new Agent/TCP/Newreno]
$tcp2 set class_ 2
```

```
$tcp2 set packetSize_ 600
set sink2 [new Agent/TCPSink]
$ns attach-agent $n(1) $tcp2
$ns attach-agent $n(4) $sink2
$ns connect $tcp2 $sink2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ns at 50.0 "$ftp2 start"

#defining heads
$ns at 0.0 "$n(0) label CH"
$ns at 0.0 "$n(1) label Source2"
$ns at 0.0 "$n(2) label Source1"
$ns at 0.0 "$n(4) label Destination2"
$ns at 0.0 "$n(3) label Destination1"

$ns at 100.0 "$n(5) setdest 385.0 228.0 5.0"
$ns at 60.0 "$n(2) setdest 200.0 20.0 5.0"
$ns at 30.0 "$n(3) setdest 115.0 85.0 5.0"
$ns at 45.0 "$n(1) setdest 375.0 80.0 5.0"
$ns at 89.0 "$n(4) setdest 167.0 351.0 5.0"
$ns at 78.0 "$n(0) setdest 50.0 359.0 5.0"

#Color change while moving from one group to another
$ns at 73.0 "$n(2) delete-mark N2"
$ns at 73.0 "$n(2) add-mark N2 pink circle"
$ns at 124.0 "$n(3) delete-mark N11"
$ns at 124.0 "$n(3) add-mark N11 purple circle"
$ns at 87.0 "$n(4) delete-mark N26"
```

```
$ns at 87.0 "$n(4) add-mark N26 yellow circle"
$ns at 92.0 "$n(1) delete-mark N14"
$ns at 92.0 "$n(1) add-mark N14 green circle"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 20 defines the node size for nam
$ns initial_node_pos $n($i) 20
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$n($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
exec nam simwrls.nam &
}

$ns run
```

Screenshots of NAM Output:

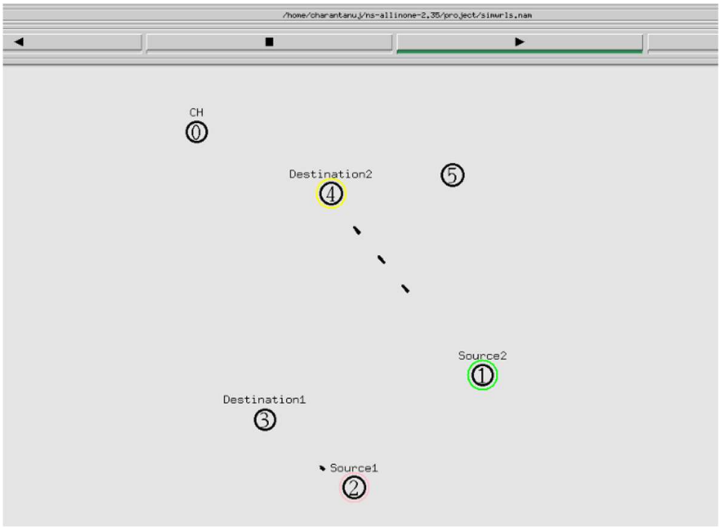

Fig 1

Fig 2

Screenshots of TERMINAL Output:

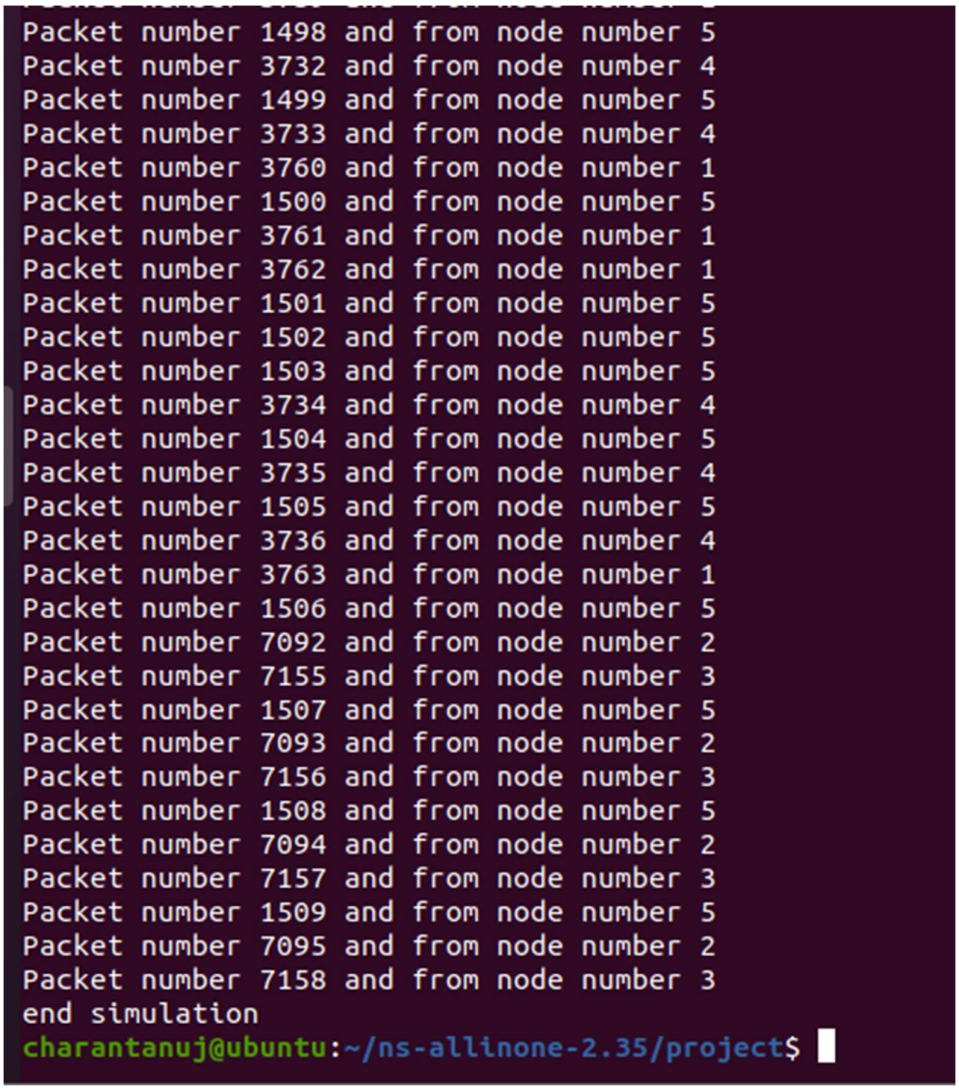

Fig 3