TCP over Ad Hoc Networks : NS-2 Simulation Analysis

Ren Mao, Haobing Wang, Li Li, Fei Ye

Chapter 1

Introduction

This report is for simulation of TCP transimition in Ad-hoc networks.To begin with,we have done some research on backgroud and basic knowledge for this lab .

1.1 Protocols in Ad-hoc Networks

Unlike the cellular networks where base stations are essential, ad-hoc networks is backed up by communications directly between mobiles, thus the routing protocols are central and deserve our focus on their mechanisms. And in ad-hoc networks, there exists several routing protocols as listed below, which will be demonstrated in this report:

1. DSDV :Destination Sequenced Distance Vector
2. AODV :Ad-hoc On Demand Distance Vector
3. TORA :Temporally Ordered Routing Algorithm
4. DSR :Dynamic Source Routing

Next, the details about these routing protocols will be shown.

1.1.1 DSDV : Destination Sequenced Distance Vector

DSDV is a distant vector routing protocol. Each node has a routing table that indicates for each destination, which is the next hop and number of hops to the destination. Each node periodically broadcasts routing updates. A sequence number is used to tag each route. It shows the freshness of the route: a route with higher sequence number is more favorable. In addition, among two routes with the same sequence number, the one with fewer hops is more favorable. If a node detects that a route to a destination has broken, then its hop number is set to infinity and its sequence number updated but assigned an odd number: even numbers correspond to sequence numbers of connected paths.

1.1.2 AODV : Ad-hoc On Demand Distance Vector

AODV is a distance vector type routing. It does not require nodes to maintain routes to destinations that are not actively used. As long as the endpoints of a communication connection have valid routes to each other, AODV does not play a role.
The protocol uses different messages to discover and maintain links: Route Requests(RREQs), Route Replies(RREPs), and Route Errors(RERRs). These message types are received via UDP, and normal IP header processing applies.
AODV uses a destination sequence number for each route entry. The destination sequence number is created by the destination for any information it sends to request nodes. Using destination sequence numbers ensures loop freedom and allows to know which of several routes is more fresh. Given the choice between two routes

to a destination, a requesting node always selects the one with the greatest sequence number.
When a node wants to find a route to another one, it broadcasts a RREQ to all the network till either the destination is reached or another node is found with a fresh enough route to the destination (a fresh enough route is a valid route entry for destination whose associated sequence number is at least as great as that contained in the RREQs). Then a RREQ is sent back to the source and the discovered route is made available.
Nodes that are part of an active route may offer connectivity information by broadcasting periodically local Hello messages (special RREQ messages) to its immediate neighbors. If Hello messages stop arriving from a neighbor beyond some given time threshold, the connection is assumed to be lost.
When a node detects that a route to a neighbor node is not valid it removes the routing entry and send a REER message to neighbors that are active and use the route; this is possible by maintaining active neighbor lists. This procedure is repeated at nodes that receive REER messages. A source that receives an REER can reinitiate a RREQ message.AODV does not allow to handle unidirectional links.

1.1.3 TORA : Temporally Ordered Routing Algorithm

This protocol is of the family of link reversal protocols. It may provide several routes between a source and a destination. TORA contains three parts: creating, maintaining, and erasing routes. At each node, a separate copy of TORA is run per each destination. TORA builds a directed acyclic graph rooted at the destination. It associates a height with each node in the network (with respect to a common destination). Messages flow from nodes with higher height to those with lower heights. Routes are discovered using Query (QRY) and Update (UPD) packets.
When a node with no downstream links need a route to a destination, it broadcasts a QRY packet that propagates till it either find s a node with a route to the destination or the destination itself. That node will respond by broadcasting a UPD packet containing the nodes height. A node receiving the UPD packet updates its height accordingly and broadcasts another UPD. This may results in a number of directed paths from the source to the destination.
If a node discovers a particular destination to be unreachable, it sets the corresponding local heights to a maximum value. In case the node cannot find any neighbor with finite height with respect to this destination, it attempts to find a new route. In case there is no route to a destination (i.e. of a network partition), the node broadcasts a Clear (CLR) message resetting all routing states and removing invalid routes from its part of the network.

1.1.4 DSR : Dynamic Source Routing

Designed for mobile ad-hoc networks with up to around two hundred nodes with possibly high mobility rate, the protocol works on demand, i.e. without any periodic updates.
Packets carry along the complete path they should take. This reduces overheads for large routing updates at the network. The nodes stored in their cache all know routes. The protocol is composed of route discovery and route maintenance.
At route discovery, a source requesting to send a package to a destination broadcasts a Route Request (RREQ) packet. Nodes receiving RREQ search in their Route Cache for a route to the destination. If a route is not found, the RREQ is further transmitted and the node adds its own address to the recorded hop sequence. This continues till the destination or a node with a route to the destination is reached. The route back can be retrieved by the reverse hop record. As routes need not be symmetric, DSR checks the Route Cache of the replying node and if a route is found, it is used instead. Alternatively, one can piggyback the reply on a RREQ targeted at the originator. Hence unidirectional links can be handled.
And at route maintenance, when originating or forwarding a packet using a source route, each node transmitting the packet is responsible for confirming that data can flow over the link from that node to the next hop. An acknowledgment can provide confirmation that a link is capable of carrying data. Acknowledgments are often already part of the MAC protocol in use (such as the link-layer acknowledgment frame defined by IEEE 802.11), or are passive acknowledgment, i.e. a node knows that its packet is received by an intermediate node since it can hear that the intermediate node further forwards it. If such acknowledgments are not available then a node can request an acknowledgment (which can be sent directly to the source using another route).

Acknowledgments may be requested several times till some given bound, and in the persistent absence of acknowledgment, the route is removed from the Route Cache and return a Route Error to each node that has sent a packet routed over that link since acknowledgment was last received. Nodes overhearing or forwarding packets should make use all carried routing information to update its own Route Packet.

1.2 What we are concerned about

In this report, we will simulate an ad-hoc network using different routing protocols with the help of NS and then make a comparison based on the result.

1.2.1 Simulation scenario

According to the lab notes, there is a 3-nodes network over an area of a size of 500m over 400m depicted in Fig.1.1

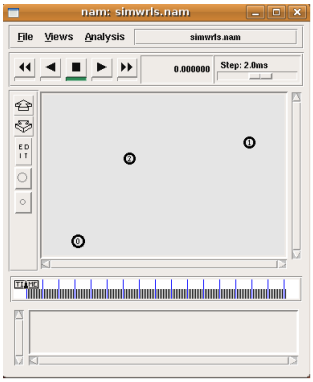

Figure 1.1: Basic topology of 3-nodes network

The initial location of nodes 0,1,and 2 are respectively (5,5), (490,285) and (150,240) (the z coordinate is assumed throughout to be 0).
At time 10, node 0 starts moving towards point (250,250) at a speed of 3m/sec. At time 15, node 1 starts moving towards point (45, 258) at a speed of 5m/sec. At time 110, node 0 starts moving towards point (480,300) at a speed of 5m/sec.
The simulation lasts 150 sec. At time 10, a TCP connection using the DSDV ad-hoc routing protocol and the IEEE802.11 MAC protocol is initiated between node 0 and node 1.
All the simulation is based on this basic simulation scenario,and changes on the scenario will be described in later chapters.

1.2.2 Tips on evaluate the different protocols

we choose two factors to judge the advantages of each protocol:

- *Window Size*: Window size shows the size of buffer in the receivers, so it will influence the communication procedure. So, we write a function to print out a figure to reflect the change of window size during transmitting.
- *Total number of TCP packets*: We can obtain this information by using the command "grep" to get the trace information file we need and then counting the number of lines. Or we can be more precise and look at the sequence number of the last received TCP packet.More specified usage will be showed in results analysis.

with these two methods ,we can do our simulation and compare the results of simulation for different protocols.

Chapter 2

Basic DSDV Simulation

2.1 Simulation scenario modification

During our process of doing the task, we found that in the scenario given in the note, we can not get the outcome like what the note showed us. So there must be some mistakes with the parameters of the note. To get the similar result shown in the note, we change the parameters of scenario. The result turns out to be perfect now.
The mistaken scenario given in the note is as follow:
At time 15, node 1 starts moving towards point (45, 285) at a speed of 5m/sec
At time 10, node 0 starts moving towards point (480,300) at a speed of 5m/sec.
We change it as follow:
At time 110, node 0 starts moving towards point (480,300) at a speed of 5m/sec.
At time 15, node 1 starts moving towards point (45, 258) at a speed of 5m/sec

2.2 NS2 coding

2.2.1 Definition of TCP protocol

To start with, we define a set of global variables to describe a TCP protocol in NS2.

```
set val(chan)           Channel/WirelessChannel;
set val(prop)           Propagation/TwoRayGround;
set val(netif)          Phy/WirelessPhy ;
set val(mac)            Mac/802_11 ;
set val(ifq)            Queue/DropTail/PriQueue;
set val(ll)             LL ;
set val(ant)            Antenna/OmniAntenna  ;
set val(ifqlen)         50 ;
set val(nn)             3  ;
set val(rp)             DSDV;
set val(x)              500;
set val(y)              400;
set val(stop)           150;
```

In this TCP protocol, The channel type is set to be wireless channel. The radio-propagation is set to be TwoRayGround model. The network interface type is set to be Wireless. The MAC type is set to be fit in IEEE 802_ 11 MAC protocol. The interface queue type is set to be priqueue with droptail. The antenna model is set to be OmniAntenna The max packet in interface queue is set to be 50. The number of mobile nodes is set to be 3. The routing protocol is set to be DSDV . The X dimension of topography is set to be 500. The Y dimension of topography is set to be 400. The time of simulation end is set to be 150.

2.2.2 Creation of necessary files

In this section, we create some necessary files for the simulation.

```
set ns           [new Simulator]
set tracefd       [open simple.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open simwrls.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)
```

simple.tr is used to keep track of action of the network; win.tr is an output figure reflecting the relationship between window size of TCP and time(set time as x-axis and window size as y-axis). As the other parts of the code above are discussed in Lab1, so we do not discuss them repeatedly here.

2.2.3 Create the topology

The topology represents a communication system, we can easily create the to topology as in Lab1.

```
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
```

Then,we use the following command to set parameters in configuring of the nodes fit in the TCP protocol which is referred above

```
$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
```

We need to create the initial node position for nam using

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

Then, we Provide initial location of mobile nodes:

```
$ 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
```

Three nodes' movement will be written as follow. A linear movement of a node is generated by specifying the time in which it starts, the x and y values of the target point and the speed.

```
$ns at 10.0  "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0  "$node_(1) setdest 45.0 258.0 5.0"
$ns at 100.0 "$node_(0) setdest 480.0 300.0 5.0"
```

Then, we define the node size for nam to be 30

```
for { set i 0}  { $i <$val(nn)}  { incr i }  {
    $ns initial_node_pos $node_($i) 30
}
```

2.2.4 Create connections among nodes

We create a TCP connection between node0(sender) and node1(receiver).

```
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"
```

Window size of TCP represents the size of receiver's buffer, which will affect the transmission process. So we define a function to print the window size of TCP. We set time as x-axis and window size as y-axis: Printing the window size:

```
proc plotWindow { tcpSource file}  {
          global ns
          set time 0.01
          set now [$ns now]
          set cwnd [$tcpSource set cwnd_ ]
          puts $file "$now $cwnd"
          $ns at [expr $now+$time] "plotWindow $tcpSource $file"
}

$ns at 10.1 "plotWindow $tcp $windowVsTime2"
```

2.2.5 End of the script

We tell each node that the program ends at 150:

```
   for { set i 0}  { $i < $val(nn) }  {  incr i }  {
        $ns at $val(stop) "$node_($ i) reset";
   }
```

Then we invoke the stop function and start nam:

```
$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 &
        exit 0
}
```

2.3 Simulation results

2.3.1 Trace Format

Run the script, and we can get the file sipmgle.tr. As mentioned before, simple.tr is used to keep track of action of the network. Now Let us see what the output trace is. We take some examples arbitrarily:

s 55.032641812 _ 1_ AGT — 26 ack 40 [0 0 0 0] ——- [1:0 0:0 32 0] [1 0] 0 0
r 55.032641812 _ 1_ RTR — 26 ack 40 [0 0 0 0] ——- [1:0 0:0 32 0] [1 0] 0 0
s 55.032641812 _ 1_ RTR — 26 ack 60 [0 0 0 0] ——- [1:0 0:0 32 2] [1 0] 0 0
r 55.034847281 _ 2_ RTR — 26 ack 60 [13a 2 1 800] ——- [1:0 0:0 32 2] [1 0] 1 0
f 55.034847281 _ 2_ RTR — 26 ack 60 [13a 2 1 800] ——- [1:0 0:0 31 0] [1 0] 1 0
r 55.036972761 _ 0_ AGT — 26 ack 60 [13a 0 2 800] ——- [1:0 0:0 31 0] [1 0] 2 0

The first field is a letter that can have the values r, s, f, D for "received", "sent", "forwarded" and "dropped", respectively.
The second field is the time.
The third field is the node number.
The fourth field is MAC to indicate if the packet concerns a MAC layer, it is AGT to indicate a the transport layer packet, or RTR if it concerns the routed packet. It can also be IFQ to indicate events related to the interference priority queue.
After the dashes come the global sequence number of the packet.
At the next field comes more information on the packet type.
Then comes the packet size in bytes.
The 4 numbers in the first square brackets concern mac layer information.
The first hexadecimal number, specifies the expected time in seconds to send this data packet over the wireless channel.
The second number, stands for the MAC-id of the sending node.
The third, is that of the receiving node.
The fourth number, 800, specifies that the MAC type
The next numbers in the second square brackets concern the IP source and destination addresses, then the ttl (Time to Live) of the packet.

So the procedure can be expressed as:
It is first sent by the TCP agent at node 1,then received by the routing protocol of the same node and sent from there with an additional header. It is then received and forward by node 2. Then it is finally received by node 0.

Let see another form:

M 15.00000 1 (490.00, 285.00, 0.00), (45.00, 258.00), 5.00

M stands for giving a location or a movement indication. The first number is the time, the second is the node number, then comes the origin and destination locations, and finally is given the speed.
So this means node 1 begin move from (490.00, 285.00, 0.00) at 15.00 to (45.00, 258.00) with the speed of 5.00.

2.3.2 Windowsize Figure

Then we run the following command to plot the graph Fig.2.2:

```
$xgraph win.tr
```

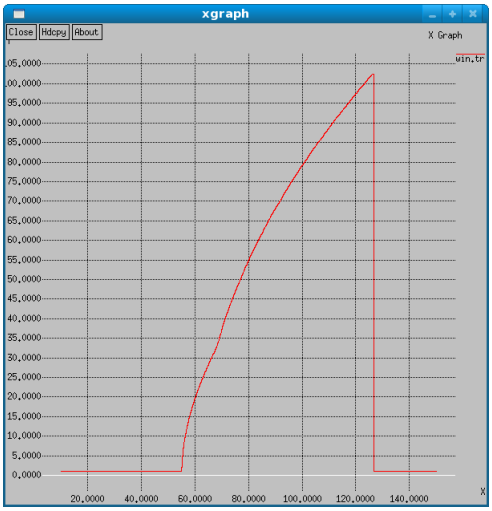

Figure 2.1: Windows Evolution of DSDV before Changes

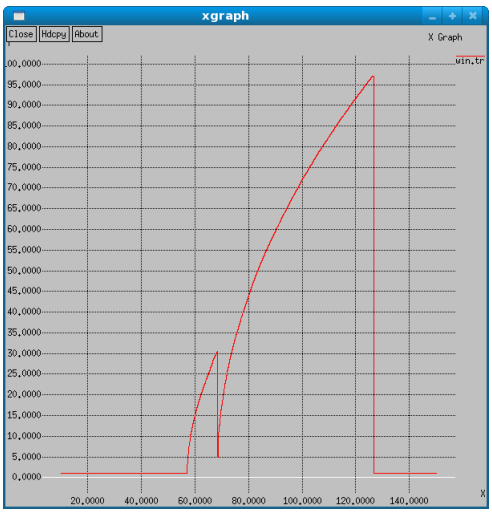

Figure 2.2: Windows Evolution of DSDV after Changes

Actually,the graph is not the same as the figure showed in the notes since we have changed several parameters of the scenario. Nevertheless,the tendancy of the curve is the same,and it also shows the whole transimition:
At the beginning the nodes are too far away and a connection cannot be set. The first TCP signaling packet is transmitted at time 10 sec but the connection cannot be opened. Meanwhile nodes 0 and nodes 1 start moving towards node2. After 6 second (timeout) a second reatempt occurs but still the connection cannot be established and the timeout valure is doubled to 12sec. At time 28 another transmission attempt occurs.While the connection still could not be established.Then at around 55 sec, both nodes 0 as well as node 1 to be within the radio of node 2 so that when tcp connection is reattempted at that time a two hop path is established between node 0 a direct connection is established.At the moment of the path change there is

a single TCP packet loss that cause the window to decrease slightly.At time 125.5 nodes 0 and 1 are too far apart for the connection to be maintained and the conncetion breaks.

Run the command and counting the number of lines of the file DSDV.tr:

```
grep "^r" simple.tr | grep "tcp"  |  grep "_1_  AGT" > DSDV.tr
```

We can get the total number of TCP packets transfered using DSDV is 5263.

Next we slightly change the parameters of the simulation,we can see more obvious loss of TCP packets. The only change is in fact that the ftp transfer will start now at time 12 instead of at time 10. This will cause the loss of TCP packets much larger at the moment of the path change so that the window decrease heavily.The window size graph is Fig2.3.

Chapter 3

Extend Simulation

In this chapter, we have done several extension of this lab,which means that we have edit several options to simulate this TCP transmition.

3.1 TCP over AODV

3.1.1 Simulation Procedure

This simulation is almost the same as the basic simulation over DSDV above.Nearly all of the parameters as above is repeated.We only change the routing protocol parameter as follow:

```
set val(rp) AODV ; # routing protocol
```

Run this tcl script,we get the window size Figure3.1as below:

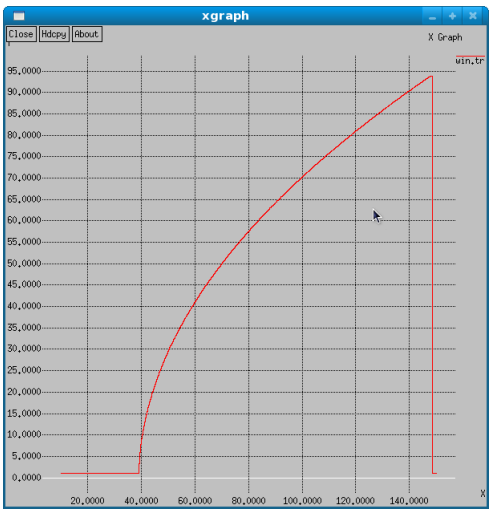

Figure 3.1: Window size evolution over AODV with the basic 3 nodes scenario

Figure 3.2: Window size evolution over AODV with the 2000 maximam window size

Actually, according to instruction pdf, we are interested in losses due to buffer overflow.Since we used the default value of the maximum window size of TCP of 20 while the actual window that is used is the minimum between the congestion window and 20,we change the maximum window size of 2000 by following command:

```
.....
$tcp set class_2
$tcp set window_ 2000 #this is for changing maximum window
set sink [new Agent/TCPSink]
......
```

Therefore,we can get the Figure3.2 as above:

3.1.2 Analysis of Results

To analyze this route protocol,we first analyze the Figure3.1.It had throughout a long single phase in which the same two hop path was used,in which node 2 relayed the packets.
To prove our anlysis,we run the nam and simulate the whole procedure.The Figure 3.3 is the nam graph at time 90. From the figure above, we can easily find that the connection has two hop path where node 2 is a

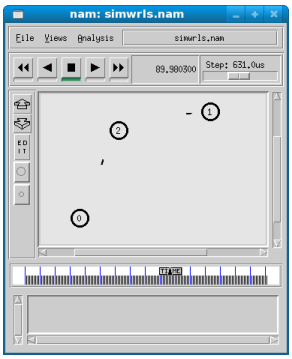

Figure 3.3: TCP over AODV at time 90

relay. By grep and count the number of tcp packets,we found that 4421 packets of TCP had been transferred during the whole connection.
Take the buffer overflow into account,we got the Figure3.2,which the maximam of window size is set into 2000.From that figure, we see that w obtain losses due to overflow.

3.2 TCP over DSR

3.2.1 Simulation Procedure

For DSR protocol, we first change the routing protocol by changing in tcl script the corresponding line to

```
set val(ifq) CMUPriQueue;
......
set val(rp) DSR
```

Change the Queue/DropTail/PriQueue to CMUPriQueue is mandatory to avoid runtime errors. Actually, for further research,we found if we don't change the queue type, it will result in a segmentation fault.During debugging,we found that the error has occurred in the funcion `CMUPriQueue::prq_get_nexthop(nsaddr_t id)` of Dsr-priqueue.cc.Specifically in the following part:

```
for(q = 0; q < IFQ_MAX; q++) {
   ifq = &prq_snd_[q];
   pprev = 0;
   for(p = ifq->ifq_head; p; p = p->next_) {
       struct hdr_cmn *ch = HDR_CMN(p);

       if(ch->next_hop() == id)
```

```
                break;
            pprev = p;
        }
        ....................
    }
```

In the code,we found all elements in `prq_snd_` has not been initialized rationally.With the help of Internet, we acknowledged that due to that `prq_snd_` is one member of CMUPriQueue,the DSR protocol has to use CMUPriQueue.Otherwise,the `prq_snd_` will not be initalized and results in error.
After changing the script,we perform the simulation,We observe the following Figure3.3 of window size More detailed analysis can be found in chapter 4.

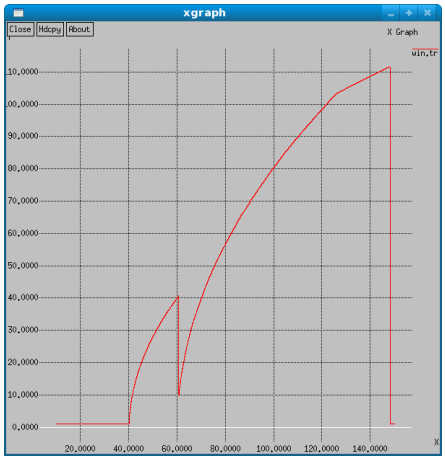

Figure 3.4: Window size evolution of the TCP connection for DSR

3.2.2 Analysis of Results

When performing the simulation, we observe five phases of operation ,as Figure 3.4 shows.
In the first and last,the nodes are too far away and there is no connectivity.During phase 2 and 4,connection between node 0 and 1 use node 2 as a relay, whereas in the 3rd phase ,there is a direct paht between node 0 and 1.

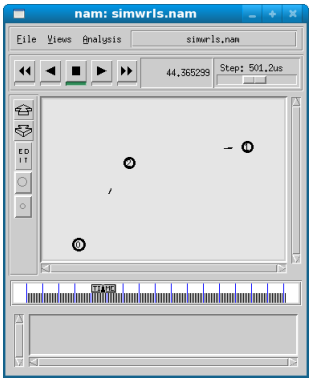

Figure 3.5: TCP over DSR ,time 44,a 2 hop path

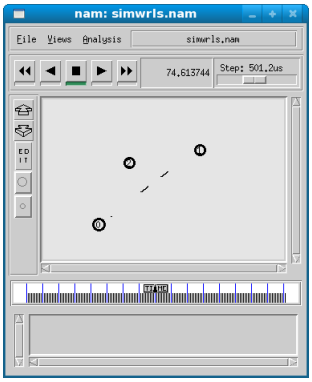

Figure 3.6: TCP over DSR ,time 74 ,a single hop path

To point out these conclusions,we get Figure 3.5 at time 44 and Figure 3.6 at time 74 as above:

Phase 2 starts at around time 40.Phase 3 starts at around 60 sec. At time 125.50 the fourth phase starts and at tim 149 sec it ends,which ends the whole connection.These all can be found in Figure 3.3.
We can note that in the DSDV, the system was not able to provide the 4th phase, so the connection was ended much earlier.
We also found that the total number of TCP packets transfered using DSR is 7042,which is larger than in DSDV,whereas in DSDV with the same parameters(corresponding to Figure 2.2) the number is 5263.

3.3 TCP over TORA

For TORA protocol,since the NS2 software has several bugs in tora.h and tora.cc file,if we directly simulate this tcl file,it shows that errors in call function of tora command.Therefore,before we do the simulation, we have edit these files in this way :
In tora.h: add the following line in private varables declariation:

```
NsObject *port_dmux_;
```

In tora.cc: add the following codes after the last "else" in function definition of toraAgent::command:

```
 else if (strcmp(argv[1], "port-dmux") == 0) {
     if((port_dmux_=(NsObject *) TclObject::lookup(argv[2]))==0) {
return TCL_ERROR;
}
```

Then we recompile the ns2 program. Next, change the tcl script into TORA router protocol and run .However,another fatal problem comes out that it seems the program falls into dead cycles and there is no response for simulation results.By now ,we still can not figure out the solution of this problem.So the TCP simulation over TORA can only postponed to further research.

Chapter 4

Answers for Questions

1. How many packets hav been transmitted by TCP using AODV as underlying protocol?
   Sol:
   By the command:

   ```
   grep "^r" simple.tr |grep "tcp"|grep "_1_ AGT" >AODV_tcp.tr
   ```

   and then we look at the sequence number of the last received tcp packet.We can find that the total number of TCP packets transfered using ADOV is 4421.

2. Try to plot the windows size evolution in DSR(TORA) vs DSDV, save and submit your figure.Compare their windows size evolution performance.Give your judgement.
   Sol:
   The figure of comparison between DSR and DSDV is :
   It is easy to find that in DSDV,the system was not able to provide the 4th phase ,so the connection

Figure 4.1: Comparison between DSR and DSDV

   was ended much earlier.While in DSR,the windows size evolution is better.

3. In the protocols we discussed here,which one have transmitted most packets? How to calculate the total packets transmitted or received by a node?
   Sol:
   Compare these total packets:
   5263 in DSDV,
   4421 in AODV,

7042 in DSR,
It is easy to know that in the DSR, it have transfered most packets.
To calculate the total packets recieved by a node n(n=0,1,2),just use the command:

```
grep "^r" simple.tr |grep "_n_ AGT" >n_tcp.tr
```

and then to count the number of lines.
Change `"^r"` into `"^s"` can calculate the total packets transmitted.

4. Among the DSDV,DSR and AODV, which protocol most suits highly mobility systems?Provide the reasons to justify your answer.
   Sol:
   DSR maybe most suits highly mobility systems.
   Because according to the simulation results above,the DSR can transmit most packets in scenairo (actually with most direction connections),which is best to highly mobility systems.

Chapter 5

Appendix

5.1 What we have learned

Thanks to Lab2, through this experience, we have learned much fundamental knowledge about the theory of ad-hoc networks, especially the principle of different routing protocols, which is really helpful for our further study in wireless networks.
According to the result, we can figure out that each kind of routing protocols has its own merits and flaws in some sense, so they should be applied in different conditions with respect to the area required to be covered and various mobility. This point of view renders it a significant job to have a careful investigation on those protocols, exactly what we have done in this experiment.
And through the progress, we acquire some understanding of basic performing principle of routing protocols. In general, the routing algorithm works in an interactive way. To be more specific, before sending out the data, the sender gives out a signal that tells others that Im going to send out data, and not until receiving a signal from someone which means that You can send the data now, the formation about the chosen route also is informed to the sender. And, it should contain some methods to detect route error and attempt to fix it.
And with the help of routing protocols, ad-hoc networks enable us to choose route in a flexible manner: each mobiles in this network can act as a transshipment station to help transmitting the data to the terminal. In this way, under the condition that the network can obtain a general realization of each node, it will be much more speedy and convenient to find a proper path for transferring.
Although it is far from saying that this experience will benefit in a long-term, we should admit that what we have gained in this lab will help us a lot in the further research about the field of ad-hoc networks. So, we should give a sincere thank to dear teacher and TA for providing us such a good chance to explore our potential. Thanks very much!

5.2 Codes

```
# 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)             DSDV                       ;# routing protocol
#set val(rp)             AODV                       ;
#set val(rp)             DSR                       ;
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 windowVsTime2 [open win.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 < $val(nn) } { incr i } {
set node_($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 45.0 258.0 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]
$tcp set class_ 2
#$tcp set window_ 2000
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"
#ns at 12.0 "$ftp start"

# Printing the window size
proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"

# 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
    exec nam simwrls.nam &
    exit 0
}

$ns run
```