# Define options

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) 2

#define the routing protocol

set val(rp) DSDV

#set val(rp) DSR

set val(x) 500

set val(y) 500

# Initialize Global Variables

set ns\_ [new Simulator]

set tracefd [open wireless\_mitf.tr w]

$ns\_ trace-all $tracefd

set namtracefd [open wireless\_mitf.nam w]

$ns\_ namtrace-all-wireless $namtracefd $val(x) $val(y)

#read trace file in a meaningful way

#$ns\_ use-newtrace

# set up topography object

set topo [new Topography]

#define topology

$topo load\_flatgrid $val(x) $val(y)

# Create God

create-god $val(nn)

# New API to config node:

# 1. Create channel (or multiple-channels);

# 2. Specify channel in node-config (instead of channelType);

# 3. Create nodes for simulations.

# Sequence:

# 1) create channel

# 2) configure node

# 3) create node

# 4) attach node to channel.

# create two channels as channel #1 and channel #2

set chan\_1\_ [new $val(chan)]

set chan\_2\_ [new $val(chan)]

# configure node

$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) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace ON \

-movementTrace OFF \

-channel $chan\_1\_

# node\_(0) and node\_(1) are created under chan\_1\_ and attatch # to chan\_1\_

set node\_(0) [$ns\_ node]

set node\_(1) [$ns\_ node]

# node\_(1) can also be created with the same configuration,or

# with a different channel specified by chan\_2\_

# Uncomment below two lines will create node\_(1) with a different channel.

# $ns\_ node-config \

# -channel $chan\_2\_

#set node\_(1) [$ns\_ node]

# node\_(0) and node\_(1) are created under chan\_2\_ and attatch # to chan\_2\_

#set node\_(0) [$ns\_ node]

#set node\_(1) [$ns\_ node]

#diable random motion of nodes

$node\_(0) random-motion 0

$node\_(1) random-motion 0

#define node initial positions in NAM

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

$ns\_ initial\_node\_pos $node\_($i) 20

}

# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes

$node\_(0) set X\_ 5.0

$node\_(0) set Y\_ 2.0

$node\_(0) set Z\_ 0.0

$node\_(1) set X\_ 20.0

$node\_(1) set Y\_ 8.0

$node\_(1) set Z\_ 0.0

# Now produce some simple node movements

# Node\_(1) starts to move towards node\_(0)

#

$ns\_ at 3.0 "$node\_(1) setdest 50.0 40.0 25.0"

$ns\_ at 3.0 "$node\_(0) setdest 48.0 38.0 5.0"

# Node\_(1) then starts to move away from node\_(0)

$ns\_ at 20.0 "$node\_(1) setdest 490.0 480.0 30.0"

# Setup traffic flow between nodes

# TCP connections between node\_(0) and node\_(1)

set tcp [new Agent/TCP]

$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

$ns\_ at 3.0 "$ftp start"

# Tell nodes when the simulation ends

#

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

$ns\_ at 30.0 "$node\_($i) reset";

}

$ns\_ at 30.0 "stop"

$ns\_ at 30.01 "puts \"NS EXITING...\" ; $ns\_ halt"

proc stop {} {

global ns\_ tracefd namtracefd

$ns\_ flush-trace

close $tracefd

close $namtracefd

exec nam wireless\_mitf.nam &

exit 0

}

puts "Starting Simulation..."

$ns\_ run

**AWK Script**

**Wireless Trace File Format \_ OLD**

**Find out the THROUGHPUT of the wireless network**

BEGIN {

recvdSize = 0

startTime = 400

stopTime = 0

}

{

event = $1

time = $2

node\_id = $3

level = $4

pkt\_size = $8

# Store start time

if (level == "AGT" && event == "s" && pkt\_size >= 512) {

if (time < startTime) {

startTime = time

}

}

# Update total received packets' size and store packets

# arrival time

if (level == "AGT" && event == "r" && pkt\_size >= 512) {

if (time > stopTime) {

stopTime = time

}

# Rip off the header

hdr\_size = pkt\_size % 512

pkt\_size = pkt\_size - hdr\_size

# Store received packet's size

recvdSize = recvdSize + pkt\_size

}

}

END {

printf("Average Throughput[kbps] = %.2f\t\t StartTime=%.2f\tStopTime=%.2f\n",(recvdSize/(stopTime-startTime))\*(8/1000),startTime,stopTime)

}

**To calculate the Send, Received, Dropped Packets, Received Packets, Packet Delivery Ratio and Average end to End Delay**

BEGIN {

seqno = -1;

droppedPackets = 0;

receivedPackets = 0;

count = 0;

}

{

#packet delivery ratio

if($4 == "AGT" && $1 == "s" && seqno < $6) {

seqno = $6;

} else if(($4 == "AGT") && ($1 == "r")) {

receivedPackets++;

} else if ($1 == "D" && $7 == "tcp" && $8 > 512){

droppedPackets++;

}

#end-to-end delay

if($4 == "AGT" && $1 == "s") {

start\_time[$6] = $2;

} else if(($7 == "tcp") && ($1 == "r")) {

end\_time[$6] = $2;

} else if($1 == "D" && $7 == "tcp") {

end\_time[$6] = -1;

}

}

END {

for(i=0; i<=seqno; i++) {

if(end\_time[i] > 0) {

delay[i] = end\_time[i] - start\_time[i];

count++;

}

else {

delay[i] = -1;

}

}

for(i=0; i<count; i++) {

if(delay[i] > 0) {

n\_to\_n\_delay = n\_to\_n\_delay + delay[i];

}

}

n\_to\_n\_delay = n\_to\_n\_delay/count;

print "\n";

print "GeneratedPackets = " seqno+1;

print "ReceivedPackets = " receivedPackets;

print "Packet Delivery Ratio = " receivedPackets/(seqno+1)\*100"%";

print "Total Dropped Packets = " droppedPackets;

print "Average End-to-End Delay = " n\_to\_n\_delay \* 1000 " ms";

print "\n";

}

**To print Packet Delivery Ratio**

BEGIN {

sendLine = 0;

recvLine = 0;

fowardLine = 0;

}

$0 ~/^s.\* AGT/ {

sendLine ++ ;

}

$0 ~/^r.\* AGT/ {

recvLine ++ ;

}

$0 ~/^f.\* RTR/ {

fowardLine ++ ;

}

END {

printf "cbr s:%d r:%d, r/s Ratio:%.4f, f:%d \n", sendLine, recvLine, (recvLine/sendLine),fowardLine;

}

**To Calculate Jitter**

BEGIN {

num\_recv = 0

printf("# %10s %10s %5s %5s %15s %16s %16s %16s %16s\n\n", \

"flow","flowType","src","dst","time","jitter1","jitter2","jitter3","jitter4")

}

{

# Trace line format: normal

if ($2 != "-t") {

event = $1

time = $2

if (event == "+" || event == "-") node\_id = $3

if (event == "r" || event == "d") node\_id = $4

flow\_id = $8

pkt\_id = $12

pkt\_size = $6

flow\_t = $5

level = "AGT"

}

# Trace line format: new

if ($2 == "-t") {

event = $1

time = $3

node\_id = $5

flow\_id = $39

pkt\_id = $41

pkt\_size = $37

flow\_t = $45

level = $19

}

# Store packets send time

if (level == "AGT" && flow\_id == flow && node\_id == src &&

sendTime[pkt\_id] == 0 && (event == "+" || event == "s") && pkt\_size >= pkt) {

sendTime[pkt\_id] = time

}

# Store packets arrival time

if (level == "AGT" && flow\_id == flow && node\_id == dst &&

event == "r" && pkt\_size >= pkt) {

recvTime[pkt\_id] = time

num\_recv ++

}

}

END {

# Compute average jitter

jitter1 = jitter2 = jitter3 = jitter4 = tmp\_recv = 0

prev\_time = delay = prev\_delay = processed = currTime = 0

prev\_delay = -1

for (i=0; processed<num\_recv; i++) {

if(recvTime[i] != 0) {

tmp\_recv++

if(prev\_time != 0) {

delay = recvTime[i] - prev\_time

e2eDelay = recvTime[i] - sendTime[i]

if(delay < 0) delay = 0

if(prev\_delay != -1) {

jitter1 += abs(e2eDelay - prev\_e2eDelay)

jitter2 += abs(delay-prev\_delay)

jitter3 += (abs(e2eDelay-prev\_e2eDelay) - jitter3) / 16

jitter4 += (abs(delay-prev\_delay) - jitter4) / 16

}

# This 'if' is introduce to obtain clearer

# plots from the output of this script

if(delay >= tic\*10) {

printf(" %10g %10s %5d %5d %15g %18g\n", \

flow,flow\_t,src,dst,(prev\_time+1.0),0)

printf(" %10g %10s %5d %5d %15g %18g\n", \

flow,flow\_t,src,dst,(recvTime[i]-1.0),0)

}

currTime += delay

if (currTime >= tic) {

printf(" %10g %10s %5d %5d %15g %16g %16g %16g %16g\n", \

flow,flow\_t,src,dst,recvTime[i],jitter1\*1000/tmp\_recv, \

jitter2\*1000/tmp\_recv,jitter3\*1000,jitter4\*1000)

jitter1 = jitter2 = jitter3 = jitter4 = 0

currTime = 0

tmp\_recv = 0

}

prev\_delay = delay

prev\_e2eDelay = e2eDelay

}

prev\_time = recvTime[i]

processed++

}

}

}

END {

printf("\n\n")

}

function abs(value) {

if (value < 0) value = 0-value

return value

}