NETWORK SIMULATOR

NS2

JUST THE NEEDED STUFF :D

No guarantee for the authenticity of information. To be used just as a guide for NITT ‘15-’19 CSE batch

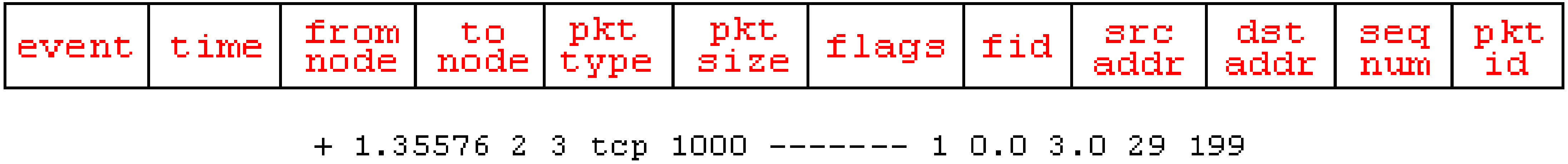
COMPONENTS

Any network simulation includes :

- **tcl** script to specify **ns file.tcl**
  - Nodes
  - Links
  - Traffic
  - Scheduling
- **tr** Trace files
  - Trace file to trace the whole simulation and for analysis (post processing)
  - Trace file for nam (network animation) **nam file.nam**
- **awk** scripts for analysis/performance evaluation **awk –f file.awk file.tr**
- Graph for graphical representation of the analysis **xgraph file.txt**

TRACE FILE

WIRED - invoke by **trace-all**
Nam file - invoke by **namtrace-all**



event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

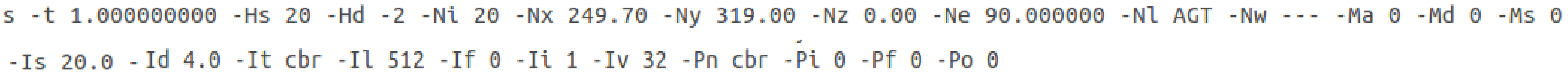
```
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
```

WIRELESS - has different formats. Additional trace information can be turned ON/OFF

Old format **trace-all**

r 1.728405901 _2_ RTR --- 10 message 32 [0 ffffffff 8 800] [energy 999.820034 ei 0.172 es 0.000 et 0.001 er 0.007] ------- [8:255 -1:255 32 0]

New format (so much more information) **use-newtrace**



```
s -t 1.000000000 -Hs 20 -Hd -2 -Ni 20 -Nx 249.70 -Ny 319.00 -Nz 0.00 -Ne 90.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0
-Is 20.0 -Id 4.0 -It cbr -Il 512 -If 0 -Ii 1 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
```

Nam file **namtrace-all-wireless**

We used old format in lab. Use format depending on the need.
This is just 1 example. Many formats are there in old and new formats.

TRACE FILE FORMAT

```
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
```

WIRED

1. Event

+ enqueue
- dequeue
d dropped
r received

- Data is sent in the form of packets
- Every link has a queue into which packets are enqueued
- Packets are then dequeued from the queue and transmitted through the link
- When queue is full, packets are not accepted and/or dropped

2. Time - Time at which the record is made from the simulation (accurate up to 5 to 6 decimals).
That's pretty much why trace files are HUGE!

3,4. Source, Dest Node - Source node and destination node of the packet (intermediate source and destination)

5. Packet type - tcp, cbr, ack, etc.

6. Packet size - in bytes

7. Flag - 7 flag strings are available *(don't bother)*
'-' is used if disabled

8. Flow ID - usually specified by us using ***fid_***

9, 10. Source, Destination Address – in the form ***address.port*** (node.port)

11. Sequence number – A packet *can* have a sequence number
(to be processed in that sequence)

12. Packet ID – every packet has a unique ID
assigned to it when generated

TRACE FILE FORMAT WIRELESS

Old trace format

```
s 0.032821055 _1_ RTR  --- 0 message 32 [0 0 0 0]  ------- [1:255 -1:255 32 0]
```

There are no enqueue/dequeue actions since there are no links

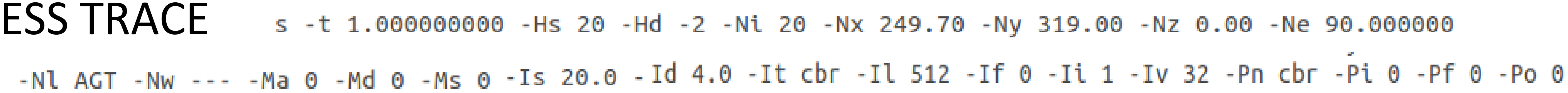
Column	Function
1	Action s (send), r (receive), d (dropped)
2	Time
3	Node at which action occurs
4	Layer (PHY, MAC, LL, AGT, etc.)
5	Flag
6	Sequence number
7	Packet type (CBR, RTS, DSR, etc.)
8	Size of packet at current layer
[9 10 11 12]	[duration in mac layer destination source mac type]

OLD WIRELESS TRACE (cont)

s 0.032821055 _1_ RTR --- 0 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]

13 to 19	Flags (including energy)
[20 21 22 23]	[source destination ttl header next hop]

NEW WIRELESS TRACE



```
s -t 1.000000000 -Hs 20 -Hd -2 -Ni 20 -Nx 249.70 -Ny 319.00 -Nz 0.00 -Ne 90.000000
-Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Is 20.0 -Id 4.0 -It cbr -Il 512 -If 0 -Ii 1 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
```

- s (send), r (receive), d (dropped), f (forward)
- -t (time)
- -H (next hop info)
- -N (node properties - x y z coordinates of node position, energy level, etc.)
- -M (Packet info at mac level)
- -I (Packet info at IP level)
- -P (Packet info at Application level)

XGRAPH xgraph filename

Operates on a file with 2 columns (x and y coordinates) separated by a space. We create this file from the trace file.

- Plot multiple graphs together xgraph file1 file2 file3
- Specify the dimensions of the graph - geometry 800x400

(it chooses an appropriate geometry if not specified)

- Specify that it is a bar graph -bar
- Specify the bar width -brw 1
- Specify there should be no lines -nl *(usually used with bar graphs)*
- Specify a title -t string

PERFORMANCE METRICS WIRED

- Throughput - Average transform rate (amount of data that moved successfully from one node to another in a given time period)

$$\text{throughput} = \frac{\text{total bits received by the destination}}{\text{Time of observation}}$$

- Packet Delivery Ratio $= \dfrac{\text{packets received}}{\text{Packets sent}}$

While computing packets for wireless, ensure layer is AGT

- Packet Loss Ratio $= \dfrac{\text{packets dropped (or) packets sent – received}}{\text{Packets sent}}$

- End to end delay – time taken for a packet to reach the destination from the source

$$\text{average delay} = \frac{\text{total delay for all packets}}{\text{Total packets}}$$

PERFORMANCE METRICS WIRED

- Fairness - to determine if users are receiving a fair share of resources

$$\frac{(\sum_{i=1}^{n} t_i)^2}{n * \sum_{i=1}^{n}(t_i)^2}$$

T_i – throughput of ith communication
n – no of communications

- Control overhead = $\frac{\text{No of control packets}}{\text{Total no of packets}}$

- Hop Count - no of hops required to reach the destination from the source *(can be calculated from the trace file by keeping track of the no of packets 'sent')*

AWK SCRIPTS

2 ways for computation of parameters – AWK script using trace file, procedure in the tcl script *(AWK scripts are needed for all metrics other than throughput)*

An AWK script has 3 parts

- BEGIN – executed prior to text processing. Usually for initialization of variables (global)
- Content – Computation part which processes the text file
- END – gets executed at the end

The only statements we 'll use are IF ELSE, basic arithmetic operations and print statements. Similar to C

- We 'll run the awk script on our trace file, every line of the trace file runs through the script
- Execution : *awk –f awkfile.awk tracefile.tr > outputfile*
- Pass arguments from tcl script : *awk –v var=value –f awkfile.awk tracefile.tr*

Example 1 : THROUGHPUT

```
BEGIN {
                recvsize=0;
                currenttime=0;
}
{
        event  = $1;
        toNode = $4;
        currenttime = $2;
         if(event =="r" && toNode=="5" ) {
                recvsize+=$6;
         }
printf("%f %f\n", currenttime, (8*recvsize)/(100000*currenttime);
}
END {
}
```

TIPS

- To print 1 value, ***printf*** statement is specified in the END part
- Values from trace file are accessed as ***$column***
- No separate initialization for arrays is needed (*you can just assign a value as delay[i]=val*)

While executing in the terminal,

- **>**   will write/replace the output to a new/existing file
- **>>**   will append the output to a file

Example 2 : PACKET LOSS RATIO

```
BEGIN {
                pksend=0;
                pktrec=0;
                currenttime=0;
}
{
                event  = $1;
                toNode = $4;
                fromNode = $3;
                currenttime = $2;
                if(event =="r" && (toNode=="9"  || toNode=="5"))
                {
                                pktrec++;
                }
```

```
 if (event=="+" && (fromNode=="1"  || fromNode=="2"))
           {
                                pksend++;
           }
     if (pksend > 0)
     {
        printf("%f %f\n",currenttime,(pksend-pktrec)/pksend);
      }
}
END {
}
```

Example 3 : END TO END DELAY

```
BEGIN {
            pksend=0;
            pktrec=0;
            currenttime=0;
           tdelay=0;
}
{
            event  = $1;
            toNode = $4;
            fromNode = $3;
            currenttime = $2;
```

```
if(event =="r" && (toNode=="5"  || toNode=="4"))
  {
      tdelay+=currenttime-starttime[$11];
  }
if (event=="+" && (fromNode=="1"  || fromNode=="2"))
  {
       starttime[$11]=currenttime;
  }
printf("%f %f\n",currenttime,tdelay);
}
END {
}
```

Example 4 : HOPCOUNT

```
BEGIN {
Hopcount=0;
}
{
  If($1=="+“ && $3!=srcNode && $5="tcp“) {
            Hopcount++;
  }
}
END{
print Hopcount;
}
```

Example 5 : ENERGY (wireless)

```
BEGIN {
energy=0;
}
{
    If ($2<now){
    energy+=$22+$20+$18+$16;
}
END{
print energy
}
```

srcNode and now are passed as arguments from the TCL script

TCL BASICS

- Every program starts with creating a Simulator *o*bject and ends with invoking the simulator
- A variable is declared using ***set <var_name> <var_value>***

  set n 5 ***set X_*** ***set y $x***

- If value is not specified after variable name, it returns the value of the variable
- A variable is accessed using ***$***
- ***exec*** – to execute a command

  exec nam out.nam

- We exit the program by ***exit 0***

```
set ns [new Simulator]
#rest of the program
$ns run
```

FILES

- Any file needs to be created before we can write data into it
- Can be created in write (**w**) or append (**a**) access
- We set a file handler (variable) for the file and then access it

- we use **puts** to write data into a file from the tcl script

```
set file1 [open test.txt w]
set x 2
set y 3
puts $file1 “1  $x  $y“
```

this prints **1 2 3** in test.txt

Creating trace files

```
set trfile [open out.tr w]
$ns trace-all $trfile
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

SCHEDULING AND FINISH

- Every simulation has a finish procedure, which is called at the end
- Procedures are declared using ***proc***
- We need to flush all buffers and close the trace files

(in the example we instruct the finish procedure to execute the nam file before exiting)

```
proc finish {} {
            global ns trfile namfile
            $ns flush-trace
            close $namfile
            close $trfile
            exec nam out.nam &
            exit 0
}
```

- We need to schedule events and procedures we create
- The time and the event/procedure is specified within “ “

<simulator object> at <time> <event>

```
$ns at 0.5 "$ftp start"
$ns at 3 "$ftp stop“
$ns at 3.1 “finish”
```

NODES AND LINKS

- Nodes are created using ***node*** **set n0 [$ns node]** *(a node n0 is created)*
- To change shape of a node **$n0 shape box** *(circle/hexagon/box) circle by default*
- To change color of a node **$n0 color red** *(color is case insensitive – red/Red)*
- To attach a label to a node **$n0 label “I am node 0"**

- Links are created as **$ns duplex-link $n0 $n1 2Mb 10ms DropTail**

  - Link type can be simplex/duplex
  - Queue type can be DropTail/SFQ/RED etc
  - Link Bandwidth needs to be specified in Mb
  - propagation delay needs to be specified in ms

Here a duplex link of BW 1Mbps and delay of 10ms is created between n0 and n1

LINKS AND QUEUES

- Orientation of links - left, right, up, down, left-up, right-down etc.

  - specified using ***op***

  $ns duplex-link-op $n0 $n1 orient down

Refer this link for queue types goo.gl/MWEKGo

- Each queue type has its own default buffer size

  $ns queue-limit $n0 $n1 100 (queue size is set to 100 packets for this specific queue)

  Queue/RED set queue-limit 200 (for specific class of queues)

  Queue set limit_ 5 (all queues)

- Queue statistics can be analyzed using **queue monitor**

  (actually pretty useful, but we don’t utilize it. Refer internet)

TRANSMISSIONS - AGENT

- An agent needs to be attached to a node to enable transmission
- There are around 28 agents like TCP, UDP, TCPSink, etc
- TCP – Connection is established before transmission of data
- UDP – Connectionless transfer of data

STEP 1 : *Create an instance of the agent (source)* **set tcp1 [new Agent/TCP]**

STEP 2 : *Attach the agent to a node* **$ns attach-agent $n0 $tcp1**

- Similarly **TCPSink** agent is created for the destination. **Null** agent for UDP

STEP 3 : *Connect the source and the sink* **$ns connect $tcp1 $sink1**

STEP 4 : *Flow id can be set (used to specify colors/analysis)* **$tcp1 set fid_ 1**

TRANSMISSIONS - APPLICATION

- Now that transport protocol is specified, we need to create the application
- Applications can be FTP, Telnet, Traffic (CBR, Exponential, Pareto, Trace)

(In our examples, we usually use FTP with TCP and CBR with UDP)

- We need to create an instance of the application and attach it to the agent

```
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
```

- We can additionally specify packet size and rate for CBR (Constant Bit Rate) traffic

```
$cbr2 set packetSize_ 1024
$cbr2 set rate_ 1Mb
```

ADDITIONAL TCL TIPS

- We implement mac protocols in the bus topology ***Csma*** or ***Csma/Cd***

set lan [$ns newLan "$n1 $n2 $n3 $n4 $n5" 0.1Mb 50ms LL Queue/DropTail MAC/Csma Channel]

- Using loops/conditions *(it is very convenient to use for loops for node/link creation)*

```
while { $i < 10 } {
set node($i) [$ns node]
set i [expr $i+1]
}
```

```
for { set i 0 } { $i < 10 } { set i [expr $i+2] } {
puts $i
}
```

```
if { condition } {
if - body
} else {
else - body
}
```

- Executing an expression **expr ($i+1)%7**

- To select a random number *(between 0 and 19)* **expr { int(rand()*20 }**

WIRELESS NETWORKS

Wireless networks require extra lines of code to

- Create god
- Create topography object
- Configure wireless nodes

Inbuilt source codes works only with **ns_** *(hence we usually create a simulator object like this)*

```
set ns_  [new Simulator]
```

Trace commands

```
$ns_ trace-all $trfile
$ns_ namtrace-all-wireless $namfile $val(x) $val(y)
$ns_ eventtrace-all
```

(x and y are the dimensions of your network topology)

GOD & TOPOLOGY

- General Operation Director – stores information about each node
- Hence no of nodes must be specified while creation to reserve space
- We use **create-god <no of nodes>** **set god_ [create-god $val(nn)]**

(god_ is used when using inbuilt source codes, similar to ns_)

- We use a flat-grid topology for wireless

set topo [new Topography] *create a topography object*

$topo load_flatgrid $val(x) $val(y) *specify dimensions of topology*

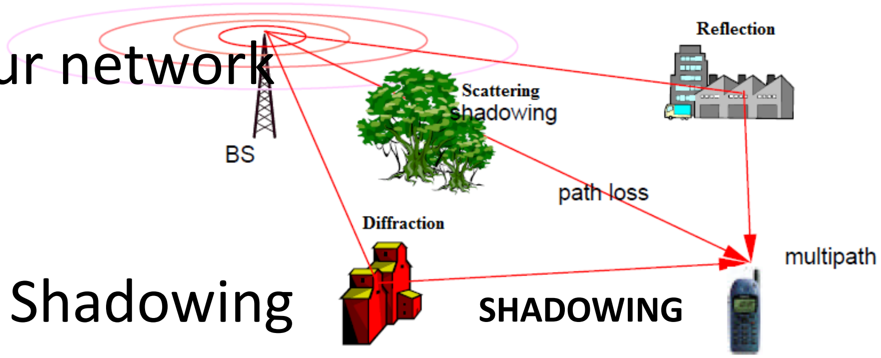
NODE CONFIGURATION

- We need to configure the nodes before creating them
- We usually set default values for all our properties beforehand

SETTING UP VALUES

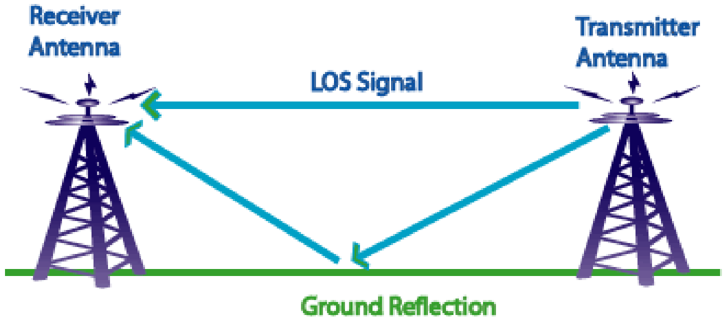
Channel to be used. We use a wireless channel for our network

```
set val(chan) Channel/WirelessChannel;
```



Signal propagation type – can be Two Ray Ground or Shadowing

```
set val(prop) Propagation/TwoRayGround;
```



Physical layer type

```
set val(netif) Phy/WirelessPhy;
```

NODE CONFIGURATION (cont)

Link layer type

set val(ll) LL;

Medium Access control - 802_11 (csma/ca for wireless), Tdma, etc

set val(mac) Mac/802_11;

Type of antenna *(ns2.35 has support only for omni antenna)*

set val(ant) Antenna/OmniAntenna;

Type of interface queue – DropTail, DropTail/PriQueue

set val(ifq) Queue/DropTail;

Interface queue length

set val(ifqlen) 50;

Routing Protocol – DSR, AODV, DSDV, FLOODING

set val(rp) DumbAgent;

(DumbAgent is used when no routing is required)

OTHER VALUES

No of nodes in the network

```
set val(nn) 20;
```

Value of network dimensions

```
set val(x) 500;
set val(y) 400;
```

We use two arbitrary variables to refer to our mobility and transmission files generated using **setdest** and **cbrgen** files

```
set val(cp) “trans_10”;
set val(sc) “dest_10”;
```

To include these in our program, add these lines after node creation

```
source $val(cp)
source $val(sc)
```

NODE CONFIGURATION

We use the values defined previously to configure our nodes

```
$ns_ node-config -adhocRouting $val(rp)\          adhoc routing protocol
                 -llType $val(ll)\                link layer
                 -macType $val(mac)\              medium access control
                 -ifqType $val(ifq)\              type of interface queue
                 -ifqLen $val(ifqlen)\            queue length
                 -antType $val(ant)\              antenna
                 -propType $val(prop)\            propagation type
                 -phyType $val(netif)\            physical layer
                 -channelType $val(chan)\         channel
                 -topoInstance $topo\             topography instance for the network
                 -agentTrace ON\                  tracing at agent level
                 -routerTrace OFF\                tracing at router level
                 -macTrace ON\                    tracing at mac level
                 -movementTrace OFF               mobile node movement logging
```

NODE CONFIGURATION

When accessing energy of the nodes, we include the following

-energyModel "EnergyModel"\	*energy model is enabled*
-initialEnergy 1000\	*initial energy for each node*
-rxPower 1.0\	*power for receiving 1 packet*
-txPower 1.0\	*power for transmitting 1 packet*
-sleepPower 0.5\	*power consumed during sleep state*
-transitionPower 0.2\	*power consumed during state transition from sleep to idle*
-transitionTime 0.001\	*time taken during transition*
-idlePower 0.1	*power consumed during idle state*

(Default values will be used for the powers if not specified)

trace file while using energy model : [**energy 998.9992 ei 1.000 es 0.000 et 0.000 er 0.001**]

ei – energy consumption in IDLE state

et – energy consumed in transmission

energy – residual energy

es – energy consumption in SLEEP state

er – energy consumed in receiving

NODES

Now that node configuration is done, we do the following to create our wireless network

- Create the node (similar to wired network)
- Set up their positions (set up the x, y, and z coordinates and their initial positions)
- Set their mobility (we usually randomize the node’s mobility)
- Set up transmission (either ftp or cbr traffic – similar to wired)

NODE CREATION

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

creates nn nodes - node_(0), node_(1),…node_(nn-1)

NODES

POSITION

```
$node_(0) set X_ 283                         node location
$node_(0) set Y_ 425
$node_(0) set Z_ 0                           z coordinate is always 0
$ns initial_node_pos $node_(0) 30            node size
```

This has to be done for every node.

```
for {set i 0} {$i < 5 } {incr i} {
        $node_($i) set X_ [expr rand()*500]
        $node_($i) set Y_ [expr rand()*500]
        $node_($i) set Z_ 0.0
        $ns initial_node_pos $node_($i) 20
}
```

To set node radius : **$n0 radius 20** *sets radius of n0 as 20*

To set distance : **$god_ set-dist 3 4 2** *sets distance between node 3 and 4 as 2 hops*

NODES

MOBILITY

$ns at 1.2 “$n0 setdest 300 200 20” *at 1.2, n0 will start moving to (300,200,20)*

$n0 random-motion 1 *random motion is enabled by 1 and disabled by 0*

To generate random node positions and mobility, we use **setdest** file. Execute as **./setdest**

TRANSMISSIONS

Similarly, random transmissions can be generated using **cbrgen** file. Execute as **ns cbrgen**

(1 less than the total nodes must be specified in cbrgen)

Transmissions can be generated manually also, just like wired networks

```
set udp [new Agent/UDP]
$ns attach-agent $node_(0) $udp
set null [new Agent/Null]
$ns attach-agent $node_(1) $null
$ns connect $udp $null
```

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1024
$cbr set interval_ 0.1
$ns at 1.0 "$cbr start"
$ns at 5.0 "$cbr stop
```

ADDITIONAL TIPS

Calculating overhead for different routing protocols involves different control packets. Refer your trace file and write your awk scripts

- Control overhead = $\frac{\text{RTS or CTS packets}}{\text{Total packets}}$
- Received Signal Strength = $\frac{\text{Transmission Power}}{\text{Distance bet source and node}}$
- To get the x co-ordinate of node 0, **[$n0 set X_]**
- To disable RTS or CTS, **Mac/802_11 set RTSThreshold_ 1000**

  (RTS is used for all packets greater than size 1000)

  Thus to enable RTS for all packets, **Mac/802_11 set RTSThreshold_ 0** *(Similarly for CTS)*
- To simulate flooding, the sample flooding.tcl file is used.

  (sample tcl files are available in ~ns/ns-2.35/tcl/ex/)

ADDITIONAL TIPS

- While calculating PDR or PLR in wireless, packets in the AGT level must be considered (AgentTrace must be ON)
- Jitter is the variation of time delay for packets to reach the destination (difference in delay)
- In case of Packet Delivery (or Loss) **Ratio**, we divide the received packets by total packets sent. In case of **Rate**, we divide by the time
- MANET (Mobile Adhoc Network) - all nodes are mobile and they use an adhoc routing protocol (no central coordinator) like AODV, DSDV, DSR
- VANET (Vehicular Adhoc Network) and WSN (Wireless Sensor Network) are types of MANETs
- VANET – characterized by high node mobility and rapid topology changes

ADDITIONAL TIPS

- WSN – consists of multiple sensor nodes. Sense power needs to be specified for sensor nodes. Communication range and sensing range need to be specified.

```
$val(netif1) set CSThresh_ 2.28289e-11 ;        #sensing range of 500m
$val(netif1) set RXThresh_ 2.28289e-11 ;        #communication range of 500m
```

- Commonly used Routing protocols for wired networks, **$ns rtproto DV** (or LS)

  Distance Vector Routing – routing tables are sent to neighbors

  Link State Routing – information about the whole topology is known

- Commonly used Routing protocols for wireless networks – AODV, DSR, DSDV

  TABLE DRIVEN – all route information is maintained in the routing table - **DSDV**

  ON-DEMAND/EVENT DRIVEN – routing table contains information only about routes currently in use

  AODV (periodic routing packets)   **DSR** (routing packets are sent only when necessary)

https://www.ijarcce.com/upload/2013/december/IJARCCE2C-rajesh_COMPARATIVE-FINAL.pdf