UCF Stands For Opportunity

CDA6530: Performance Models of Computers and Networks

Chapter 10: Introduction to Network Simulator (NS2)

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

Some Contents are from….

- USC ISI Network Simulator (ns) Tutorial 2002
  - http://www.isi.edu/nsnam/ns/ns-tutorial/tutorial-02/index.html
- Prof. Samir R. Das in Sonysb “CSE 590”
  - www.cs.sunysb.edu/~samir/cse590/ns2-lecture.ppt
- Tcl/TK Tutorial
  - www.umiacs.umd.edu/~hollingk/talks/tcl_tutorial.ppt
- http://www-scf.usc.edu/~bhuang
- *www.isi.edu/nsnam/**ns/ns-tutorial/wireless**.ppt*
- Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator "ns“
  - http://www.isi.edu/nsnam/ns/tutorial/index.html
- http://www.winlab.rutgers.edu/~zhibinwu/html/network_simulator_2.html

Where to Run NS2

- Our department unix server - eustis.eecs.ucf.edu has installed ns2
- First, you need to change default configuration
  - Modify the hidden file .profile under home directory
  - Add the following configuration

```
export PATH=$PATH:/usr/local/ns2/bin:/usr/local/ns2/tcl8.4.18/unix:/usr/local/ns2/tk8.4.18/unix
export LD_LIBRARY_PATH=/usr/local/ns2/otcl-1.13:/usr/local/ns2/lib
export TCL_LIBRARY=/usr/local/ns2/tcl8.4.18/library
```

  - Run ns2:
    - czou@eustis:~$ ns
- Unix Based. Runs also in windows using *cygwin*
  - Quit complicated to install in Windows
  - Windows installation and usage not introduced here

ns2- Network Simulator

- One of the most popular simulator among networking researchers
  - Open source, free
- Discrete event, Packet level simulator
  - Events like ‘received an *ack* packet’, ‘enqueued a data packet’
- Network protocol stack written in C++
- Tcl (Tool Command Language) used for specifying scenarios and events.
- Simulates both wired and wireless networks.

Goal of this tutorial

- Understand how to write Tcl scripts to simulate simple network topologies and traffic patterns.
- Analyze the trace files and understand how to evaluate the performance of networking protocols and operations.

"Ns" Components

- Ns, the simulator itself
- Nam, the network animator
  - Visualize *ns* (or other) output
  - Nam editor: GUI interface to generate ns scripts
    - Since we only run ns2 in remote Unix server, we will not introduce Nam usage in this class
- Pre-processing:
  - Traffic and topology generators
- Post-processing:
  - Simple trace analysis, often in Awk, Perl, or Tcl
  - You can also use grep (under linux), or C/java

C++ and OTcl Separation

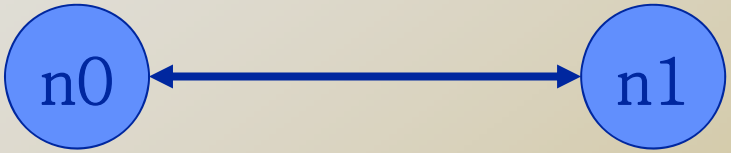
- "data" / control separation
  - C++ for "data":
    - per packet processing, core of *ns*
    - fast to run, detailed, complete control
  - OTcl for control:
    - Simulation scenario configurations
    - Periodic or triggered action
    - Manipulating existing C++ objects
    - fast to write and change

Basic Tcl

```
variables:
set x 10
set z x+10  # string 'x+10' to z
set y [expr $x+10]
puts "x is $x"

functions and expressions:
set y [expr pow($x, 2)]

control flow:
if {$x > 0} { return $x } else {
    return [expr -$x] }
while { $x > 0 } {
    puts $x
    incr x –1
}
```

```
procedures:
proc pow {x n} {
    if {$n == 1} { return $x }
    set part [pow x [expr $n-1]]
    return [expr $x*$part]
}

Arrays:
set matrix(1,1) 140
```

Simple two node wired network



Step 1:

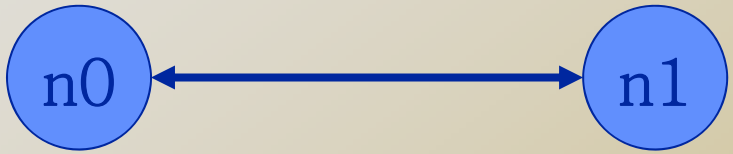
```
#Create a simulator object
# (Create event scheduler)
set ns [new Simulator]
```

Name of scheduler

Step 2:

```
#Open trace files
set f [open out.tr w]
$ns trace-all $f
```

Simple two node wired network



Step 3:

```
#Create two nodes
set n0 [$ns node]
set n1 [$ns node]
```

Step 4:

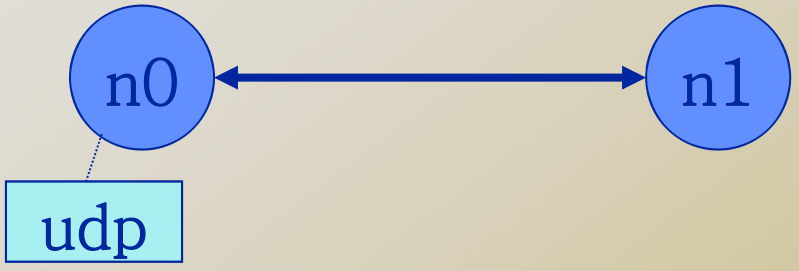
```
#Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

Simple two node wired network

```
#Create a simulator object
set ns [new Simulator]
#Open trace files
set f [open out.tr w]
$ns trace-all $f
#Define a 'finish' procedure
proc finish {} {
        global ns
        $ns flush-trace
        close $f
        exit 0
}
#Create two nodes
set n0 [$ns node]
set n1 [$ns node]
#Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

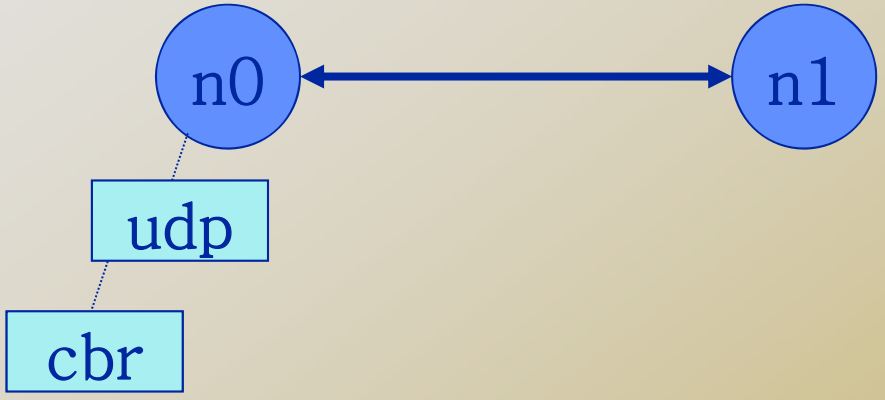
But we have no traffic!

Adding traffic to the link



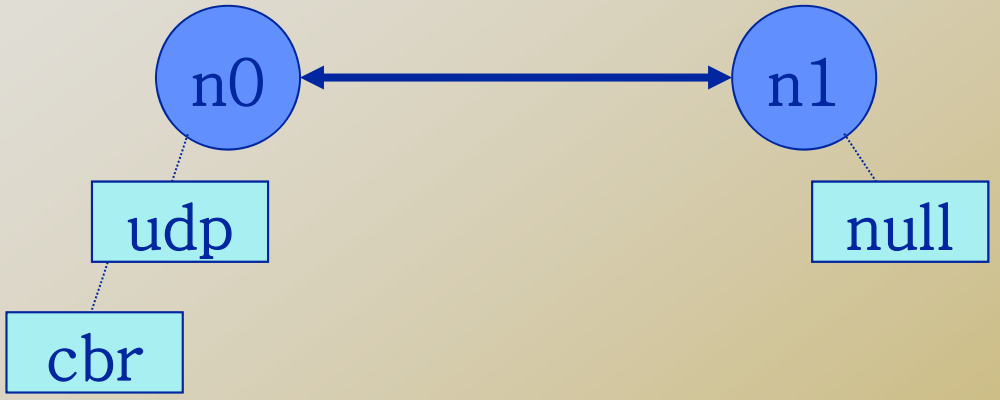

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

Adding traffic to the link



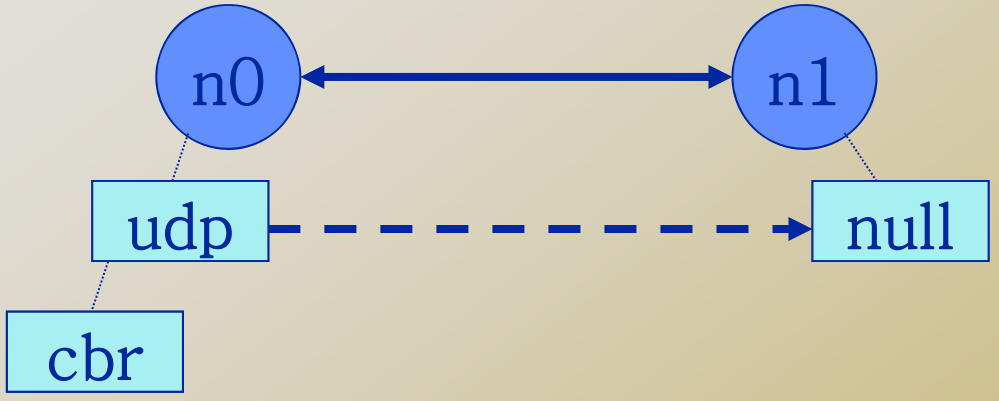

```
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

Adding traffic to the link




```
#Create a Null agent (a traffic sink) and
attach it to node n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

Adding traffic to the link




```
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop”
$ns at 5.0 "finish"
$ns run
```

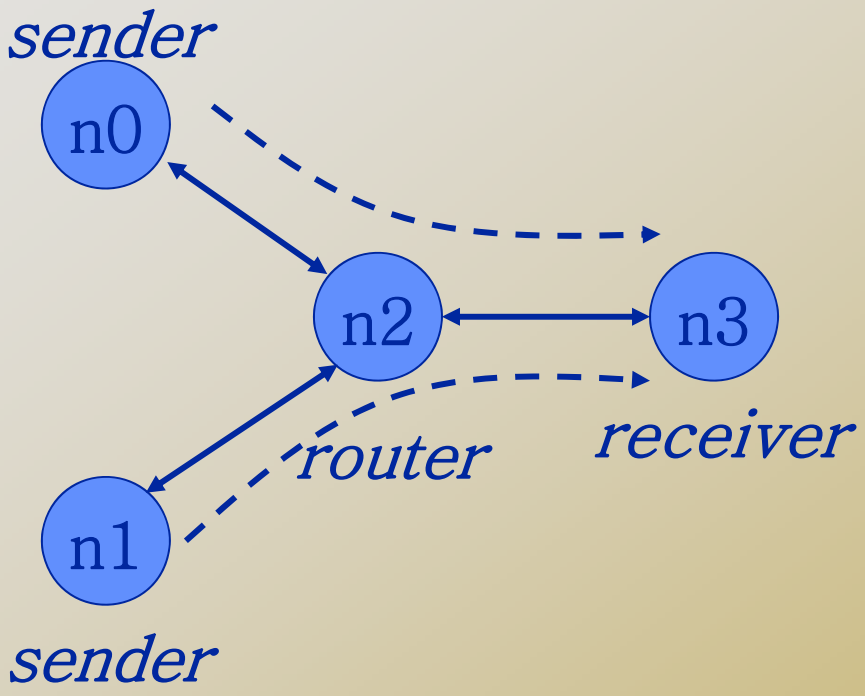
Record Simulation Trace

- Packet tracing:
  - On all links: $ns trace-all [open out.tr w]
  - On one specific link: $ns trace-queue $n0 $n1$tr

```
<Event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
    + 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    - 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    r 1.00234 0 2 cbr 210 ------- 0 0.0 3.1 0 0
```

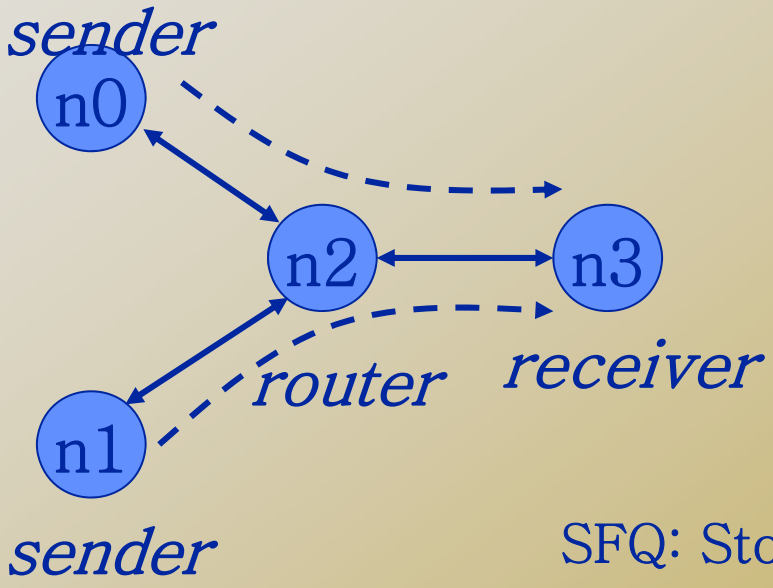
  - Event “+”: enqueue, “-”: dequeue; “r”: received

Simulate a simple topology – UDP Traffic




```
#Create a simulator object
set ns [new Simulator]
#Open trace files
set f [open out.tr w]
$ns trace-all $f
#Define a 'finish' procedure
proc finish {} {
    global ns
    $ns flush-trace
    exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

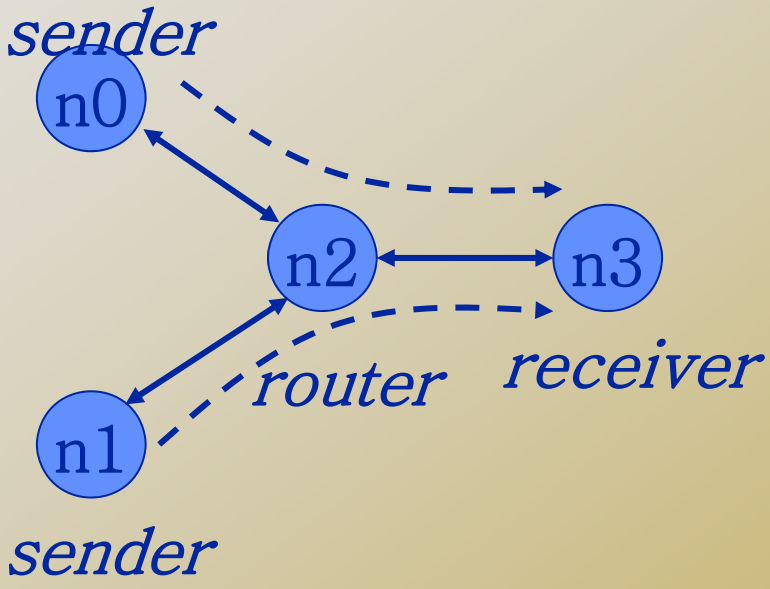
Simulate a simple topology – UDP Traffic



SFQ: Stochastic Fair queuing

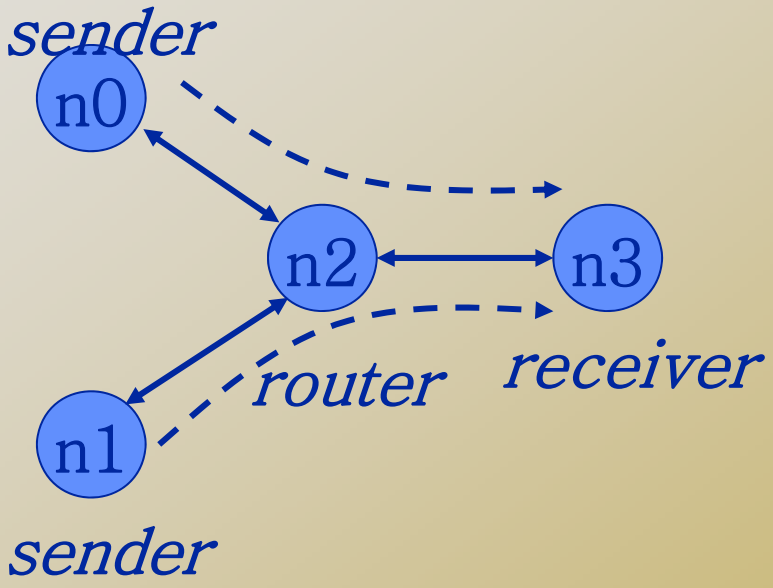
```
#Create links between the nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
```

Simulate a simple topology – UDP Traffic



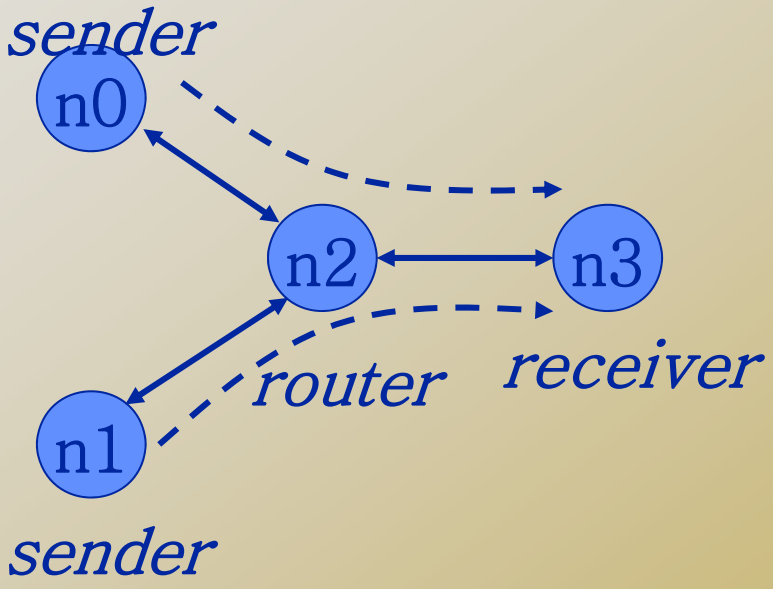

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set class_ 1  # fid in trace file
$ns attach-agent $n0 $udp0
```

Simulate a simple topology – UDP Traffic



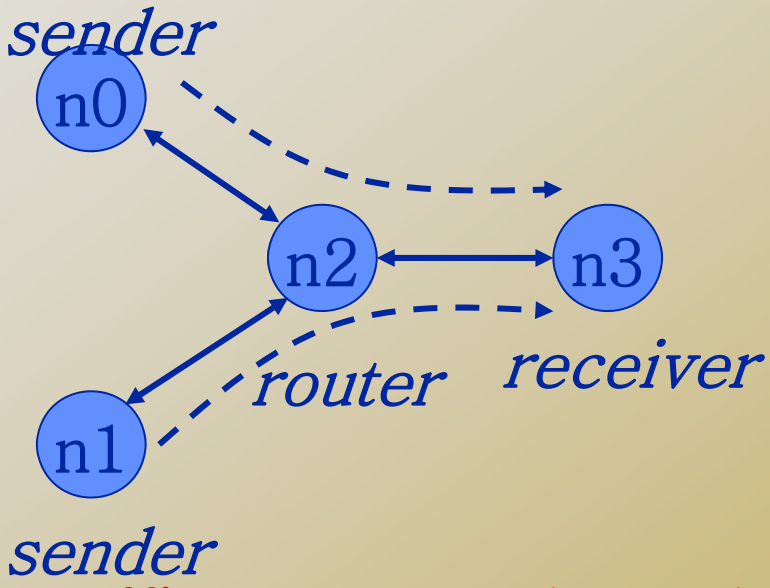

```
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

Simulate a simple topology – UDP Traffic



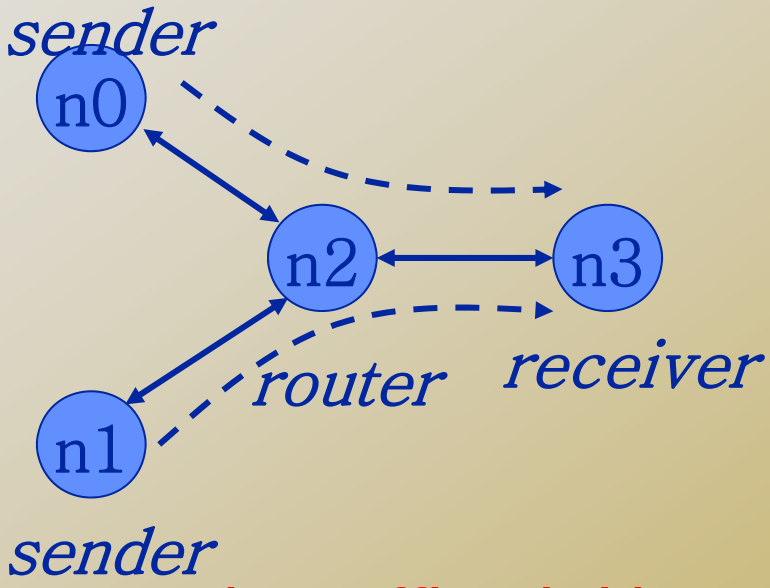

```
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1
```

Simulate a simple topology – UDP Traffic



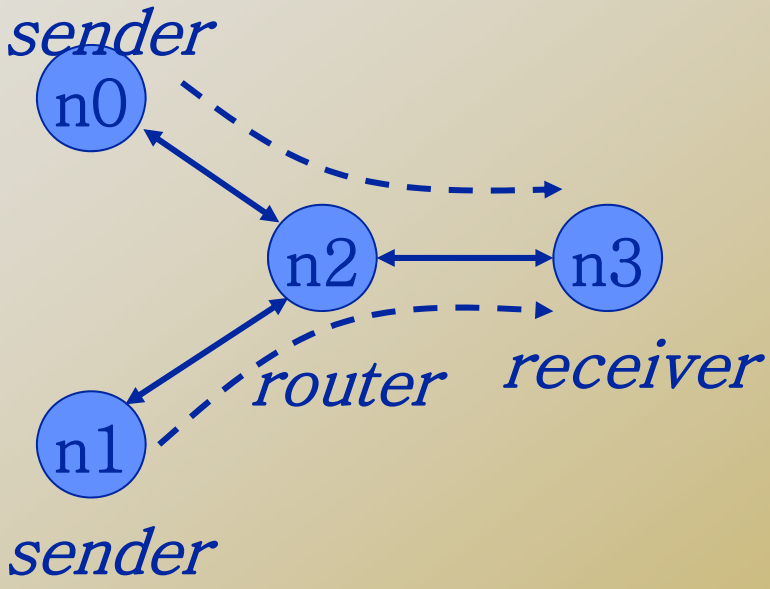

```
# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
```

Simulate a simple topology – UDP Traffic




```
#Create a Null agent (a traffic sink) and attach it to
   node n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

Simulate a simple topology – UDP Traffic



#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0

Simulate a simple topology – UDP Traffic

```
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of
   simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

Trace Analysis

http://nsnam.isi.edu/nsnam/index.php/NS-2_Trace_Formats

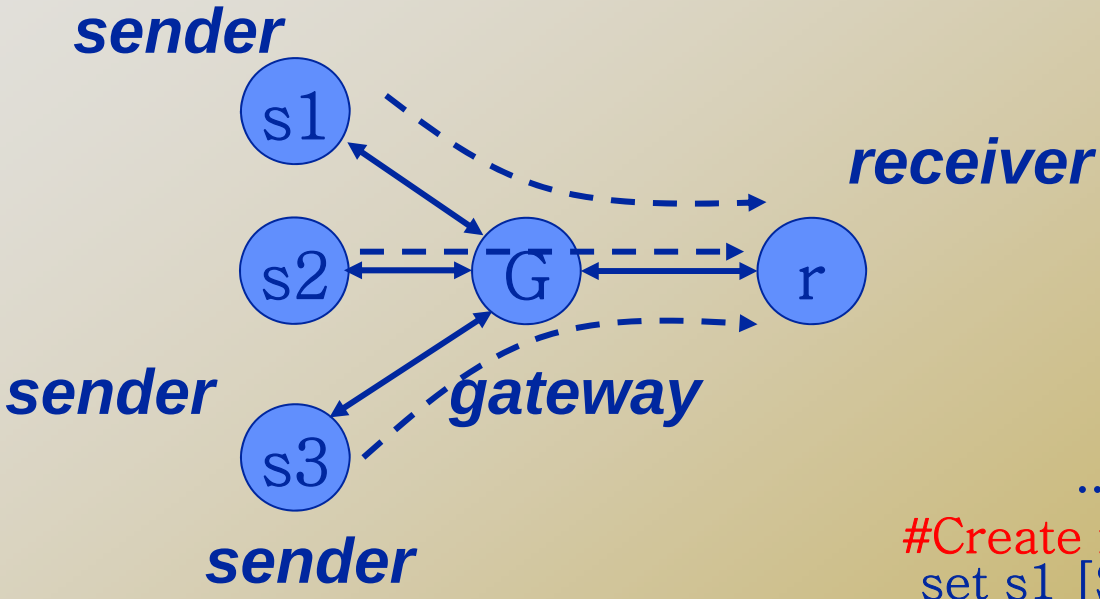
event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop    (at queue)
```

```
src_addr : node.port (3.0)
dst_addr : node.port (0.0)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

TCP Traffic



- 0, 1, 2 are senders
- 3 is a Gateway
- 4 receiver

```
……
#Create four nodes
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]
set G [$ns node]
set r [$ns node]
#Create links between the nodes
……
```

TCP Traffic

- #Create a TCP agent and attach it to node s1

```
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $s1 $tcp1
$tcp1 set window_ 8
$tcp1 set fid_ 1
```

- "window_" is the upperbound of congestion window in a TCP. It is 20 by default.

TCP Traffic

- #Create a TCP agent and attach it to node s2

```
set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $s2 $tcp2
$tcp2 set window_ 8
$tcp2 set fid_ 2
```

- #Create a TCP agent and attach it to node s3

```
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $s3 $tcp3
$tcp3 set window_ 4
$tcp3 set fid_ 3
```

TCP Traffic

- #Create TCP sink agents and attach them to node r

```
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]
set sink3 [new Agent/TCPSink]

$ns attach-agent $r $sink1
$ns attach-agent $r $sink2
$ns attach-agent $r $sink3
```

TCP Traffic

- #Connect the traffic sources with the traffic sinks

  $ns connect $tcp1 $sink1
  $ns connect $tcp2 $sink2
  $ns connect $tcp3 $sink3

- You cannot connect two TCP sources to the same TCP sink
  - You can do that for UDP traffic

TCP Traffic

- #Create FTP applications and attach them to agents

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
```

TCP Traffic

```
#Define a 'finish' procedure
proc finish {} {
        global ns
        $ns flush-trace
        exit 0
}

$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$ftp2 start"
$ns at 0.1 "$ftp3 start"
$ns at 5.0 "$ftp1 stop"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp3 stop"
$ns at 5.25 "finish"
$ns run
```

Trace Analysis

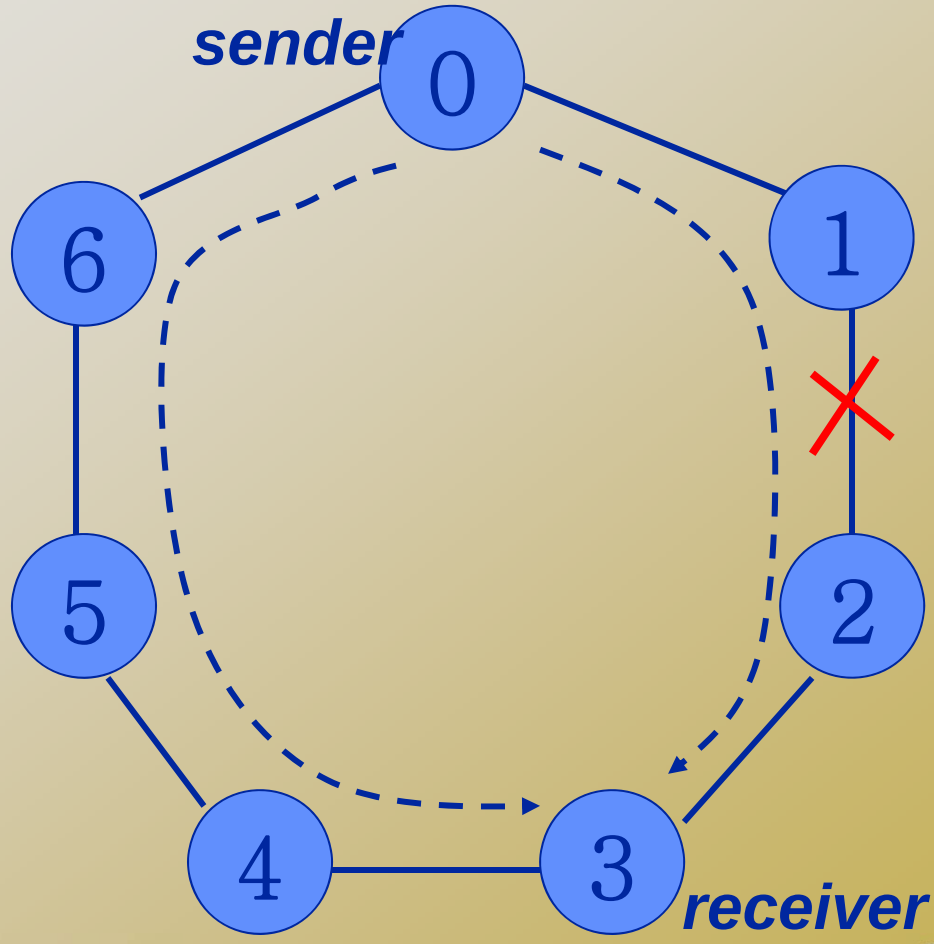
czou@eustis:~/ns2$ grep '^r' out.tr > 3TCP-receive-only.tr

```
r 0.1596 0 3 tcp 1040 ------- 1 0.0 4.0 1 6
r 0.15992 1 3 tcp 1040 ------- 2 1.0 4.1 1 8
r 0.16024 2 3 tcp 1040 ------- 3 2.0 4.2 1 10
r 0.16792 0 3 tcp 1040 ------- 1 0.0 4.0 2 7
r 0.16824 1 3 tcp 1040 ------- 2 1.0 4.1 2 9
r 0.16856 2 3 tcp 1040 ------- 3 2.0 4.2 2 11
r 0.17792 3 4 tcp 1040 ------- 1 0.0 4.0 1 6
r 0.18624 3 4 tcp 1040 ------- 2 1.0 4.1 1 8
r 0.18824 4 3 ack 40 ------- 1 4.0 0.0 1 12
r 0.19456 3 4 tcp 1040 ------- 3 2.0 4.2 1 10
r 0.19656 4 3 ack 40 ------- 2 4.1 1.0 1 13
r 0.19856 3 0 ack 40 ------- 1 4.0 0.0 1 12
r 0.20288 3 4 tcp 1040 ------- 1 0.0 4.0 2 7
r 0.20488 4 3 ack 40 ------- 3 4.2 2.0 1 14
r 0.20688 3 1 ack 40 ------- 2 4.1 1.0 1 13
r 0.2112 3 4 tcp 1040 ------- 2 1.0 4.1 2 9
r 0.2132 4 3 ack 40 ------- 1 4.0 0.0 2 17
r 0.2152 3 2 ack 40 ------- 3 4.2 2.0 1 14
```

Basic usage of Grep

- Command-line text-search program in Linux
- Some useful usage:
  - Grep ‘word’ filename # find lines with ‘word’
  - Grep –v ‘word’ filename # find lines without ‘word’
  - Grep ‘^word’ filename # find lines beginning with ‘word’
  - Grep ‘word’ filename > file2 # output lines with ‘word’ to file2
  - ls -l | grep rwxrwxrwx # list files that have ‘rwxrwxrwx’ feature
  - grep -v '^[0-9]‘ filename # find lines beginning with any of the numbers from 0-9
  - Grep –c ‘word’ filename # find lines with ‘word’ and print out the number of these lines
  - Grep –i ‘word’ filename # find lines with ‘word’ regardless of case
- Many tutorials on grep online

Complex topology and link failure



Complex topology and link failure

```
#Create a simulator object
set ns [new Simulator]
#Tell the simulator to use dynamic routing
$ns rtproto DV
#Define a 'finish' procedure
proc finish {} {
        global ns
        $ns flush-trace
        exit 0
}
```

Complex topology and link failure

```
#Create seven nodes
for {set i 0} {$i < 7} {incr i} {
set n($i) [$ns node]
}
#Create links between the nodes
for {set i 0} {$i < 7} {incr i} {
$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb
   10ms DropTail
}
```

Complex topology and link failure

```
#Create a UDP agent and attach it to node n(0)
.......
# Create a CBR traffic source and attach it to udp0
.......
#Create a Null agent (a traffic sink) and attach it to node n(3)
.......
#Connect the traffic source with the traffic sink
.......

#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

Trace Analysis

- czou@eustis:~/ns2$ grep '^r' ringLinkfailure.tr|more

```
r 0.984 0 1 cbr 500 ------- 1 0.0 3.0 94 158
r 0.987 2 3 cbr 500 ------- 1 0.0 3.0 89 153
r 0.988 1 2 cbr 500 ------- 1 0.0 3.0 92 156
r 0.989 0 1 cbr 500 ------- 1 0.0 3.0 95 159
r 0.992 2 3 cbr 500 ------- 1 0.0 3.0 90 154
r 0.993 1 2 cbr 500 ------- 1 0.0 3.0 93 157
r 0.994 0 1 cbr 500 ------- 1 0.0 3.0 96 160
r 0.997 2 3 cbr 500 ------- 1 0.0 3.0 91 155
r 0.998 1 2 cbr 500 ------- 1 0.0 3.0 94 158
r 0.999 0 1 cbr 500 ------- 1 0.0 3.0 97 161
r 1.002 2 3 cbr 500 ------- 1 0.0 3.0 92 156
r 1.004 0 1 cbr 500 ------- 1 0.0 3.0 98 162
r 1.007 2 3 cbr 500 ------- 1 0.0 3.0 93 157
r 1.009 0 1 cbr 500 ------- 1 0.0 3.0 99 163
r 1.010056 1 0 rtProtoDV 7 ------- 0 1.1 0.2 -1 164
r 1.012 2 3 cbr 500 ------- 1 0.0 3.0 94 158
r 1.012056 2 3 rtProtoDV 7 ------- 0 2.1 3.2 -1 165
r 1.014 0 1 cbr 500 ------- 1 0.0 3.0 100 166
r 1.019 0 1 cbr 500 ------- 1 0.0 3.0 101 167
r 1.020112 0 6 rtProtoDV 7 ------- 0 0.2 6.1 -1 170
r 1.022112 3 2 rtProtoDV 7 ------- 0 3.2 2.1 -1 171
r 1.022112 3 4 rtProtoDV 7 ------- 0 3.2 4.1 -1 172
```

```
r 1.044056 0 6 rtProtoDV 7 ------- 0 0.2 6.1 -1 184
r 1.048 6 5 cbr 500 ------- 1 0.0 3.0 104 174
r 1.049 0 6 cbr 500 ------- 1 0.0 3.0 107 187
r 1.05028 1 0 rtProtoDV 7 ------- 0 1.1 0.2 -1 189
r 1.05228 2 3 rtProtoDV 7 ------- 0 2.1 3.2 -1 190
r 1.053 6 5 cbr 500 ------- 1 0.0 3.0 105 181
r 1.054 0 6 cbr 500 ------- 1 0.0 3.0 108 188
r 1.057 5 4 cbr 500 ------- 1 0.0 3.0 103 173
r 1.058 6 5 cbr 500 ------- 1 0.0 3.0 106 182
r 1.059 0 6 cbr 500 ------- 1 0.0 3.0 109 191
r 1.062 5 4 cbr 500 ------- 1 0.0 3.0 104 174
r 1.063 6 5 cbr 500 ------- 1 0.0 3.0 107 187
r 1.064 0 6 cbr 500 ------- 1 0.0 3.0 110 192
r 1.067 5 4 cbr 500 ------- 1 0.0 3.0 105 181
r 1.068 6 5 cbr 500 ------- 1 0.0 3.0 108 188
r 1.069 0 6 cbr 500 ------- 1 0.0 3.0 111 193
r 1.071 4 3 cbr 500 ------- 1 0.0 3.0 103 173
r 1.072 5 4 cbr 500 ------- 1 0.0 3.0 106 182
r 1.073 6 5 cbr 500 ------- 1 0.0 3.0 109 191
r 1.074 0 6 cbr 500 ------- 1 0.0 3.0 112 194
r 1.076 4 3 cbr 500 ------- 1 0.0 3.0 104 174
r 1.077 5 4 cbr 500 ------- 1 0.0 3.0 107 187
r 1.078 6 5 cbr 500 ------- 1 0.0 3.0 110 192
r 1.079 0 6 cbr 500 ------- 1 0.0 3.0 113 195
r 1.081 4 3 cbr 500 ------- 1 0.0 3.0 105 181
```

Inserting Errors

- Creating Error Module
  - set loss_module [new ErrorModel]
  - $loss_module set rate_ 0.01
  - $loss_module unit pkt
  - $loss_module ranvar [new RandomVariable/Uniform]
  - $loss_module drop-target [new Agent/Null]
- Inserting Error Module
  - $ns lossmodel $loss_module $n0 $n1

Setup Routing

- Unicast
  - $ns rtproto <type>
  - <type>: Static, Session, DV, cost, multi-path
- Multicast
  - $ns multicast (right after [new Simulator])
  - $ns mrtproto <type>
  - <type>: CtrMcast, DM, ST, BST
- Other types of routing supported: source routing, hierarchical routing

Network Dynamics

- Link failures
  - Hooks in routing module to reflect routing changes
- Four models

```
$ns rtmodel Trace <config_file> $n0 $n1
$ns rtmodel Exponential {<params>} $n0 $n1
$ns rtmodel Deterministic {<params>} $n0 $n1
$ns rtmodel-at <time> up|down $n0 $n1
```

  - Parameter list

```
[<start>] <up_interval> <down_interval> [<finish>]
```

Wireless Network Simulation

- This section is mainly based on Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator "ns“
  - http://www.isi.edu/nsnam/ns/tutorial/index.html
- Others:
  - http://www.cs.binghamton.edu/~kliu/research/ns2code/

Simple 2 Nodes Simulation

- Simulate a very simple 2-node wireless scenario
- The topology consists of two mobilenodes
- The mobilenodes move about within 500mX500m area
- A TCP connection is setup between the two mobilenodes.
  - Packets are exchanged between the nodes as they come within hearing range of one another.
  - As they move away, packets start getting dropped.

- Define options:

```
# Define options #
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(ant) Antenna/OmniAntenna ;# Antenna type
set val(ll) LL ;# Link layer type
set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type
set val(ifqlen) 50 ;# max packet in ifq
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(rp) DSDV ;# ad-hoc routing protocol
set val(nn) 2 ;# number of mobilenodes
```

- Define NS simulator

  set ns_ [new Simulator]
- Define trace file

  set tracefd [open simple.tr w]

  $ns_ trace-all $tracefd
- Create topology object

  set topo [new Topography]
- Topography object with (x=500, y=500)

  $topo load_flatgrid 500 500

God (General Operations Director) Object

- Create God object:

  create-god $val(nn)
- God object stores:
  - number of mobilenodes
  - table of shortest number of hops required to reach from one node to another

Define how a mobile node should be created

```
$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 \
                 -channelType $val(chan) \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace OFF \
                 -movementTrace OFF
```

Manual Create Node Motion

- Create two nodes

```
for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node ]
        $node_($i) random-motion 0 ;# disable random motion
}
```

- Provide node position and movement(speed & direction)

```
# Provide initial (X,Y, for now Z=0) co-ordinates
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
$node_(1) set Z_ 0.0
```

- Produce some node movements

```
# Node_(1) starts to move towards node_(0)
  $ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
  $ns_ at 10.0 "$node_(0) setdest 20.0 18.0 1.0"
# Node_(1) then starts to move away from node_(0)
  $ns_ at 100.0 "$node_(1) setdest 490.0 480.0 15.0“
```

- $ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0" means at time 50.0s, node1 starts to move towards the destination (x=25,y=20) at a speed of 15m/s.

- Setup traffic flow between the two nodes:

```
# TCP connections between node_(0) and node_(1)
set tcp [new Agent/TCP]
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"
```

```
# Tell nodes when the simulation ends
 for {set i 0} {$i < $val(nn) } {incr i} {
            $ns_ at 150.0 "$node_($i) reset";
 }
$ns_ at 150.0001 "stop"
$ns_ at 150.0002 "puts \"NS EXITING...\“ ; $ns_ halt"
proc stop {} {
            global ns_ tracefd
            close $tracefd
}

puts "Starting Simulation..."
$ns_ run
```

Wireless Trace File Analysis

ACTION:	[s|r|D]: s -- sent, r -- received, D -- dropped
WHEN:	the time when the action happened
WHERE:	the node where the action happened
LAYER:	AGT -- application, RTR -- routing, LL -- link layer (ARP is done here) IFQ -- outgoing packet queue (between link and mac layer) MAC -- mac, PHY -- physical
flags:	
SEQNO:	the sequence number of the packet
TYPE:	the packet type cbr -- CBR data stream packet DSR -- DSR routing packet (control packet generated by routing) RTS -- RTS packet generated by MAC 802.11 ARP -- link layer ARP packet
SIZE:	the size of packet at current layer, when packet goes down, size increases, goes up size decreases
[a b c d]:	a -- the packet duration in mac layer header b -- the mac address of destination c -- the mac address of source d -- the mac type of the packet body
flags:	
[.....]:	[source node ip : port_number destination node ip (-1 means broadcast) : port_number ip header ttl ip of next hop (0 means node 0 or broadcast)]

Example of Trace Intepretation

s 76.000000000 _98_ AGT --- 1812 cbr 32 [0 0 0 0] ------- [98:0 0:0 32 0]

- Application 0 (port number) on node 98 sent a CBR packet whose ID is 1812 and size is 32 bytes, at time 76.0 second, to application 0 on node 0 with TTL is 32 hops. The next hop is not decided yet.

r 0.010176954 _9_ RTR --- 1 gpsr 29 [0 ffffffff 8 800] ------- [8:255 -1:255 32 0]

- The routing agent on node 9 received a GPSR broadcast (mac address 0xff, and ip address is -1, either of them means broadcast) routing packet whose ID is 1 and size is 29 bytes, at time 0.010176954 second, from node 8 (both mac and ip addresses are 8), port 255 (routing agent).

Trace beginning:

```
s 0.029290548 _1_ RTR  --- 0 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
s 1.119926192 _0_ RTR  --- 1 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
M 10.00000 0 (5.00, 2.00, 0.00), (20.00, 18.00), 1.00
s 10.000000000 _0_ AGT  --- 2 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 10.000000000 _0_ RTR  --- 2 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 12.941172739 _1_ RTR  --- 3 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
s 13.000000000 _0_ AGT  --- 4 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 13.000000000 _0_ RTR  --- 4 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 13.242656084 _0_ RTR  --- 5 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
s 19.000000000 _0_ AGT  --- 6 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 19.000000000 _0_ RTR  --- 6 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 24.799296167 _1_ RTR  --- 7 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
s 27.719583723 _0_ RTR  --- 8 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
```

Using node-movement/traffic-pattern files

- Node movements for this example shall be read from a node-movement file called scen-3-test.
- scen-3-test defines random node movements for the 3 mobilenodes within a topology of 670mX670m.
- Provided by NS2 at:
  - /usr/local/ns2/ns-2.34/tcl/mobility/scene/scen-3-test

- Traffic pattern file
  - Provided by NS2 at:
  - /usr/local/ns2/ns-2.34/tcl/mobility/scene/cbr-3-test

```
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(x)            670   ;# X dimension of the topography
set val(y)            670   ;# Y dimension of the topography
set val(ifqlen)        50            ;# max packet in ifq
set val(seed)          0.0
set val(adhocRouting)   DSR
set val(nn)            3             ;# how many nodes are simulated
set val(cp)           "../mobility/scene/cbr-3-test"
set val(sc)           "../mobility/scene/scen-3-test"
set val(stop)          2000.0          ;# simulation time
```

- “Source” node-movement and connection pattern files

```
#
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)
```

Creating random traffic-pattern for wireless scenarios

- ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate]
  - Cbrgen.tcl is a traffic generator script to generate TCP or CBR traffic
  - 1/rate is the average interval time between CBR packets
  - Connections is the maximum # of connections
  - The start times for the TCP/CBR connections are randomly generated with a maximum value set at 180.0s
- Example: ns cbrgen.tcl -type cbr -nn 10 -seed 1.0 -mc 8 -rate 4.0 > cbr-10-test
  - create a CBR connection file between 10 nodes, having maximum of 8 connections, with a seed value of 1.0 and a rate of 4.0.

- Example: ns cbrgen.tcl -type tcp -nn 25 -seed 0.0 -mc 8 > tcp-25-test
  - Create a maximum 8 TCP connections (FTP traffic) between 25 nodes.

Creating node-movements for wireless scenarios

- Setdest is the program under ~ns/indep-utils/cmu-scen-gen/setdest
- ./setdest [-n num_of_nodes] [-p pausetime] [-M maxspeed] [-t simtime] \ [-x maxx] [-y maxy] > [outdir/movement-file]
- ./setdest -n <nodes> -s <speed type> -m <min speed> -M <max speed> -t <simulation time> -P <pause type> -p <pause time> -x <max X> -y <max Y> > [outdir/movement-file]

- Example: ./setdest -n 20 -p 2.0 -M 10.0 -t 200 -x 500 -y 500 > scen-20-test
  - an average pause between movement being 2s. Simulation stops after 200s and the topology boundary is defined as 500 X 500.

- Line in the file:
  - $ns_ at 2.000000000000 "$node_(0) setdest 90.441179033457 44.896095544010 1.373556960010”
    - node_(0) at time 2.0s starts to move toward destination (90.44, 44.89) at a speed of 1.37m/s.
  - $ns_ at 899.642 "$god_ set-dist 23 46 2”
    - shortest path between node 23 and node 46 changed to 2 hops at time 899.642.