Network Simulator (*NS*)

Prof. Nelson L. S. da Fonseca

State University of Campinas, Brazil

Outline

- Introduction to ns
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

Introduction

- NS began as a variant of the REAL network simulator in 1989. Currently ns development is supported through DARPA SAMAN project and NSF CONSER project.
  - contributions from other researchers.
- Can be installed on Unix, Linux and Windows.
- Download: http://www.isi.edu/nsnam/ns/

NS goals

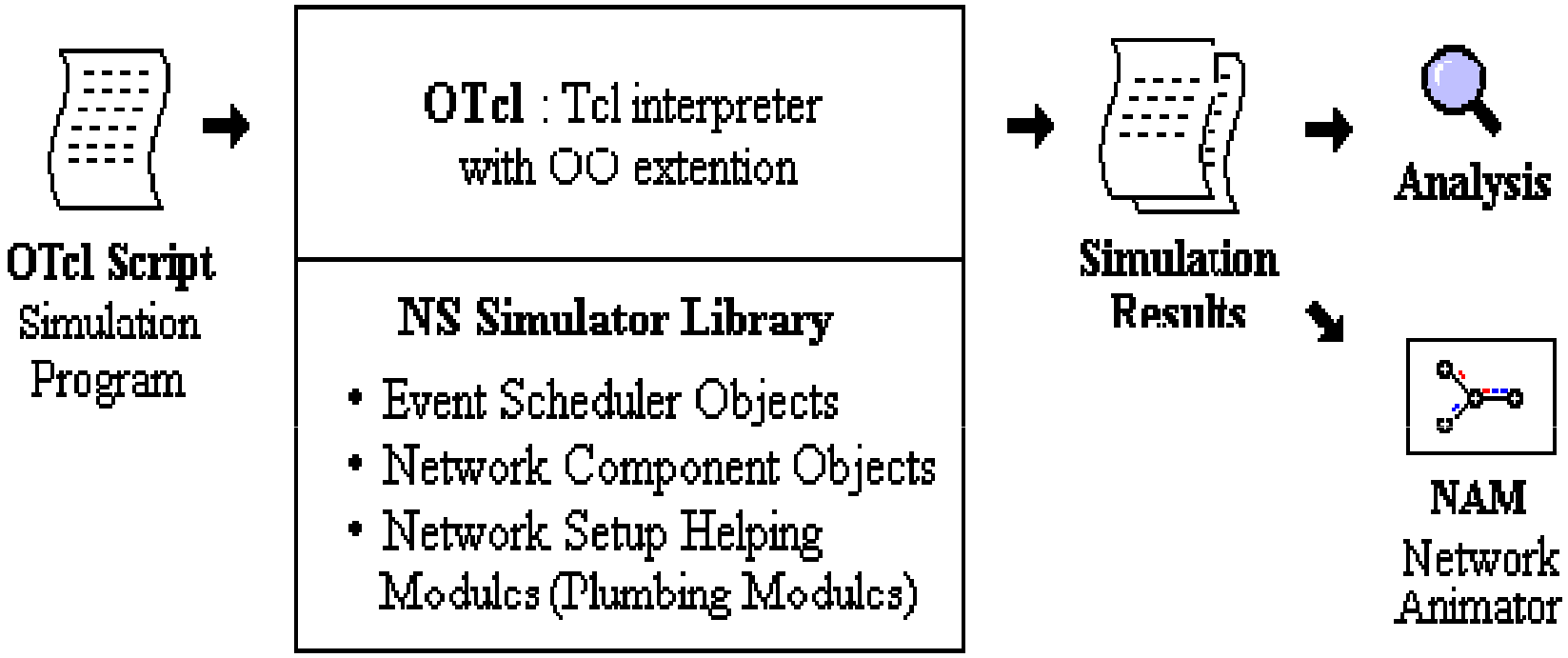
- Support networking research and education:
  - protocol design, protocol comparison, traffic studies, etc.
- Provide a collaborative environment
  - freely distributed, open source
    - share code, protocols, models, etc.
  - allow easy comparison of similar protocols
  - increase confidence in results
    - more people look at models
    - experts develop models
- Multiple levels of detail in one simulator.

NS Functionalities

- Wired networks:
  - Routing: DV, LS, PIM-SM
  - Transportation: TCP e UDP
  - Traffic sources: web, ftp, telnet, cbr, pareto on/off, etc.
  - Queueing disciplines: drop-tail, RED, FQ, SFQ, DRR
  - QoS: IntServ and DiffServ
- Wireless networks:
  - ad-hoc routing and mobile IP
  - MAC 802.*11 and Preamble based TDMA protocol*
- *Tracing* and visualization (nam)

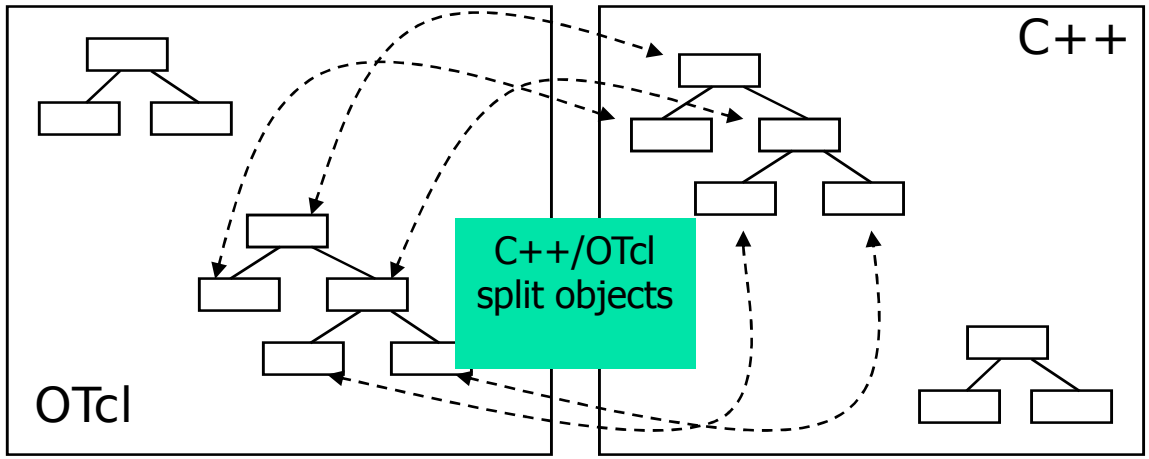
NS Structure

Simplified user's view of ns:



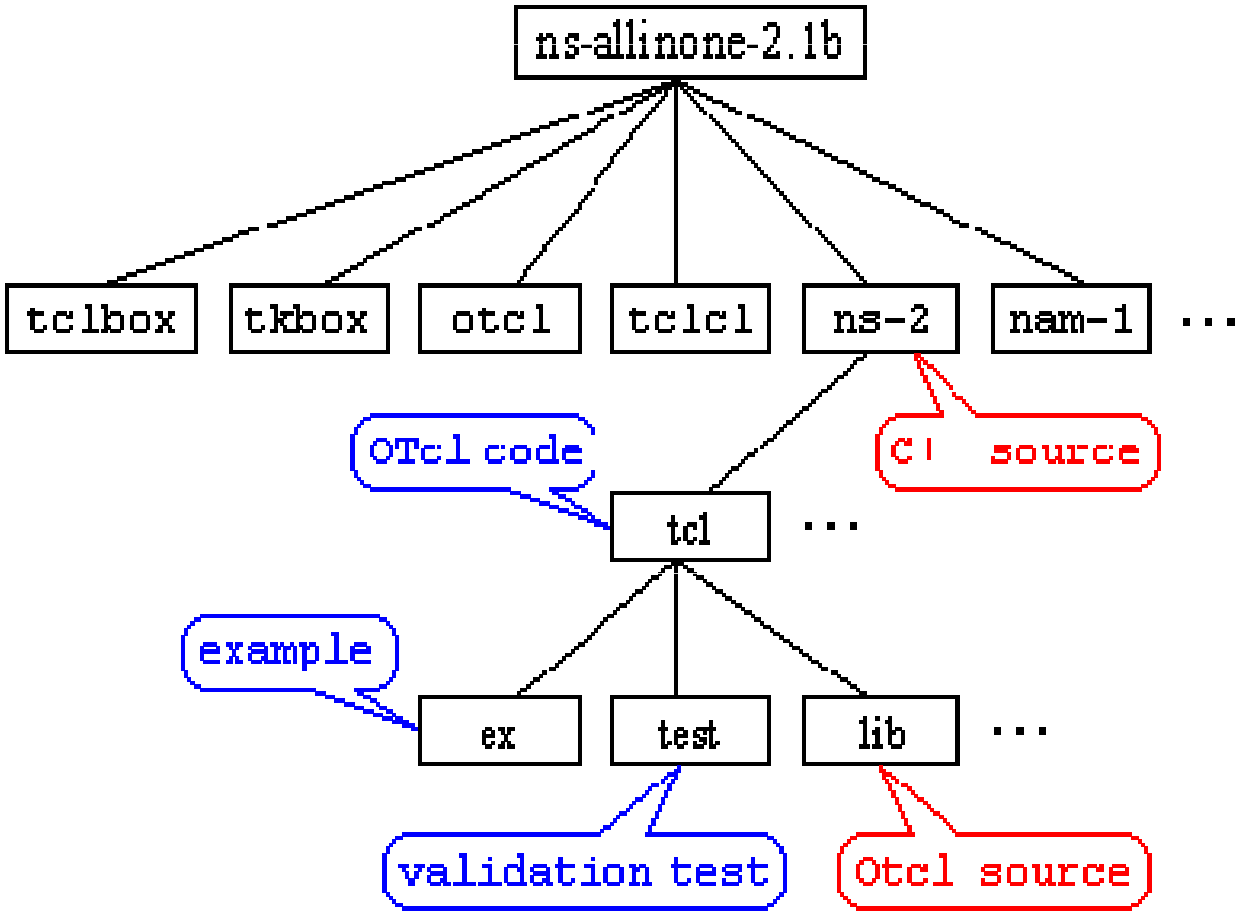
NS Structure

- OTcl (Object Tcl) and C++ share class hierarchy:



- C++ for "data":
  - per packet processing, core *ns*
  - fast to run, detailed, complete control
- OTcl for control:
  - simulation scenario configurations
  - Manipulating existing C++ objects
  - fast to write and change

NS Directory Structure



Discrete events simulation

- The scheduler is the core of the simulator.
- Event = an action to be done after a certain time in the future.
- Each event in NS is an object in the C++ hierarchy with:
  - an unique ID;
  - a scheduled time;
  - a pointer to the object that handles the event.

Discrete events simulation

- How does the scheduler work?
  - At the beginning of the simulation, the user schedules a certain number of events to be executed during the simulation lifetime, e.g., start of an application.
  - The objects of the simulation schedule other events.
  - All the events are placed in one queue by the order of their due time.
  - The scheduler dequeues the event at the head of this queue, advances the time of the simulation, executes the event, then dequeues another event, and so on, until the event "exit" is found.

Outline

- Introduction to *ns*
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

Programming in Tcl and OTcl

- Assign a value to a variable: set x 10
- Read the content of a variable: set y $x
- Arithmetic operations: set z [expr $x + $y]

  set x [expr 1/60] -> x = 0

  set x [expr 1.0/60.0] -> x = 0,01666...
- # this is a comment
- Open a file in mode write: set f [open filename w]
- Write the content of a variable in a file: puts $f "x = $x"
- Array: set tab($index) 5 ;# $index = 1, 2, 3, ...

Programming in Tcl and OTcl

- The structure of an **if** command:

```
if {expression} {
    <execute commands>
} else {
    <execute commands>
}
```

- Loops:

```
for {set i 0} {$i < 10} {incr i} {
    <execute commands>
}
```

Programming in Tcl and OTcl

- Procedures:

```
proc my_procedure {par1 par2} {
    global var1 var2  #global variables
    <commands>
    return $something
}
.....
#calling the procedure
my_procedure $x $y
```

Outline

- Introduction to ns
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

Inicialization

- create the event scheduler:

```
set ns [new Simulator]
```

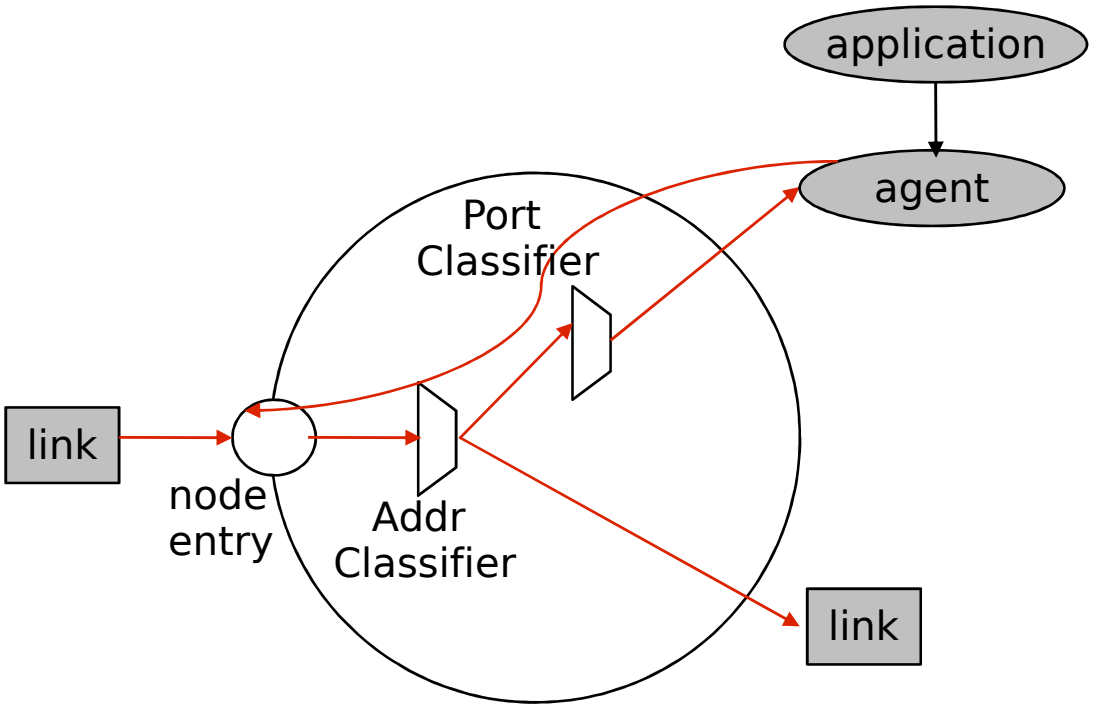
- create the output files:

```
#Open the trace file
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1   #trace all the events
#Open the NAM trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

Definition of nodes and links

- define the nodes:

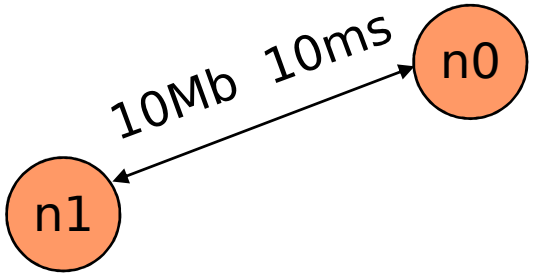
```
set n0 [$ns node]
set n1 [$ns node]
```



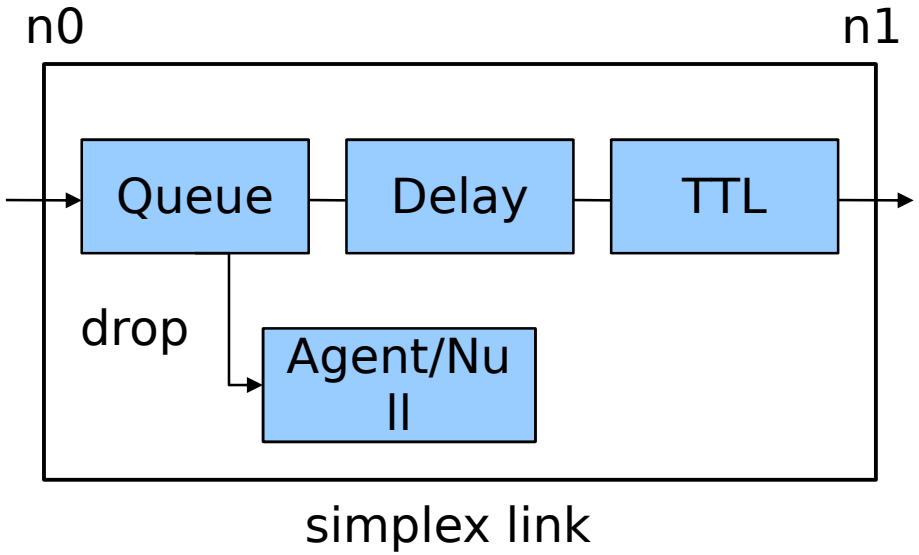
Unicast node structure

Definition of nodes and links

- define the links that connect the nodes:




```
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns queue-limit $n0 $n1 20
Queue set limit_ 50  #default value in the ns-default.tcl file
```



Agents and Applications

- FTP over TCP

```
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns connect $tcp $sink
$tcp set packetSize_ 552
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

Agents and Applications

- CBR over UDP

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set rate_ 100Kb     #$cbr set interval_ 0.01ms
$cbr set random_ 1
```

Agents and Applications

- exponential on-off traffic application

```
set source [new Application/Traffic/Exponential]
$source set packetSize_ 500
$source set burst_time_ 200ms
$source set idle_time_ 400ms
$source set rate_ 150Kb
```

Agents and Applications

- pareto on-off traffic application

```
set source [new Application/Traffic/Pareto]
$source set packetSize_ 500
$source set burst_time_ 200ms
$source set idle_time_ 400ms
$source set rate_ 150Kb
$source set shape_ 1.5
```

Agents and Applications

- trace driven application

  set tracefile [new Tracefile]

  $tracefile filename <file_name>

  set source [new Application/Traffic/Trace]

  $source attach-tracefile $tracefile

Scheduling events

```
$ns at 0.1 "$cbr start"
$ns at 0.5 "$ftp start"
$ns at 100.1 "$cbr stop"
$ns at 100.5 "$ftp stop"
$ns run
```

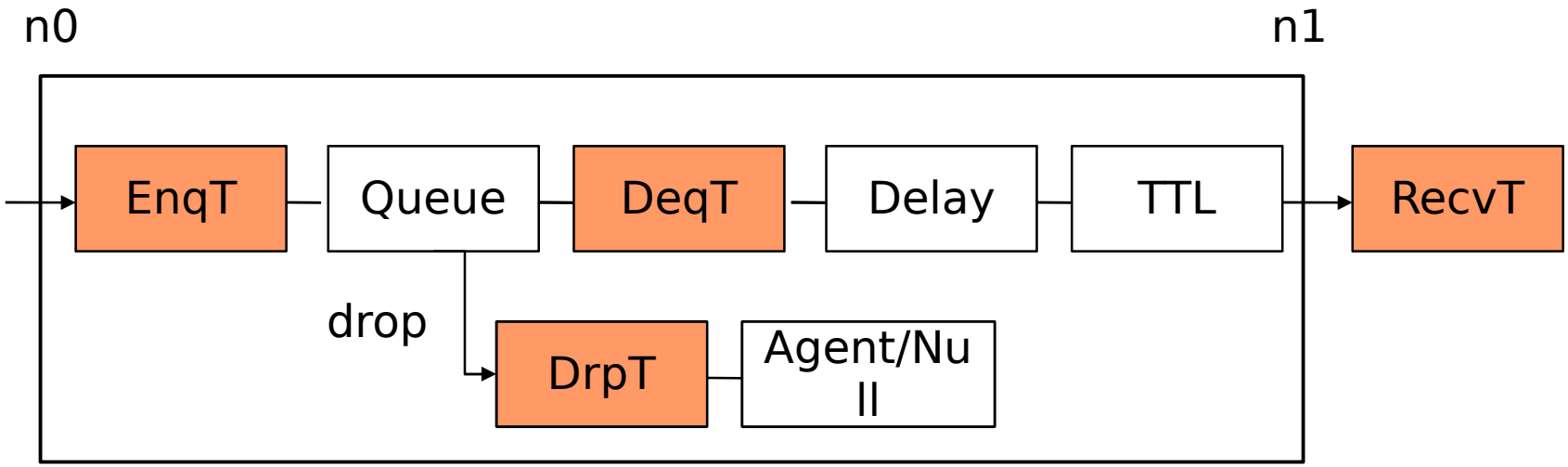
- Finally, we can run the simulation

```
ns my_script.tcl
nam out.nam  # executes the nam program for visualization
```

Outline

- Introduction to ns
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

Tracing



Tracing objects in a simplex link

- Trace all simulated events:

  $ns trace-all [open out.tr w]

- Trace events on one specific link:

  $ns trace-queue $n0 $n1 $tr

Tracing

Trace format:

```
Event time   from to pkt size  flags  fid src  dst seq pkt_id
  +  1.01016  2   3  tcp  40 -------   1  0.0  4.0  0    2
  -  1.01016  2   3  tcp  40 -------   1  0.0  4.0  0    2
  +  1.04066  3   5  cbr  1000 -------  2  1.0  5.0  1    1
  -  1.04066  3   5  cbr  1000 -------  2  1.0  5.0  1    1
  d  1.08666  3   5  cbr  1000 -------  2  1.0  5.0  1    1
  r  1.15186  2   3  tcp  40  -------  1  0.0  4.0  0    2
  +  1.25186  3   2  ack  40  -------  1  4.0  0.0  0    3
  -  1.25186  3   2  ack  40  -------  1  4.0  0.0  0    3
  r  1.29251  3   2  ack  40  -------  1  4.0  0.0  0    3
```

Tracing

- Queue monitor: get statistics about the motion of packets through a particular buffer of the simulated topology, e.g., number of packets (bytes) that have arrived to the buffer, number of packets (bytes) that have left, number of packets (bytes) that have been dropped, etc.

```
set monitor [$ns queue-monitor $n1 $n2 $file $sample_interval]

$monitor set pdepartures_

$monitor set barrivals_
```

Tracing

- Flow Monitor: get statistics about the motion of packets of a particular flow through a buffer of the topology. Define the Flow Monitor that filters packets based on their Flow ID, then associate it to the desired link:

```
set flowmon [$ns makeflowmon Fid]

set L [$ns link $n1 $n2]

$ns attach-fmon $L $flowmon

$ns at <time> “puts $flowmon set pdrops_”
```

Outline

- Introduction to ns
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

Processing trace files

- One can write programs in any programming language that can handle data files, e.g., C, awk, perl.

- ... or use existing network trace files analyser, e.g., Tracegraph:
  - 238 2D graphs
  - 12 3D graphs
  - delays, jitter, processing times, round trip times, throughput graphs and statistics
  - ...

Processing trace files

- **Using awk:** allows us to do simple operation on data files

```
BEGIN{n=0; sum=0}{
  n++;
  sum = sum + $2
}
END{print "average:" sum/n} out.tr > "average.txt"
```

- Using grep: allows us to filter a file

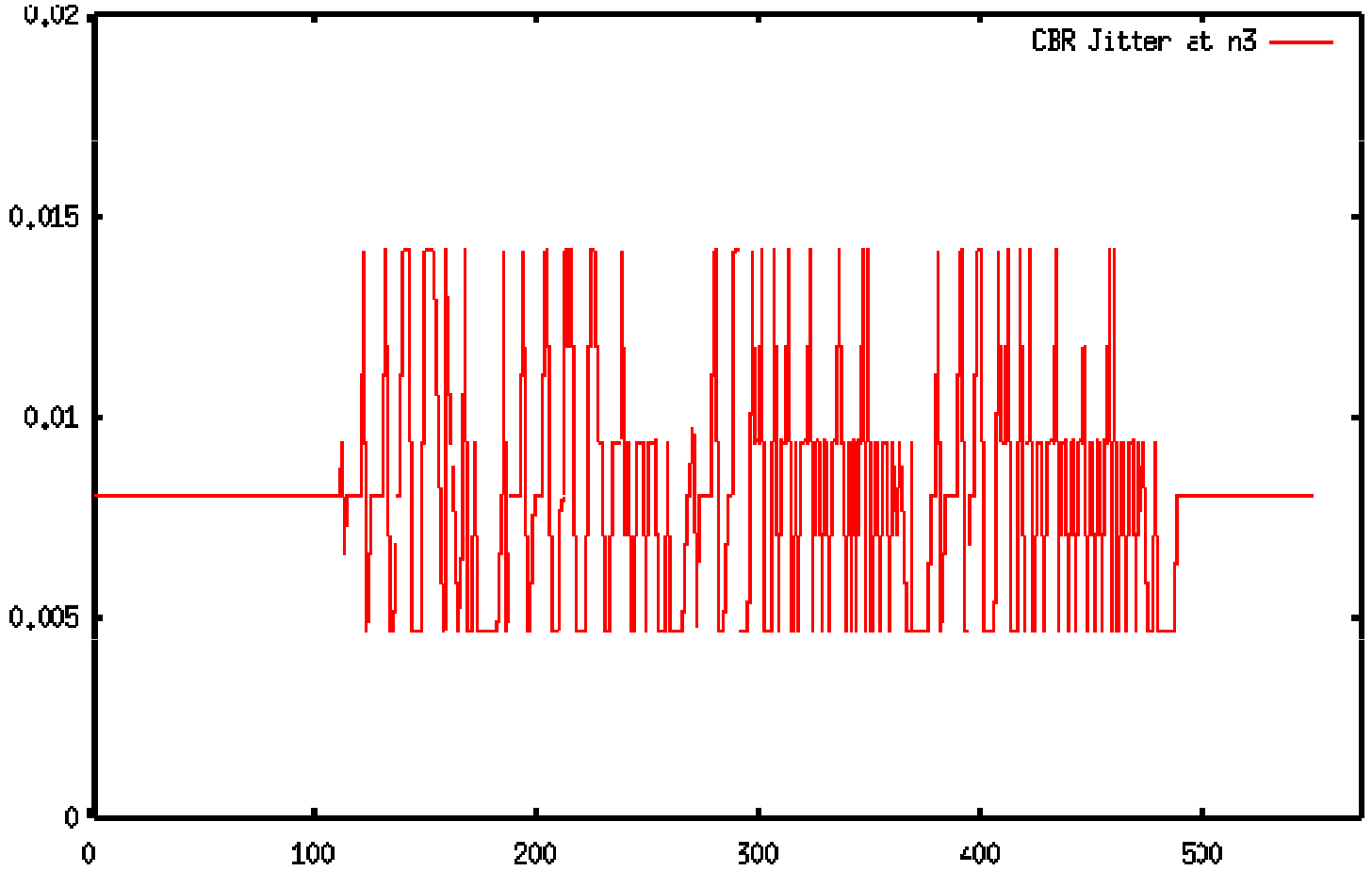
```
grep "0 2 cbr" out.tr > out02.tr
```

Processing trace files

- Script which calculates CBR traffic jitter at receiver node (n3) using data in "out.tr", and stores the resulting data in "jitter.txt".

```
cat out.tr | grep " 2 3 cbr " | grep ^r | column 1 10 > rec.txt
 awk '{
       dif = $2 - old2;
       if(dif==0) dif = 1;
       if(dif > 0) {
          print $2  ($1 - old1) / dif;
          old1 = $1;
          old2 = $2
       }
}' > jitter.txt
```

Processing trace files



CBR Jitter at The Receiving Node (n3)

Outline

- Introduction to ns
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

Random Number Generation

- A default RNG (defaultRNG) is created at simulator initialization time.
- It is not necessary to set a seed (the default is 12345).
- If you wish to change the seed:

```
# seed the default RNG
 global defaultRNG
 $defaultRNG seed 9999
```

Random Number Generation

- If multiple random variables are used in a simulation, each random variable should use a separate RNG object:

```
set r1 [new RNG]
set r2 [new RNG]
```

- When a new RNG object is created, it is automatically seeded to the beginning of the next independent stream of random numbers.

Random Variables

- The currently defined distributions, and their associated parameters are:
  - Pareto: avg_ shape_
  - Constant: val_
  - Uniform: min_ max_
  - Exponential: avg_
  - HiperExponential: avg_ cov_
  - Normal: avg_ std_
  - LogNormal: avg_ std_

Random Variables

- To create a random variable that generates number uniformly on [10, 20]:

```
set urv [new RandomVariable/Uniform]
$urv set min_ 10
$urv set max_ 20
$urv value
```

- By default, RandomVariable objects use the default RNG. The use-rng method can be used to associate a RandomVariable with a non-default RNG:

```
set r1 [new RNG]
$urv use-rng r1
```

Random Variables

- Example of random variables usage.
  - random distribution for the idle_time_ of an Exponential on-off application:

```
set source [new Application/Traffic/Exponential]

$source set packetSize_ 500

$source set burst_time_ 200ms

$source set idle_time_ [$urv value]

$source set rate_ 150Kb
```

Random Variables

- Example of random variables usage.
  - setting a random start time for an application:

```
$ns at [$urv value] "$cbr start"
```

Outline

- Introduction to ns
- Programming in TCL and OTCL
- A simple NS simulation script
- Tracing
- Processing trace files
- Random Number Generation and Random Variables
- Running Wireless Simulations in ns

A simple wireless scenario

- Start by creating an instance of the simulator:

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

- Set up trace support:

```
set tracefd  [open out.tr w]
$ns_ trace-all $tracefd
```

- Create a topology object that keeps track of movements of mobile nodes within the topological boundary:

```
set topo  [new Topography]
```

A simple wireless scenario

- Provide the topography object with x and y coordinates of the boundary, (x=500, y=500) :

```
$topo load_flatgrid 500 500
```

- Create the object God (General Operations Director) :

```
create-god 2          # 2 = number of mobile nodes in the network
```

  - God object stores the total number of mobile nodes and a table of shortest number of hops required to reach from one node to another.

A simple wireless scenario

- Define how a mobile node should be created:

```
$ns_ node-config -addressingType  flat            # or hierarchical or expanded
                 -adhocRouting   DSDV    # or DSR or TORA
                 -llType         LL
                 -macType        Mac/802_11
                 -propType       "Propagation/TwoRayGround"
                 -ifqType        "Queue/DropTail/PriQueue"
                 -ifqLen         50
                 -phyType        "Phy/WirelessPhy"
                 -antType        "Antenna/OmniAntenna"
                 -channelType    "Channel/WirelessChannel"
                 -topoInstance   $topo
                 -agentTrace     ON              # or OFF
                 -routerTrace    ON              # or OFF
                 -macTrace       ON              # or OFF
```

A simple wireless scenario

- Create the mobile nodes:

```
set n1 [$ns node]
$n1 random-motion 0        ;# disable random motion
set n2 [$ns node]
$n2 random-motion 0
```

Use "for loop" to create more nodes:

```
for {set i 0} {$i < 10 } {incr i} {
    set node_($i) [$ns_ node ]
}
```

A simple wireless scenario

- Now that we have created mobile nodes, we need to give them a position to start with:

```
# Provide initial (X,Y, Z=0) coordinates for n1 and n2
$n1 set X_ 5.0
$n1 set Y_ 2.0
$n1 set Z_ 0.0
$n2 set X_ 390.0
$n2 set Y_ 385.0
$n2 set Z_ 0.0
```

A simple wireless scenario

- Produce some node movements:

```
# n2 starts to move towards n1
$ns at 5.0 "$n2 setdest 25.0 20.0 15.0"
$ns at 10.0 "$n1 setdest 20.0 18.0 1.0"

# n2 then starts to move away from n1
$ns at 100.0 "$n2 setdest 490.0 480.0 15.0"
```

A simple wireless scenario

- Setup traffic flow between the two nodes:

```
# TCP connections between n1 and n2
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns_ attach-agent $n1 $tcp
$ns_ attach-agent $n2 $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"
```

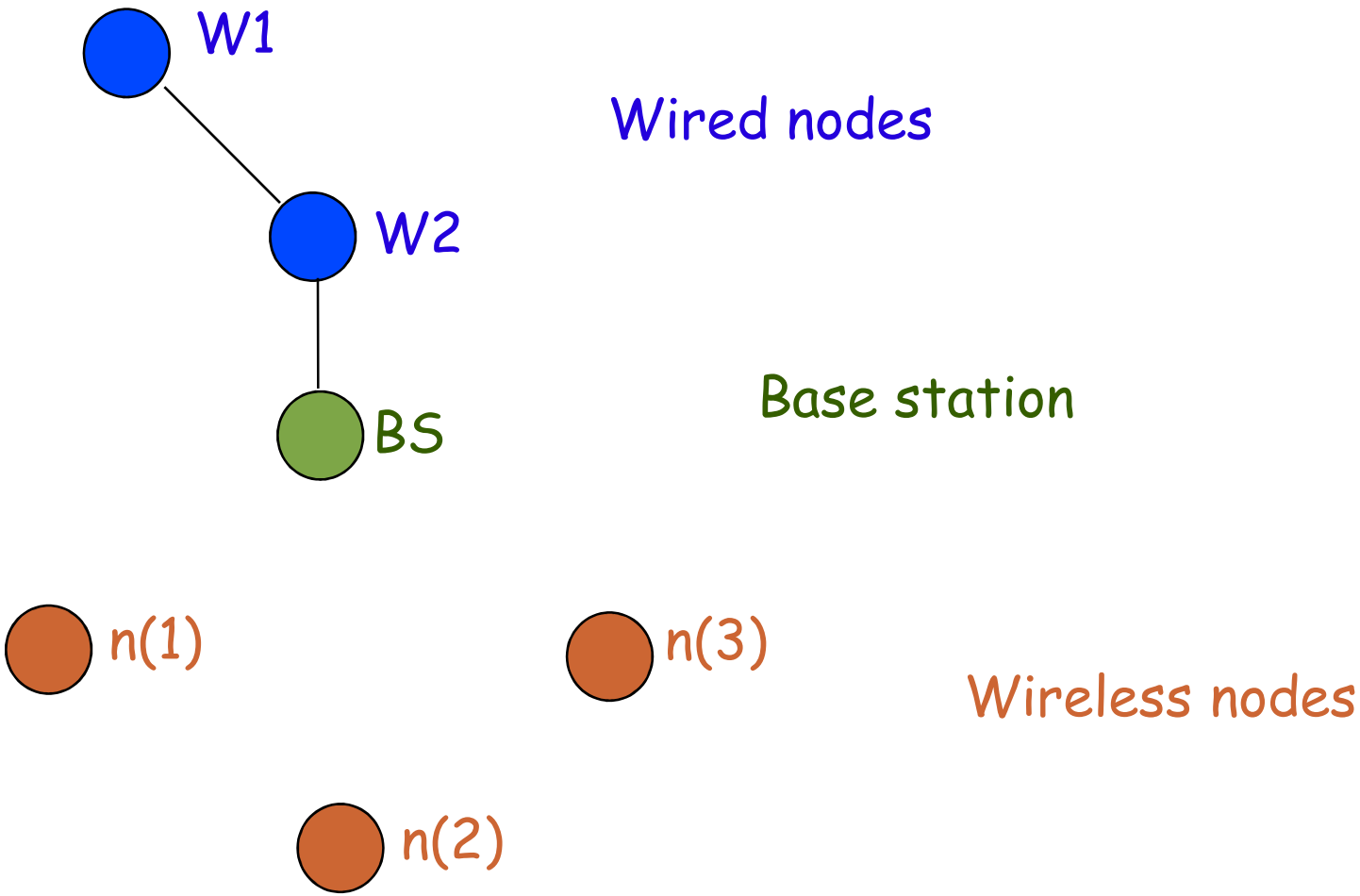
A simple wireless scenario

- Define stop time when the simulation ends and tell mobile nodes to reset which actually resets their internal network components:

```
$ns_ at 150.0 "$n1 reset"
$ns_ at 150.0 "$n2 reset"
$ns_ at 150.0001 "stop"
```

Wired-cum-wireless scenario

Topology for wired-cum-wireless simulation example:



Wired-cum-wireless scenario

- For mixed simulations we need to use hierarchical routing in order to route packets between wireless and wired domains:

```
set ns [new Simulator]
$ns_ node-config -addressType hierarchical
```

Wired-cum-wireless scenario

- Wired and wireless nodes are placed in different domains. Domains and sub-domains (clusters) are defined by means of hierarchical topology structure:

```
AddrParams set domain_num_ 2    ;# number of domains
lappend cluster_num 2 1         ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 4       ;# number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel   ;# for each domain
```

Wired-cum-wireless scenario

- Set up tracing for the simulation:

```
set tracefd  [open out.tr w]
$ns_ trace-all $tracefd
```

- Create the wired nodes:

```
set temp {0.0.0 0.1.0}          ;# hierarchical addresses to be used
set W1 [$ns_ node [lindex $temp 0]]
set W2 [$ns_ node [lindex $temp 1]]
```

Wired-cum-wireless scenario

- To create base station node, configure the node structure:

```
$ns_ node-config -adhocRouting   DSDV
                 -llType         LL
                 -macType        Mac/802_11
                 -propType       "Propagation/TwoRayGround"
                 -ifqType        "Queue/DropTail/PriQueue"
                 -ifqLen         50
                 -phyType        "Phy/WirelessPhy"
                 -antType        "Antenna/OmniAntenna"
                 -channelType    "Channel/WirelessChannel"
                 -topoInstance   $topo
                 -wiredRouting   ON
                 -agentTrace     ON          # or OFF
                 -routerTrace    ON          # or OFF
                 -macTrace       ON          # or OFF
```

Wired-cum-wireless scenario

- Create base station node:

```
set temp {1.0.0 1.0.1 1.0.2 1.0.3}   ;# hier address to be used for
                                      ;# wireless domain
set BS [ $ns_ node [lindex $temp 0]]
$BS random-motion 0                  ;# disable random motion
#provide some coordinates (fixed) to base station node
$BS set X_ 1.0
$BS set Y_ 2.0
$BS set Z_ 0.0
```

Wired-cum-wireless scenario

- Create wireless nodes:

```
#configure for mobilenodes
$ns_ node-config -wiredRouting OFF


# now create mobilenodes
for {set j 1} {$j < 4} {incr j} {
    set n($j) [ $ns_ node [lindex $temp $j] ]
    # provide each mobilenode with hier address of its base station
    $n($j) base-station [AddrParams addr2id [$BS node-addr]]
}
```

Wired-cum-wireless scenario

- Connect wired nodes and BS :

```
#create links between wired and BS nodes
$ns_ duplex-link $W1 $W2 5Mb 2ms DropTail
$ns_ duplex-link $W1 $BS 5Mb 2ms DropTail
```

Wired-cum-wireless scenario

- Set up TCP traffic between wireless node n(1) and wired node W1:

```
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns_ attach-agent $n(1) $tcp
$ns_ attach-agent $W1 $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 5.0 "$ftp start"
```

Wired-cum-wireless scenario

- Set up UDP traffic between wired node W2 and wireless node n(2):

```
set  udp [new Agent/UDP]
set null [new Agent/Null]
$ns_ attach-agent $W2 $udp
$ns_ attach-agent $n(2) $null
$ns_ connect $udp $null
set cbr [new Application/CBR]
$cbr attach-agent $udp
$ns_ at 10.0 "$cbr start"
```

Documentation

- NS official site : http://www.isi.edu/nsnam/ns/
- Manual: NS Notes and Documentation
- Tutorials: NS by Example, Marc Greis's tutorial
- "NS Simulator for Beginners", Eitan Altman and Tania Jiménez.
- Trace analyser: http://www.tracegraph.com/
- Tips: http://tagus.inesc-id.pt/~pestrela/ns2/