Internet Programming & Protocols Lecture 16

ns

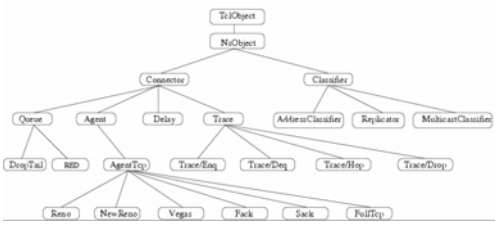
nam

www.cs.utk.edu/~dunigan/ipp/

ns

- Discrete event simulator
- Built on Tcl and C++
- Simulates following network components
  - Links
  - Routers
  - End-points
- Supports wired and wireless networks
- Many TCP flavors
- UDP
- Provides various tracing facilities
- nam, animated graphics

ns class hierarchy (partial)



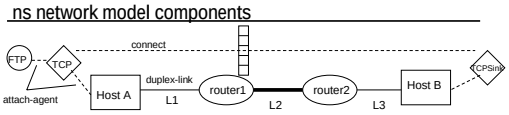
Network model in ns is constructed by interconnecting ns objects. Objects are built from a hierarchical C++ class structure. Basic methods are handle() to handle events and recv() to process packets.

Constructing a network model

- Other simulators (OPNET) have a nice GUI for drag & drop construction of nodes/links etc. … not ns ☹
- Network model is constructed with Tcl commands using the following ns objects
  - Node (host, router)
    - set n1 [$ns node]
  - Links used to connect nodes (bandwidth, delay, queue discipline)
    - $ns duplex-link $n1 $r1 8Mb 5ms Droptail
  - Agents – transport endpoints, attached to nodes
    - set udp_agent [new Agent/UDP]
    - set tcp1 [new Agent/TCP/Newreno]
  - Applications – data generators attached to transport agent
    - Traffic generators for UDP (CBR, Pareto, exponential)
    - FTP (infinite packet source) or Telnet for TCP
      - set ftp [new Application/FTP]

Agents
TCP (Tahoe)
TCP/Reno
TCP/Newreno
TCP/Sack1
TCP/Vegas
TCPSink
TCPSink/DelAck
TCPSink/Sack1

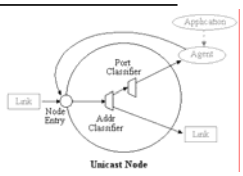
ns network model components



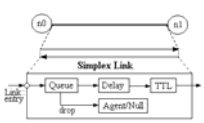
- Nodes: HostA router1 router2 Hostb
  - set hosta [$ns node]
  - set router1 [$ns node]
- Links: L1 L2 L3
  - $ns duplex-link $hosta $router1 10Mb 30ms DropTail
- Transport Agents: TCP TCPSink
  - set tcp [new Agent/TCP/Newreno]
  - $ns attach-agent $hosta $tcp
  - set sink [new Agent/TCP/Sink/DelAck]
  - $ns connect $tcp $sink
- Traffic generators: FTP
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp

ns nodes and links

ns node is a compound object composed of a node entry object and a classifiers.

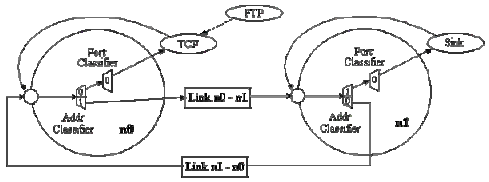


ns link (simplex or duplex) manages queuing, delay, and drops

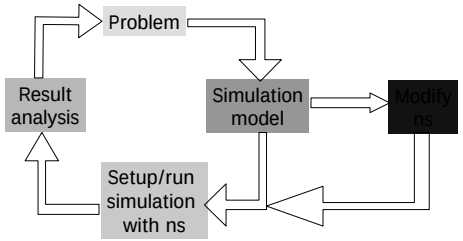


ns packet flow internals

- Full-duplex link between n0 and n1, TCP agent attached to n0, and FTP application attached to TCP agent. Sink agent attached to n1. TCP agent connected to Sink agent.



Using *ns*



ns script template

- For assignments, layout and document your Tcl as follows (see template.txt)

```
# ns tcl file should have header comments describing purpose and
#  any command line arguments

# intial values and command line arguments

# create ns simulator object and any trace files

# record and finish procedures

# topology: nodes, links  maybe ascii "picture" of topology

# transport agents, application agents and their settings

# schedule of events and run
```


ns by example

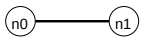
- Examples from tutorial (.tcl available)
- Intro to nam
- Examples from text chapter 4
- Tracing and monitoring and graphing
- Error loss models

Sample tcl's in ~dunigan/ipp05/ns/

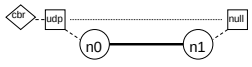
see README, you need to add things to your PATH and ENV

example1b.tcl

- **Two nodes,1 link**
  - set n0 [$ns node]
  - set n1 [$ns node]
  - $ns duplex-link $n0 $n1 1Mb 10ms DropTail
    - Delay is one-way (so RTT is 20 ms in this case)



- **Create a UDP agent and attach it to node**
  - n0 set udp0 [new Agent/UDP]
  - $ns attach-agent $n0 $udp0
- **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
- **create a Null agent which acts as traffic sink and attach it to node n1**
  - set null0 [new Agent/Null]
  - $ns attach-agent $n1 $null0
- **Connect the two agents to each other.**
  - $ns connect $udp0 $null0



example1b.tcl

- Wrap this topology with boilerplate stuff
- Set up initial values if any, create simulator object, and open trace files

```
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
```

- Set up a finish procedure

```
proc finish {} {
        global ns nf
        $ns flush-trace
   #Close the trace file
        close $nf
   #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}
```

- At the end schedule events, and start 'er up (ns example1b.tcl)

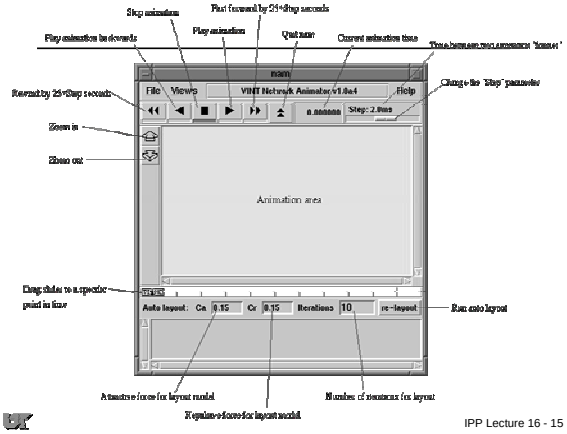
```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
$ns run
```

example1b.tcl

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
   #Close the trace file
        close $nf
   #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}
#Create two nodes
set n0 [$ns node]
set n1 [$ns node]

#Create a duplex link
$ns duplex-link $n0 $n1 1Mb 10ms
    DropTail

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

# 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

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

#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"
#Call the finish procedure after 5
    seconds of simulation time
$ns at 5.0 "finish"
$ns run
```


Running ns and nam

- ns example1b.tcl
  - Runs simulation
  - Creates a nam tracefile (out.nam)
  - Runs nam (nam out.nam) (Example: nam ex1b.nam)
- nam Network Animator
  - Mainly eye-candy, sometimes it gives some insight
  - Tcl/TK animation tool for packet animation
  - Embed Tcl commands in your ns script to control animation ☹
  - Ns produces a nam trace file
  - Use nam to view the animation



Tcl for managing nam

- The nam control commands have nothing to do with the simulation problem, but they are embedded in your .tcl (messy)
- Visualize trace in nam
  - Can collect trace on whole simulation or just one path

  $ns namtrace-all [open test.nam w]

  $ns namtrace-queue $n0 $n1
- Annotation
  - Add textual explanation to your simulation (appears in lower box)

  `$ns at 3.5 "$ns trace-annotate \"packet drop\""`
- Variable tracing in nam

  Agent/TCP set nam_tracevar_ true

  $tcp tracevar srtt_

  $tcp tracevar cwnd_
  - The changing value of these will appear in the lower nam window, (Example: nam var.nam)

ns→nam Interface

- Color
- Node manipulation
- Link manipulation
- Topology layout
- Protocol state
- Misc

nam Interface: Color

- Color mapping

  `$ns color 40 red`

  `$ns color 41 blue`

  `$ns color 42 chocolate`
- Color ↔ flow id association

  `$tcp0 set fid_ 40 ;# red packets`

  `$tcp1 set fid_ 41 ;# blue packets`

  `$udp0 set fid_ 42 ;# chocolate packets`

nam Interface: Nodes

- Color
  $node color red
- Shape (can't be changed after sim starts)
  $node shape box ;# circle, box, hexagon
- Marks (concentric "shapes")
  $ns at 1.0 "$n0 add-mark m0 blue box"
  $ns at 2.0 "$n0 delete-mark m0"
- Label (single string)
  $ns at 1.1 "$n0 label \"web cache 0\""

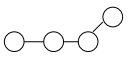
nam Interfaces: Links

- Color
  $ns duplex-link-op $n0 $n1 color "green"
- Label
  $ns duplex-link-op $n0 $n1 label "abced"
- Dynamics (automatically handled)
  $ns rtmodel Deterministic {2.0 0.9 0.1} $n0 $n1
- Asymmetric links not allowed

nam Interface: Topo Layout

- "Manual" layout: specify everything

```
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(1) $n(2) orient right
$ns duplex-link-op $n(2) $n(3) orient right
$ns duplex-link-op $n(3) $n(4) orient 60deg
```



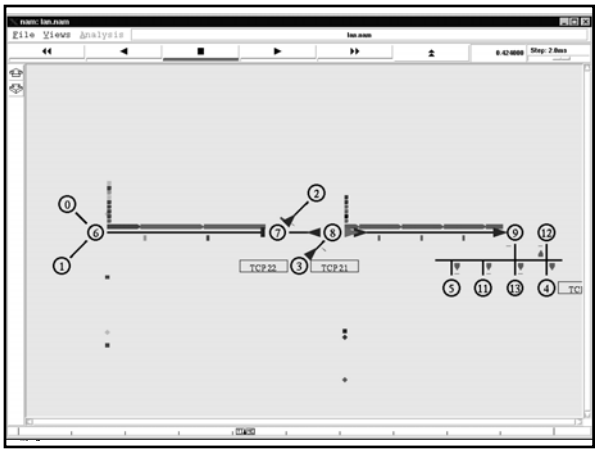
- If anything missing → automatic layout
  - Use the Edit button on nam to re-arrange

nam Interface: Misc

- Monitor a queue
  - $ns duplex-link-op $n2 $n3 queuePos 0.5
- Set animation rate

```
$ns at 0.0 "$ns set-animation-rate 0.1ms"
```

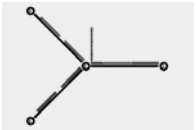
You won't be using nam that much, unless you just want to, but you need to recognize (disregard) these commands in ns scripts that you might encounter.



example2.tcl

- 3 nodes, 2 UDP/CBR sources (n0 n1) to 2 sinks at n3

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
# 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
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
# Create a CBR traffic source and attach it to udp1
…ditto for cbr1
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

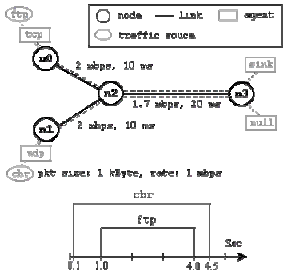


example nam ex2.nam

ns by example ns-simple.tcl

- TCP competing with UDP
- Four nodes, 3 links
- TCP agent (Tahoe) and TCPSink
  - set tcp [new Agent/TCP]
  - $ns attach-agent $n0 $tcp
  - set sink [new Agent/TCPSink]
  - $ns attach-agent $n3 $sink
  - $ns connect $tcp $sink
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp
- UDP agent with CBR
- Limit n2-n3 queue size to 10
  - $ns queue-limit $n2 $n3 10
- generate nam trace simple.nam
- Assignment 7

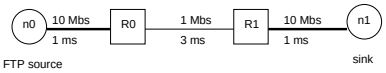
http://nile.wpi.edu/NS/



Example chap. 4.1 chap4-1.tcl

- TCP with varying window size
- metric: TCP throughput
  - goodput – unique data arriving at destination
  - Reported by finish procedure along with count of retransmitted packets
- generate a packet trace
  - $ns trace-all [open out.all w]
- record procedure to write current time and data rate to a trace file
- 4 nodes

```
#   object       from  to  bandwith   delay    queue
$ns duplex-link $n0   $R1 10Mb         1ms    DropTail
$ns duplex-link $R1   $R2  1Mb         3ms    DropTail
$ns duplex-link $R2   $n1 10Mb         1ms    DropTail
```



TCP agents and public variables

ns-2.28/tcl/lib/ns-default.tcl

- Agents
  - Sender: **TCP TCP/Reno TCP/NewReno TCP/Sack1 TCP/Fack**
    - Plus **TCP/WestwoodNR** and other trickier variations (later)
  - Receiver: **TCPSink TCPSink/DelAck TCPSink/Sack1 TCPSink/Sack1/DelAck**
  - Note **Agent/TCP/FullTcp** is bidirectional, different semantics (not covered)
- Variables for controlling TCP operation
  - Defaults in **ns-2.28/tcl/lib/ns-default.tcl**
  - **packetSize_ window_ tcpTick_ numdupacks_ decrease_num_**
  - Set for all TCP flows in your tcl **Agent/TCP tcpTick_ 0.5**
  - Set for paritcular flow **$tcp0 set window_ 20**
- Variables for tracing TCP behavior
  - sender side: **ndatabytes_ ndatapack_ nrexmitpack_ cwnd_ ssthresh_ dupacks_ srtt_** (AIMD: **increase_num_ decrease_num_**)
  - sink side: **bytes_**
- (later) monitor-queue: pdrops_ size_

finish

- Report final values of metrics of interest
- Schedule to run at end of simulation

  $ns at 10.0 "finish"

```
proc finish {} {
    global ns tcp0 tcpsink0 f2 window

        set now [$ns now]
        set bw [$tcpsink0 set bytes_]
        puts "window $window pkts  bw [expr $bw/$now*8/1000] Kbs rexmit
   packets: [$tcp0 set nrexmitpack_]"
        $ns flush-trace
        close $f2
 #  do other things like awk trace file, start nam etc.
         exit 0
}
```


record

- Your own monitoring procedure
  - Write variables of interest at this instant to files
  - Re-schedule record procedure for next delta
- Open any files you'll need in the initialization and schedule record

  set f2 [open thruput.dat w]

  $ns at 0.0 "record"

```
set mybytes 0;  # global for holding value across record calls

proc record {} {
        global ns tcp0 tcpsink0 f2 mybytes

        set delta 0.1
        set now [$ns now]
        set tombytes [$tcp0 set ndatabytes_ ]
        set bw [expr $tombytes - $mybytes]
        puts $f2 "$now [expr $bw/$delta*8/1000000]"
        set mybytes $tombytes

        #Re-schedule the procedure
        $ns at [expr $now+$delta] "record"
}
```

Don't forget

Trace file

- Trace packets on all links

  $ns trace-all [open test.out w]
- For every packet, record for entry and exit of each node on the path
  - This file can get really big (you're seeing the event scheduler in action)
  - Restrict to a particular link: $ns trace-queue $n0 $n1

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)                    src_addr : node.port (3.0)
+ : enqueue (at queue)
- : dequeue (at queue)                      dst_addr : node.port (0.0)
d : drop    (at queue)

        r 1.3838 3 2 ack 40 ------- 1 3.0 0.0 25 201
        + 1.3838 2 0 ack 40 ------- 1 3.0 0.0 25 201
        - 1.3838 2 0 ack 40 ------- 1 3.0 0.0 25 201
        r 1.38576 0 2 tcp 1000 ------- 1 0.0 3.0 25 199
        + 1.38576 2 3 tcp 1000 ------- 1 0.0 3.0 25 199
        d 1.38576 2 3 tcp 1000 ------- 1 0.0 3.0 25 199
        + 1.384 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
        - 1.384 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Life of packet and its ACK

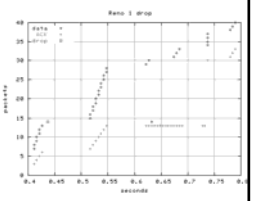
- Can see how much time packet spends at each node
  - Link delay, queue delay etc.
  - Can track RTT with seq #
  - Need flow id for complicated flows $tcp0 set fid_1; $tcp1 set fid_ 2

```
+ 9.877424 0 1 tcp 1040 ------- 0 0.0 3.0 1189 2374
- 9.877424 0 1 tcp 1040 ------- 0 0.0 3.0 1189 2374
r 9.879256 0 1 tcp 1040 ------- 0 0.0 3.0 1189 2374
+ 9.879256 1 2 tcp 1040 ------- 0 0.0 3.0 1189 2374
- 9.900488 1 2 tcp 1040 ------- 0 0.0 3.0 1189 2374
r 9.911808 1 2 tcp 1040 ------- 0 0.0 3.0 1189 2374
+ 9.911808 2 3 tcp 1040 ------- 0 0.0 3.0 1189 2374
- 9.911808 2 3 tcp 1040 ------- 0 0.0 3.0 1189 2374
r 9.91364 2 3 tcp 1040 ------- 0 0.0 3.0 1189 2374
+ 9.91364 3 2 ack 40 ------- 0 3.0 0.0 1189 2383
- 9.91364 3 2 ack 40 ------- 0 3.0 0.0 1189 2383
r 9.914672 3 2 ack 40 ------- 0 3.0 0.0 1189 2383
+ 9.914672 2 1 ack 40 ------- 0 3.0 0.0 1189 2383
- 9.914672 2 1 ack 40 ------- 0 3.0 0.0 1189 2383
r 9.917992 2 1 ack 40 ------- 0 3.0 0.0 1189 2383
+ 9.917992 1 0 ack 40 ------- 0 3.0 0.0 1189 2383
- 9.917992 1 0 ack 40 ------- 0 3.0 0.0 1189 2383
r 9.919024 1 0 ack 40 ------- 0 3.0 0.0 1189 2383
```


awk's for trace file ack seq drop

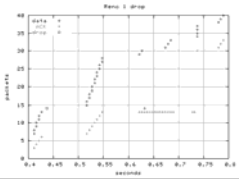
- Parse out.all, ?check node # maybe fid

```
exec awk {
        {
   if (($1 == "r") && ($5 == "ack") &&\
       ($3 == "1") && ($4 == "0"))\
              print $2, $11
        }
      } out.all > out.ack
 exec awk {
              {
    if (($1 == "+") && ($5 == "tcp") &&\
        ($3 == "0") && ($4 == "1"))\
            print $2, $11
              }
      } out.all > out.seq

exec awk {
              {
     if (($1 == "d") && ($5 == "tcp") &&\
         ($3 == "0") && ($4 == "1"))\
              print $2, $11
              }
      } out.all > out.drop
```



graphing

- Graphs more useful than nam, e.g., plot
  - bandwidth vs time
  - cwnd vs time
  - ssthresh vs time
- Plot with xgraph (in your "ns" path)
  - xgraph –m file1.dat file2.dat
  - xgraph –m –nl out.ack out.seq out.drop
- Plot with gnuplot (create .png) see ~dunigan/ipp05/ns/sample.plot



ns

- With ns you can set up a multitude of experiments
  - Vary topology
    - Number of nodes, delay (RTT), bandwdith, queue sizes
  - Vary TCP's
    - Different flavors: Tahoe Reno Newreno Sack Fack …
    - Vary window size, dup threshold, tick resolution
  - Mix in different competing traffic
    - Other TCP flows
    - UDP CBR/exponential/Pareto
  - Trace/monitor variables
    - Plot cwnd, ssthresh, datarate, RTT
    - Plot sequence number, ACK, drops
- How well does the simulation match the real world?

Next time …

- ns error models
- More TCP flavors