



Part I: Fundamentals (con't)

Outline

- ◆ Goals
- ◆ Essentials
- ◆ Primers
 - Wired world
 - **Wireless world**
 - Emulator
 - Utilities

ns Primer – Wireless World

- ◆ Ad hoc routing
- ◆ Mobile IP
- ◆ Satellite networking

Ad Hoc Routing – An Example

◆ Scenario

- 3 mobile nodes
 - moving within 670mX670m flat topology
 - using DSDV ad hoc routing protocol
 - Random Waypoint mobility model
 - TCP and CBR traffic
- ◆ *ns-2/tcl/ex/wireless-demo-csci694.tcl*

An Example – Step 1

```
# Define Global Variables
# create simulator
set ns [new Simulator]

# create a topology in a 670m x 670m area
set topo [new Topography]
$topo load_flatgrid 670 670
```

An Example – Step 2

```
# Define standard ns/nam trace
```

```
# ns trace
```

```
set tracefd [open demo.tr w]
```

```
$ns trace-all $tracefd
```

```
# nam trace
```

```
set namtrace [open demo.nam w]
```

```
$ns namtrace-all-wireless $namtrace 670 670
```

An Example – Step 3

```
# Create God  
set god [create-god 3]  
$ns at 900.00 "$god setdist 2 3 1"
```

- ◆ **God**: store an array of the smallest number of hops required to reach one node to an other
- ◆ Optimal case against which to compare routing protocol performance
- ◆ Automatically generated by scenario file

An Example – Step 4

```
# Define how a mobile node should be created
$ns node-config \
  -adhocRouting DSDV \
  -llType LL \
  -macType Mac/802_11 \
  -ifqLen 50 \
  -ifqType Queue/DropTail/PriQueue \
  -antType Antenna/OmniAntenna \
  -propType Propagation/TwoRayGround \
  -phyType Phy/WirelessPhy \
  -channelType Channel/WirelessChannel \
  -topoInstance $topo
  -agentTrace ON \
  -routerTrace OFF \
  -macTrace OFF
```


An Example – Step 5

```
# Create a mobile node, attach it to the channel
```

```
set node(0) [$ns node]
```

```
# disable random motion
```

```
$node(0) random-motion 0
```

◆ Use “for” loop to create 3 nodes:

```
for {set i 0} {$i < 3} {incr i} {  
    set node($i) [$ns node]  
}
```

An Example – Step 6

```
# Define node movement model  
source movement-scenario-files
```

```
# Define traffic model  
source traffic-scenario-files
```

Scenario Generator: Movement

◆ Mobile Movement Generator

```
setdest -n <num_of_nodes> -p  
pausetime -s <maxspeed> -t  
<simtime> -x <maxx> -y <maxy>
```

◆ Random movement

- \$node start
- Source: `ns-2/indep-utils/cmu-scen-gen/setdest/`

A Movement File

```
$node_(2) set Z_ 0.000000000000  
$node_(2) set Y_ 199.373306816804  
$node_(2) set X_ 591.256560093833  
$node_(1) set Z_ 0.000000000000  
$node_(1) set Y_ 345.357731779204  
$node_(1) set X_ 257.046298323157  
$node_(0) set Z_ 0.000000000000  
$node_(0) set Y_ 239.438009831261  
$node_(0) set X_ 83.364418416244  
$ns_at 50.000000000000 "$node_(2) setdest 369.463244915743  
170.519203111152 3.371785899154"  
$ns_at 51.000000000000 "$node_(1) setdest 221.826585497093  
80.855495003839 14.909259208114"  
$ns_at 33.000000000000 "$node_(0) setdest 89.663708107313  
283.494644426442 19.153832288917"
```

Scenario Generator: Traffic

◆ Generating traffic pattern files

■ CBR traffic

```
ns cbrgen.tcl [-type cbf|tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate]
```

■ TCP traffic

```
ns tcpgen.tcl [-nn nodes] [-seed seed]
```

■ Source: *ns-2/indep-utils/cmu-scen-gen/*

A Traffic Scenario

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(0) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 4.0
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 127.93667922166023 "$cbr_(0) start"
.....
```

An Example – Step 7

```
# Define node initial position in nam  
for {set i 0} {$i < 3} {incr i} {  
    $ns initial_node_position $node($i) 20  
}
```

```
# Tell ns/nam the simulation stop time  
$ns at 200.0 "$ns nam-end-wireless 200.0"  
$ns at 200.0 "$ns halt"
```

```
# Start your simulation  
$ns run
```

Energy Extension

- ◆ Node is energy-aware
- ◆ Define node by adding new options:

```
$ns_ node-config \  
-energyModel EnergyModel  
-initialEnergy 100.0  
-txPower      0.6  
-rxPower      0.2
```


nam Visualization

- ◆ Use nam to visualize:
 - Mobile node position
 - Mobile node moving direction and speed
 - Energy consumption at nodes (color keyed)

nam Visualization

- ◆ Replace

```
$ns namtrace-all $fd
```

with

```
$ns namtrace-all-wireless $fd
```

At the end of simulation, do

```
$ns nam-end-wireless [$ns now]
```

- ◆ See an example:



Summary

- ◆ Mac Layer: IEEE 802.11
- ◆ Address Resolution Protocol (ARP)
- ◆ Ad hoc routing protocols: DSDV, DSR, TORA, AODV
- ◆ Radio Propagation Model
 - Friss-space attenuation at near distances
 - Two ray ground at far distances
- ◆ Antenna: an omni-directional antenna having unity gain

Summary

- ◆ Energy consumption model for sensor networks
- ◆ Visualization of node movement, reachability, and energy
- ◆ Validation test suites

Credit

- ◆ CMU
- ◆ UC Berkeley
- ◆ Sun Microsystem Inc.
- ◆ USC/ISI

A Brief on Satellite Networking

- ◆ Developed by Tom Henderson (UCB)
- ◆ Supported models
 - Geostationary satellites: bent-pipe and processing-payload
 - Low-Earth-Orbit satellites
- ◆ Example: `tcl/ex/sat-*.tcl`
- ◆ Much in-development

A Brief on MobileIP Support

- ◆ Developed by Sun

- Require a different Node structure than the MobileNode
- Co-exists with wired world in ns

- ◆ Standard MobileIP

- Home Agent, Foreign Agent, MobileHosts...

- ◆ Example

- `~ns/tcl/ex/wired-cum-wireless.tcl`

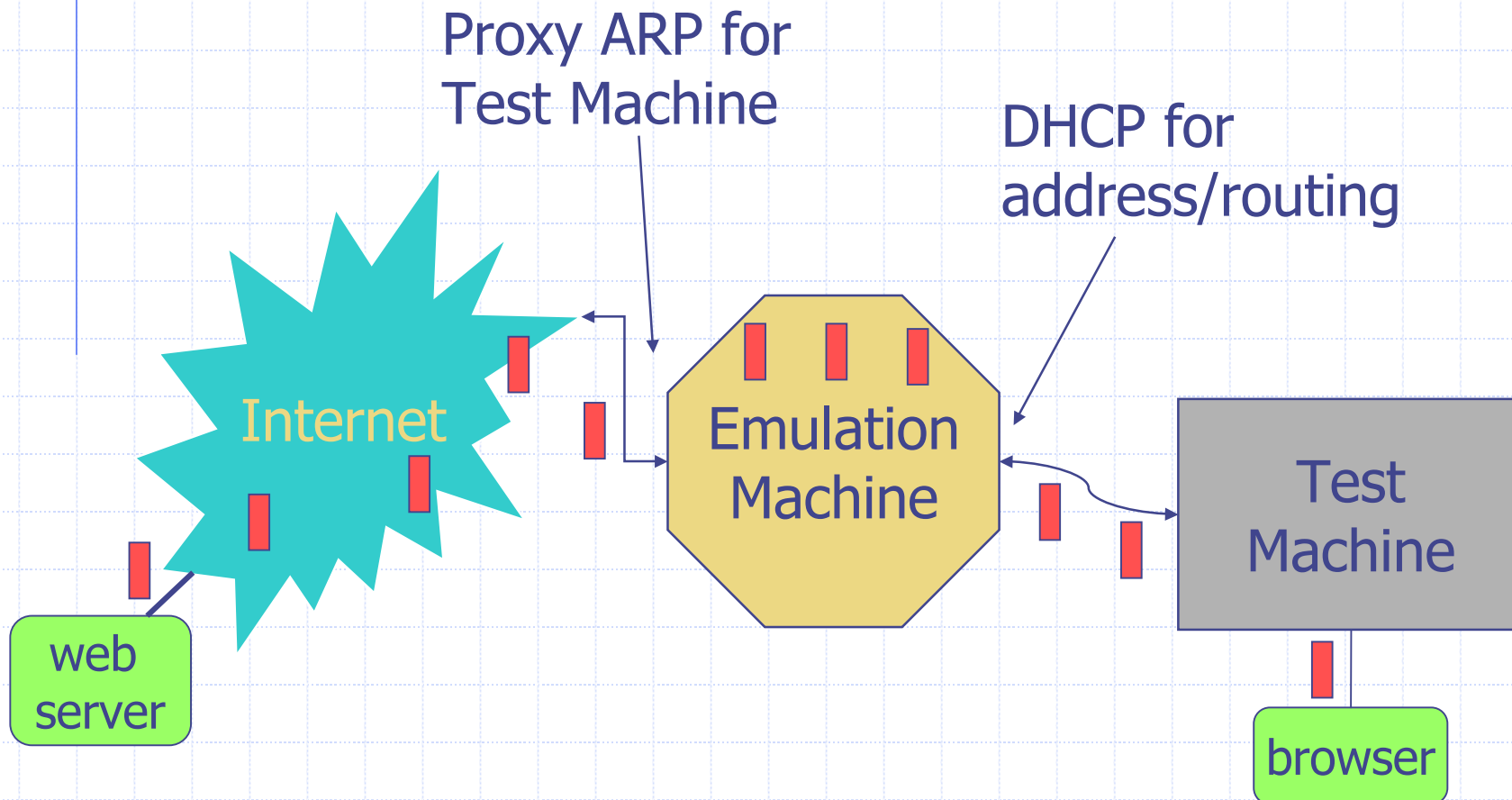
Outline

- ◆ Goals
- ◆ Essentials
- ◆ Primers
 - Wired world
 - Wireless world
 - Emulator
 - Utilities

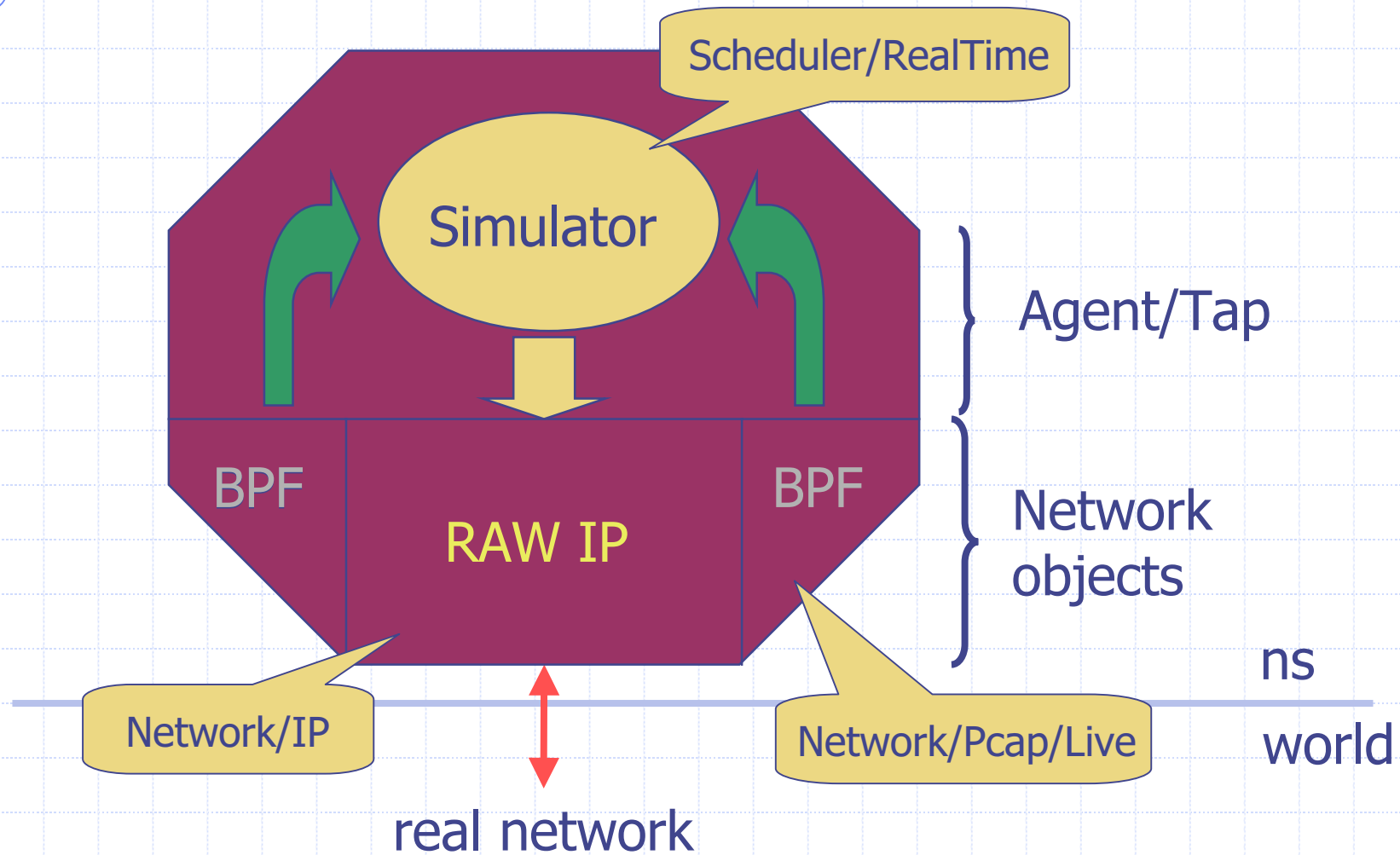
Emulation in ns

- ◆ Simulator ↔ real network
 - Inject received packets into simulation
 - Emit packets on to live network
- ◆ Usage
 - Subject real implementations to controlled conditions in the simulator
 - Subject simulations to real-world traffic
- ◆ **Currently only works on FreeBSD**

Sample Environment



Emulation Machine



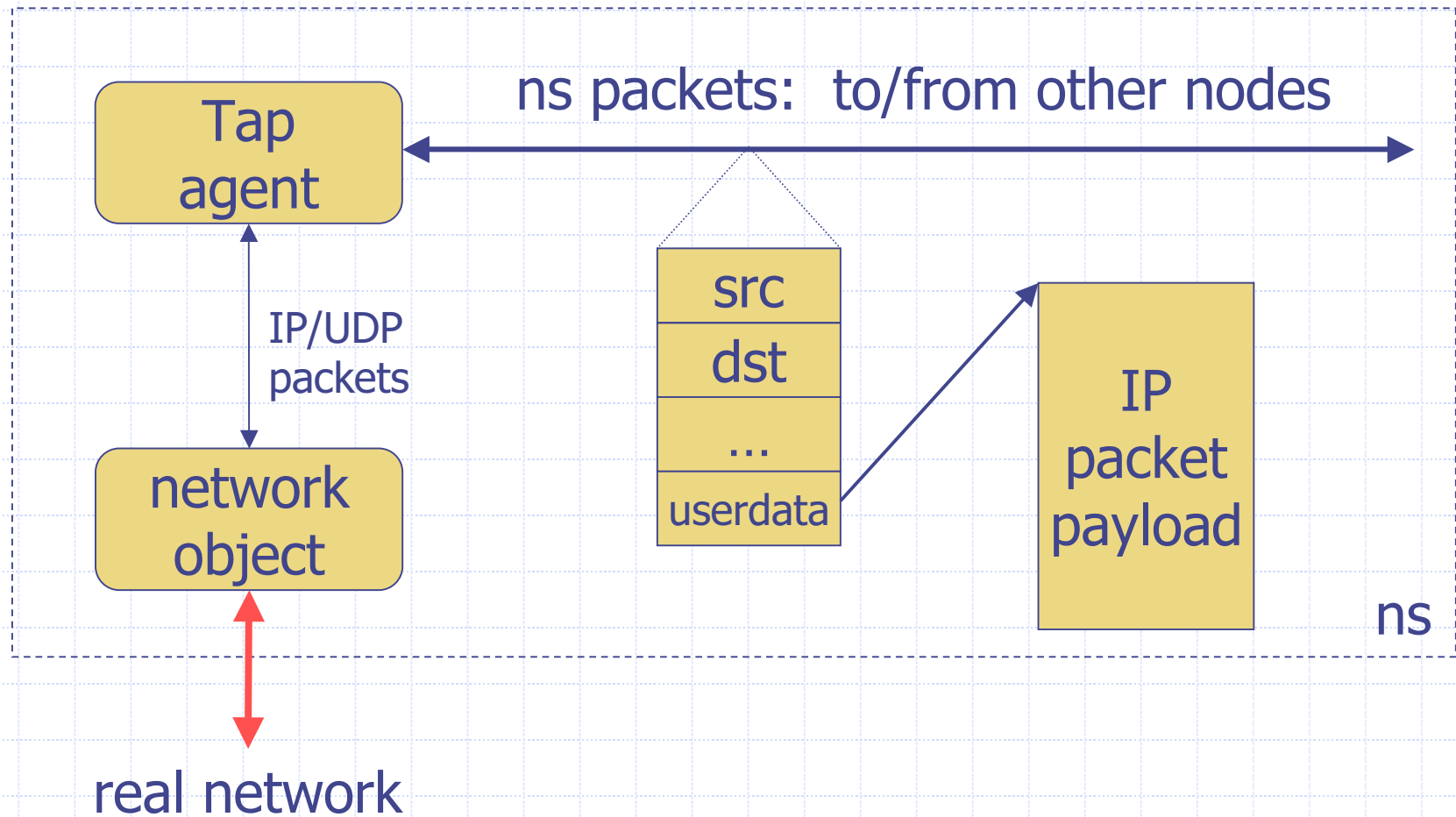
Realtime Scheduler

- ◆ Extended from Scheduler/List
- ◆ Synchronizes simulation time to real time
- ◆ Fails when simulation time falls behind
- ◆ `$ns use-scheduler RealTime`

Network Objects

- ◆ Abstraction of real traffic source/sink
- ◆ Base class for various network types
 - Opened read-only, write-only, or read-write
- ◆ Raw IP and UDP/IP network object
 - Send/receive raw IP packets or UDP/IP
 - IP multicast support
- ◆ Pcap network object
 - Send/receive link-layer frames
 - Use BPF/libpcap filtering language

Tap Agents



Emulation Modes

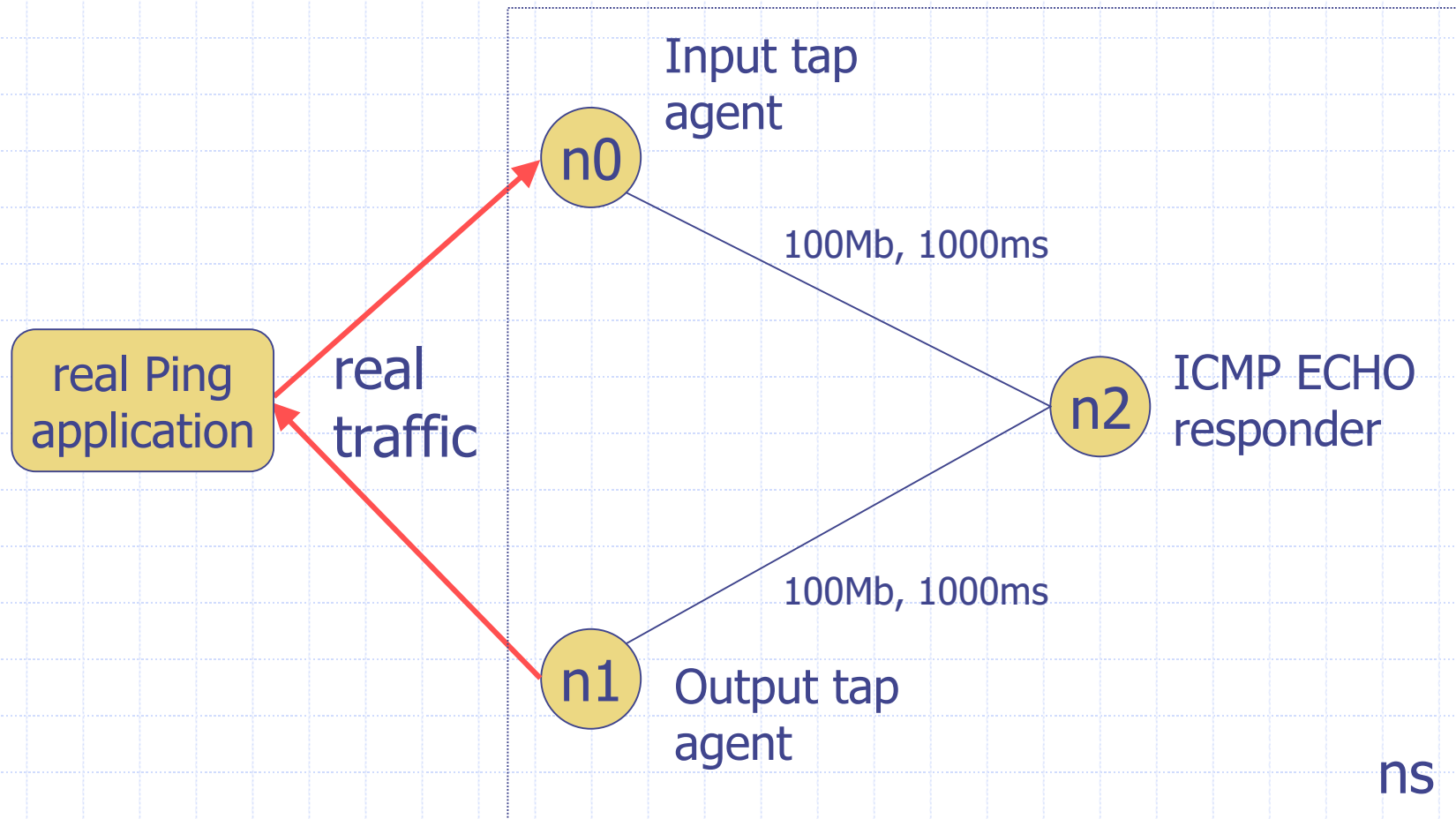
◆ Protocol mode

- Simulator interpret/generate live traffic
- Existing agents: ICMP ECHO, ICMP Redirect, ARP, TCP NAT

◆ Opaque mode

- Simulator does not interpret network data
- Operations: packet drop/reordering/delay...

Protocol Mode: Ping Responder



Ping: Step 1

◆ Stage setup

```
# Create simulator  
set ns [new Simulator]  
$ns use-scheduler RealTime
```

```
# Emulator address  
set me [exec hostname]  
# Or an arbitrary one (may require ARP support)  
# set me "10.11.12.13"
```

Ping: Step 2

◆ Create I/O network objects

```
# Packet input
set bpf0 [new Network/Pcap/Live]
$bpf0 set promisc_ true
set nd0 [$bpf0 open readonly fxp0]
set filt "(not ip host $me)"
$bpf0 filter $filt
```

```
# Packet output
set ipnet [new Network/IP]
$ipnet open writeonly
```

Ping: Step 3

◆ Agents

```
# Input agent
```

```
set pfa [new Agent/Tap]
```

```
$pfa network $bpf0
```

```
# Output agent
```

```
set ipa [new Agent/Tap]
```

```
$ipa network $ipnet
```

```
# ICMP ECHO agent
```

```
set echoagent [new Agent/PingResponder]
```

Ping: Step 4

◆ Create network topology

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
$ns simplex-link $n0 $n2 100Mb 1000ms DropTail
$ns simplex-link $n2 $n1 100Mb 1000ms DropTail

$ns attach-agent $n0 $pfa
$ns attach-agent $n1 $ipa
$ns attach-agent $n2 $echoagent
$ns simplex-connect $pfa $echoagent
$ns simplex-connect $ipa $echoagent
```

Ping: Step 5

◆ Start

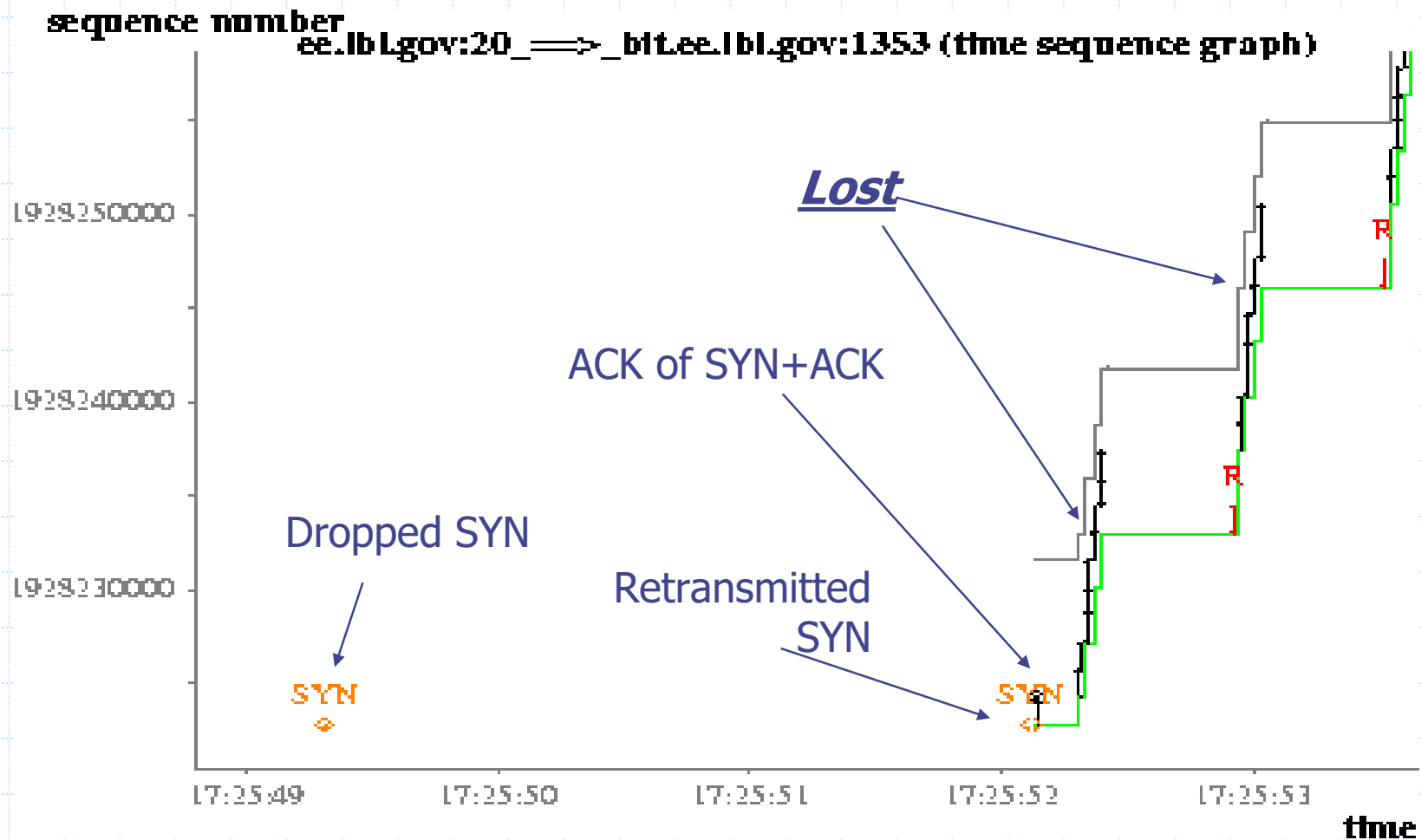
```
# Wait for ping to come in...
```

```
$ns run
```

◆ Result

- 2000.052ms ± 1.021ms

Opaque Mode (TCP: 10 packet periodic drop)



More Examples

◆ `~ns/emulate`

◆ Example scripts

- Protocol mode: `~ns/emulate/empaper.tcl`
- Opaque mode: `~ns/emulate/em3.tcl`

Outline

- ◆ Goals
- ◆ Essentials
- ◆ Primers
 - Wired world
 - Wireless world
 - Emulator
 - Utilities

Utilities

- ◆ Tcl debugger
- ◆ Topology generation
- ◆ Scenario generation
- ◆ Web cache trace converter

Debugging Your ns Script

◆ tcl-debug 1.9

- <http://expect.nist.gov/tcl-debug/>
- Works with Tcl 8.0.4 and below

◆ Installation

- [make distclean] in ns
- ./configure --with-tcldebug=<dir>
- make

Debugging Your ns Script

◆ Using tcl-debug

- Insert "debug 1" into your scripts, e.g.:

```
set tcp [new Agent/TCP]
```

```
debug 1
```

```
$tcp set window_ 200
```

- When "debug 1" is executed, ns drops to:

```
vint/ns-2(121): ./ns t.tcl
```

```
2: lappend auto_path $dbg_library
```

```
dbg2.0>
```

Debugging Your ns Script

```
dbg2.0> h
s [#]          step into procedure
n [#]          step over procedure
N [#]          step over procedures, commands, and arguments
c              continue
r              continue until return to caller
u [#]          move scope up level
d [#]          move scope down level
               go to absolute frame if # is prefaced by "#"
w              show stack ("where")
w -w [#]       show/set width
w -c [0|1]    show/set compress
b              show breakpoints
b [-r regexp-pattern] [if expr] [then command]
b [-g glob-pattern]  [if expr] [then command]
b [[file:]#]   [if expr] [then command]
               if pattern given, break if command resembles pattern
               if # given, break on line #
               if expr given, break if expr true
               if command given, execute command at breakpoint
b -#          delete breakpoint
b -           delete all breakpoints
```

Topology Generation

- ◆ <http://www.isi.edu/nsnam/ns/ns-topogen.html>

Packages	Graphs	Edge Method
NTG	n-level	probabilistic
RTG	Flat random	Waxman
GT-ITM	Flat random, n-level, Transit-stub	various
TIERS	3-level	spanning tree

GT-ITM

◆ Installation

- Comes with ns-allinone
- Require Knuth's cweb and SGB

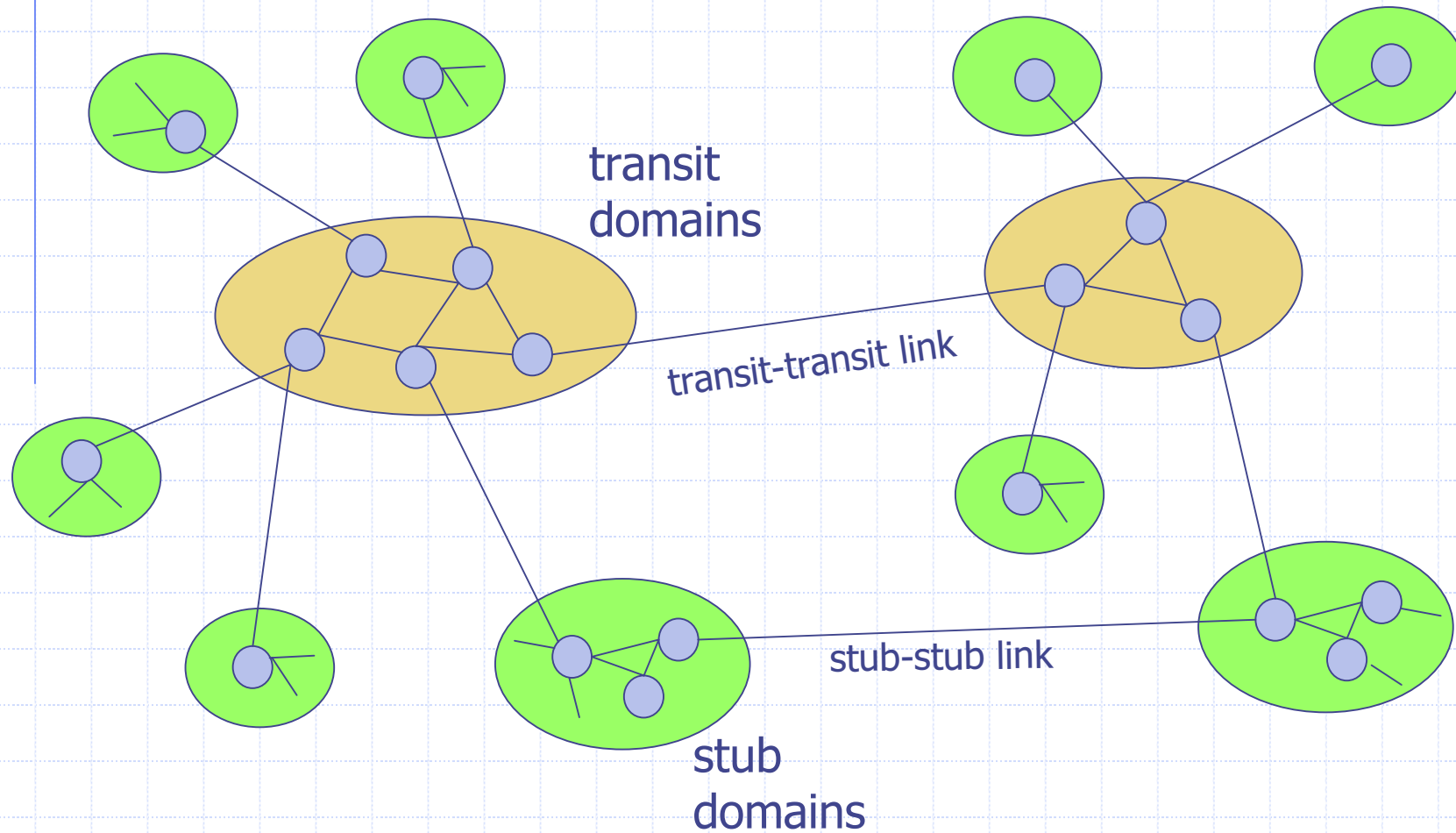
◆ Usage

- itm <config_file>

◆ Three graph models

- Flat random: Waxman
- n-level hierarchy
- Transit-stub

GT-ITM: Transit-Stub Model



GT-ITM: Example

- ◆ Transit-stub network

- ◆ Config file (e.g., ts1)

```
# <method keyword> <number of graphs> [<initial seed>]
# <# stubs/trans node> <#rand. t-s edges> <#rand. s-s
  edges>
# {<n> <scale> <edgmethod> <alpha> [<beta>] [<gamma>]}
# (average!) number of nodes =  $1 \times 2 \times (1 + 3 \times 4) = 26$ 
ts 10 47          # 10 graphs, init seed 47
3 0 0            # 2 stubs per transit nodes
1 20 3 1.0      # n. of transit domains (pure random)
2 20 3 0.5      # n. of nodes per transit domain
4 10 3 0.5      # n. nodes in each stub domain
```


GT-ITM: Example

◆ Run

- `itm ts1`
- Result: `ts1-{0-9}.gb`

◆ Result files in SGB format

Converters for GT-ITM

◆ sgb2ns

- Convert SGB format to ns config file
- `sgb2ns <SGB_file> <OTcl_file>`
- `ts2ns`: output lists of transit and stub nodes

◆ sgb2hier

- Convert transit-stub information into hierarchical addresses
- `sgb2hierns <SGBFile> <TclFile>`

Converters for GT-ITM

◆ Format of generated ns config files

```
proc create-topology {nsns node linkBW} {  
    upvar $node n  
    upvar $nsns ns  
    # Create nodes, links,  
    .....  
}
```

◆ Usage

```
source <OTcl_file>  
create-topology ns nodes 1.5Mb
```

See Your Topology

◆ Create an ns wrapper

```
# Assume you've done "sgb2ns ts1-0.gb ts1.tcl"  
source ts1.tcl  
set ns [new Simulator]  
$ns namtrace-config [open ts1.nam w]  
create-topology ns node 1.5Mb  
$ns at 1.0 "exit 0"  
$ns run
```



Summary of API Changes

New API

- ◆ \$ns_node-config
- ◆ \$ns node
- ◆ no global variable dependency
- ◆ namtrace-all-wireless
- ◆ Energy model support
- ◆ No global definition of channel and propagation models

OLD API

- ◆ {dsr/dsdv/tora}-create-mobile-node
- ◆ strong global variable dependency
- ◆ no nam support
- ◆ no energy model
- ◆ need global channel and propagation models