



Design Issues of High Speed Networks

(CSC-7601)

Lecture 2 & 3

(12 Sep. 2007)

Seung-Jong Park (Jay)

<http://www.csc.lsu.edu/~sjpark>

1



Network Simulator ns-2

2

Agenda

- Introduction
- Interface
 - Tcl and OTcl
 - TclCL
- Simulator
 - Wired network
 - Wireless network

3

Introduction

- NS-2: network simulator version 2
 - Discrete event simulator
 - Packet level simulation
- Features
 - Open source
 - Scheduling, routing and congestion control
 - Wired networks: P2P links, LAN
 - Wireless networks: terrestrial (ad-hoc, cellular; GPRS, UMTS, WLAN, Bluetooth), satellite
 - Emulation and trace

4

NS-2: Evolution

- REAL network simulator (Cornell), 1989
 - Study the dynamic behavior of flow and congestion control schemes in packet-switched data networks (written in C)
- NS (NS-1), 1995
 - Adopt the Tcl / C++ architecture
- NS-2, 1996
 - Object-oriented Tcl (Otccl)
- Wireless extensions
 - UC Berkeley Daedalus project
 - CMU Monarch project
 - Sun Microsystems

5

NS-2: Paradigm

- Object-oriented programming
 - Protocol layering
 - Modularity and extensibility
 - Large scale simulation
 - Maintenance and reusability
- Split-language programming
 - Scripting language (Tcl)
 - System programming language (C)

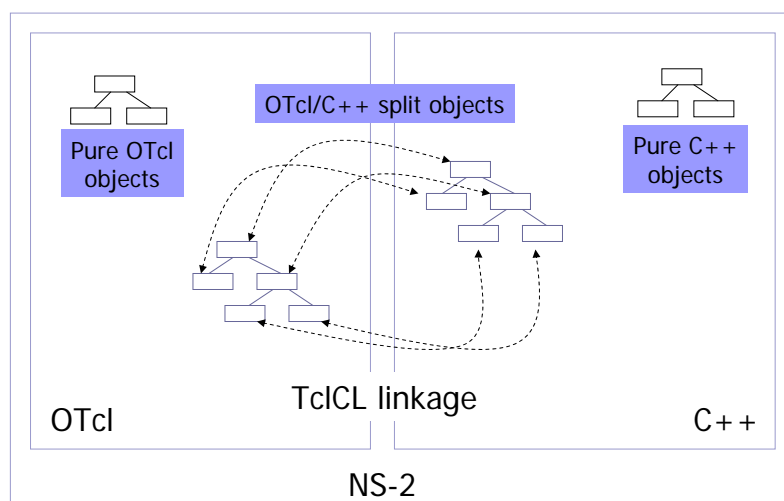
6

NS-2: Split Languages

- Tcl scripts (Tcl/OTcl)
 - Interpreted (interactive)
 - Setup and configuration
- C codes (C/C++)
 - Compiled (efficient)
 - Algorithms and protocols
- TclCL (OTcl/C++)
 - Link Tcl/OTcl scripts and C/C++ codes
 - Provide a layer of C++ glue over OTcl

7

NS-2: Split Objects



8

NS-2: A Tcl Script Example

```
#!/home/hsieh/ns-allinone-2.27/bin/ns

set ns [new Simulator]
set nf [open out.tr w];$ns trace-all $nf

for {set i 0} {$i<2} {incr i} {      ;# create the nodes
    set n($i) [$ns node]}
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail

# Create a UDP agent
set udp(src) [new Agent/UDP]
$udp(src) set packetSize_ 500
$ns attach-agent $n(0) $udp(src)

proc finish {} {
    global ns nf
    $ns flush-trace; close $nf
}

$ns at 5.0 "finish"
$ns run
```

```
/home>ns abc.tcl
/home>abc.tcl
```

9

NS-2: A C++ Code Example

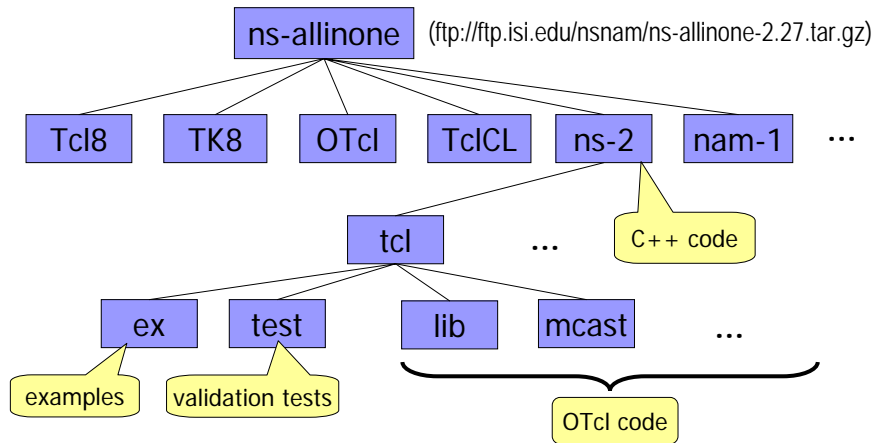
```
static class UdpAgentClass : public TclClass {
public:
    UdpAgentClass() : TclClass("Agent/UDP") {}
    TclObject* create(int, const char*const*) {
        return (new UdpAgent());
    }
} class_udp_agent;

UdpAgent::UdpAgent() : Agent(PT_UDP), seqno_(-1)
{
    bind("packetSize_", &size_);
}

void UdpAgent::sendmsg(int nbytes, AppData* data, const char* flags)
{
    Packet *p;
    p = allocpkt();
    hdr_cmh::access(p)->size() = size_;
    hdr_rtp::access(p)->seqno() = ++seqno_;
    p->setdata(data);
    target_->recv(p);
}
```

10

NS-2: Directory Structure



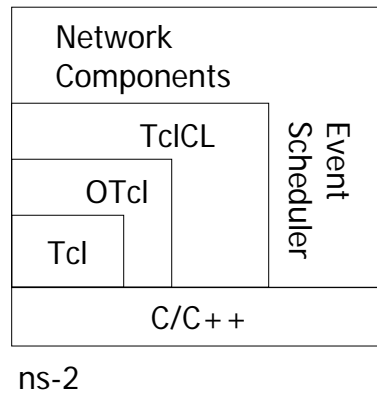
11

Network Simulator ns-2

Part I: Tcl, OTcl and TclCL

12

NS-2: A Tcl Extension



13

Tcl: Overview

- Tcl: Tool command language
- Tcl is extensible and embeddable
 - NS-2 is also a Tcl interpreter (`tclsh` or `ns`)
- Tcl is a scripting language
 - Ideal for network configuration
- A Tcl script consists of commands
- To write Tcl scripts, you need to learn
 - Tcl command
 - Tcl syntax (how commands are parsed)

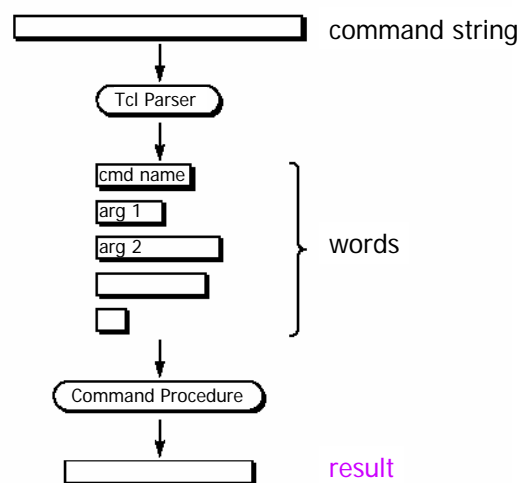
14

Tcl: Command

- A command consists of words
 - cmdName arg1 arg2 ...*
 - *cmdName*: core command or procedure
 - set, puts, expr, open, if, for, ...
 - All words are considered as **strings**
 - White space (space/tab) separates arguments
 - Newline or semicolon (;) terminates a command
- Command evaluation: parsing and execution
 - The interpreter does “substitution” and “grouping” (parsing) before running a command
 - Every command returns a result string after execution

15

Tcl: Command



16

Tcl: Command Example

```
puts hello

puts "hello"

open test.tcl w

set a 3
set b 4; set c \
5

set a

expr 1+2
expr 1 + 2
expr "1" "+2"
```

What is the use of the double quote?

Double quotes are used to group words into a single argument to a command. Dollar signs and square brackets are interpreted inside double quotes.

17

Tcl: Substitution

- Variable substitution **\$**
 - *\$varName* will be replaced by its value
 - Variables are created automatically when **assigned** to (no declaration is necessary)
- Command substitution **[]**
 - [*Tcl script*] will be replaced by its result
 - Nesting and multiple commands
- Backslash substitution ****
 - *\n, \t, \67, \x67, ...* and *\\$, \[, \\\, \", \{*
 - *\newline, \space*
- A single pass of substitution

18

Tcl: Substitution Example

```
set a 3
puts $a
puts a
puts $z

set b [expr 2+3]
set c [puts hi]
set e [set d [expr $b/2]]
set f [expr 3][set e]
set g [expr 3;set e]

expr 31 + 3
expr 031 + 3
expr 0x31 + 3
expr \x31 + 3
```

19

Tcl: Grouping (Quoting)

- Group words into a single word
 - Space, newline and semicolon are not interpreted (lose their functions when quoted)
- Grouping before substitution
 - Allow substitution: double quotes ""
 - Prevent substitution: braces {}

20

Tcl: Grouping Example

```

set msg This Is A Wrong Example
set msg This\ Is\ A\ Correct\ Example
set msg "This Is A Correct Example"
set msg {This Is A Correct Example}
set msg `This Is A Wrong Example`

puts "hello; puts hi"
puts "hello
John"

set a 3
puts "$a+2 is\t [expr $a+2]"
puts {$a+2 is\t [expr $a+2]}

for {set i 0} {$i<5} {incr i} {puts $i}

```

21

Tcl: Variable

- Variable name
 - `varName` can consist of any character
 - By default Tcl assumes `varName` contains only letters, digits and the underscore
 - Use of `#{varName}` for delimiting the name
- Simple variable
 - The variable is always stored as a string
- Associative array (`()`)
 - Variables with a **string-valued index** (mapping)
 - Array name and element name
 - Multi-dimensional array
 - `array` command

22

Tcl: Variable Example (1)

```

set c 122; set 122 c

set d value; set e d
set e
set $e
set [set e]

set "link bandwidth" 3
expr ${link bandwidth} * 5
expr $"link bandwidth" * 5

set rate [expr 5*2]
set bandwidth $rateMb
set bandwidth ${rate}Mb
set bandwidth $rate.5Mb

```

23

Tcl: Variable Example (2)

```

set x 3
set "x" "3"
expr $x * "10"

set arr(0) 7
set arr(1) hello
set arr(two) 3
set arr(the\ name) {the value}
array names arr;array size arr

set mat(1,1) 10
set mat(1,2) 5

set x 1;set y 2
puts $mat($x,$y)

```

```
set mat(1, 1) 10
```

24

Tcl: Procedure

- Define a procedure

```
proc procName arg body
```

- Procedure name and variable name are in different name spaces
- Procedure nesting
- Global scope for procedure name
- Default argument value (quoting with { })
- Variable length argument list *args*
- Return value of a procedure

25

Tcl: Procedure Example

```
proc add {a b} {expr $a + $b}
```

```
proc inc {var {dv 1}} {  
    set a [expr $var+$dv]  
    return $a  
}
```

```
proc greet {} {puts "hello there"}
```

```
proc add args {  
    set s 0  
    foreach i $args {incr s $i}  
    return $s  
}
```

26

Tcl: Scope

- Local scope inside the procedure
 - Variables defined outside the procedure (global variable) are *invisible* to the procedure
- `global varName1 varName2 ...`
 - Use array for a collection of global variables
- `upvar ?level? varName localName`
 - Level: 1 (relative level), #0 (absolute/global level)
 - Call by reference
- Static variable
 - Use global variable

27

Tcl: Scope Example

```

proc topology {link} {
    global node
    for {set i 0} {$i<$link} {incr i} {
        set node($i) [new Node]
    }
}

proc topology-2 {var link} {
    upvar $var nn
    for {set i 0} {$i<$link} {incr i} {
        set nn($i) [new Node]
    }
}

topology 3
topology-2 node 3

```

28

Tcl: Miscellaneous

- Comment (#)
 - Comment is also a command
 - It is placed where a command is expected
- Evaluation (`eval`)
- Command line arguments
 - `argc`, `argv`, `argv0`
- Script files
 - `source`

Tcl: Miscellaneous Examples

```
set x 2          # not a comment
set x 2          ;# a comment

if {$x==2} {    # a comment
  puts "x is 2"}

set bw [expr $x==1 ? 3 : 5]
set bw [expr {[info exists z] ? $z : 0}]
set bw [expr $x ? 3Mb : 5Mb]

set x "puts hello"
eval $x
```

Tcl: Core Commands

- Control flow
 - if, switch, while, for, foreach
- File access
 - open, close, flush, puts, gets
- String manipulation
 - glob-style and regular expression
- List manipulation
 - llength, lindex, linsert, lreplace
 - lappend
- string and array commands

31

NS-2: A Tcl Script (Recap)

```
#!/home/hsieh/ns-allinone-2.27/bin/ns

set ns [new Simulator]
set nf [open out.tr w];$ns trace-all $nf

for {set i 0} {$i<2} {incr i} {      ;# create the nodes
    set n($i) [$ns node]}
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail

# Create a UDP agent
set udp(src) [new Agent/UDF]
$udp(src) set packetSize_ 500
$ns attach-agent $n(0) $udp(src)

proc finish {} {
    global ns nf
    $ns flush-trace; close $nf
}

$ns at 5.0 "finish"
$ns run
```

32

OTcl: Overview

- OTcl: Object Tcl
 - Object-oriented
 - Class and inheritance
 - Dynamic
 - Class can be defined incrementally
 - Methods and classes can be modified at any time
 - Instance can behave **differently** from the class itself
 - Object command approach
 - Each object is registered as a **command** to the parser
 - Each subcommand is an "argument" to the object
- OTcl interpreter: `otclsh` or `ns`

33

OTcl: Class

- Class command
 - **class** *clsName* to create a class
 - *clsName* **instproc** to define a class method
- Class variable
 - *clsName* **set** *varName* *varValue*
 - *clsName* **instvar** to link to a class variable
- All instance variables and methods of the class are public
- Inheritance
 - *clsName* **superclass** to set parent class

34

OTcl: Class

- Comparison with C++
 - Class definition
 - instproc and instvar (set)
 - Constructor and destructor
 - init and destroy
 - Method shadowing and combination
 - next
 - Method invocation
 - self
 - Static variable
 - Instance lifecycle
 - new and delete

35

OTcl: An Example

```

Class Safety
Safety instproc init {} {
    $self next
    $self set count 0
}
Safety instproc put {thing} {
    $self instvar count
    incr count
    $self next $thing
}
Safety instproc get {} {
    $self instvar count
    if {$count==0} {return {empty!}}
    incr count -1
    $self next
}

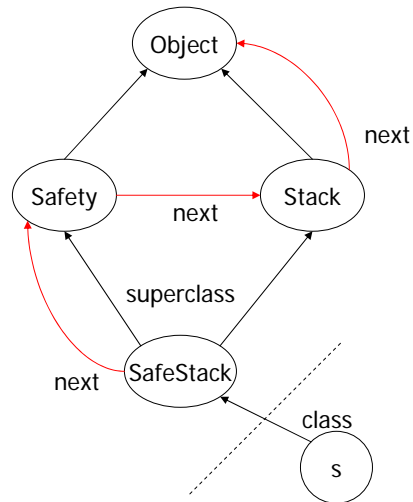
Class Stack
Stack instproc init {} {
    $self next
    $self set pile {}
}
Stack instproc put {thing} {
    $self instvar pile
    set pile [concat [list $thing] \
                $pile]
    return $thing
}
Stack instproc get {} {
    $self instvar pile
    set top [lindex $pile 0]
    set pile [lrange $pile 1 end]
    return $top
}

Class SafeStack -superclass {Safety Stack}
SafeStack s

```

36

OTcl: Inheritance



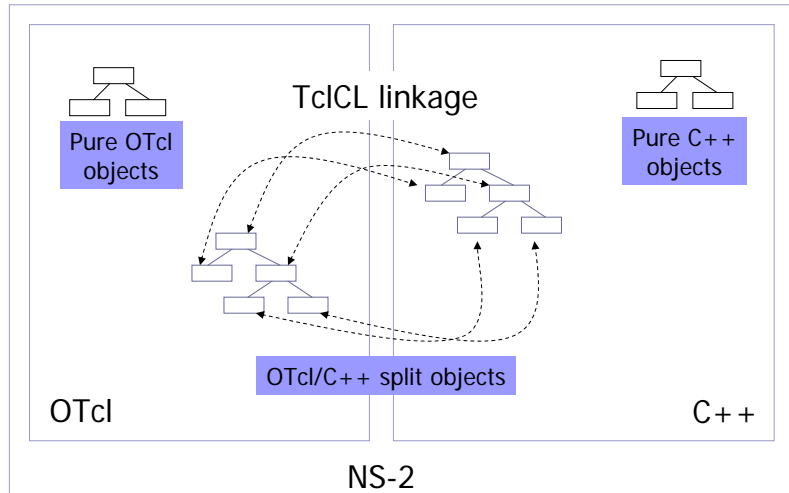
37

TclCL: Overview

- TclCL: Tcl with Classes
- NS-2 is written in C++ with OTcl interpreter as a front end
- Class hierarchy
 - Compiled hierarchy and interpreted hierarchy
 - One-to-one correspondence of objects from users' perspective
 - Simulator objects are implemented in the compiled hierarchy, but instantiated through the interpreter
 - TclObject is the root of the hierarchy

38

TclCL: NS-2 Objects



39

TcICL: OTcl/C++ Linkage

TclObject	Root of NS-2 object hierarchy
	bind(): link variable values between C++ and OTcl
	command(): link OTcl methods to C++ implementations
TclClass	Create and initialize TclObject's
Tcl	C++ methods to access the OTcl interpreter
TclCommand	Standalone global commands
EmbeddedTcl	NS-2 script initialization

40

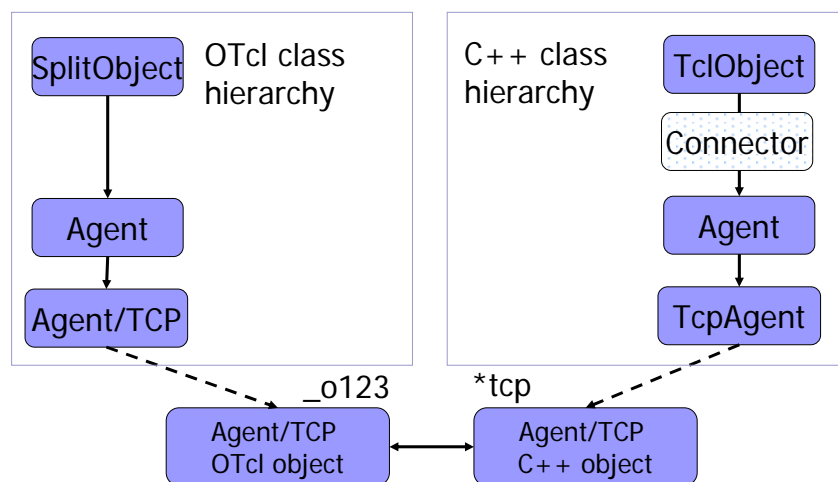
TclCL: Class TclObject

- Base class in NS-2 for split objects
 - Mirrored in both C++ (TclObject) and OTcl (SplitObject)
- Usage
 - Instantiation, bind and command
- Example

```
set tcp [new Agent/TCP]
$tcp set window_ 30
$tcp advanceby 5000
```

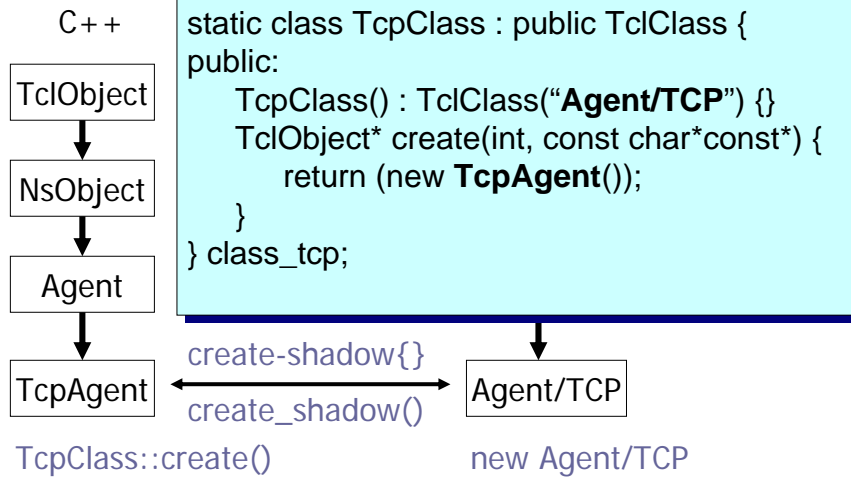
41

Class TclObject: Hierarchy



42

TclCL: Class TclClass



Class TclObject: Binding

- Bi-directional variable bindings
 - Link C++ member variables (compiled) to OTcl instance variables (interpreted)
- Initialization through the closest OTcl class variable
 - Agent/TCP set **window_** 50
- Do all initialization of bound variables in ~ns/tcl/lib/ns-default.tcl
 - Otherwise a warning will be issued when the shadow compiled object is created

Class TclObject: Binding

- C++

```
TcpAgent::TcpAgent() {
    bind("window_", &wnd_);
    ... ..
}
```

- bind(), bind_time(), bind_bool(), bind_bw()

- OTcl

```
Agent/TCP set window_ 50
set tcp [new Agent/TCP]
$tcp set window_ 100
```

45

Class TclObject: Command

- Invoke C++ compiled functions through OTcl interpreted methods

- A way of implementing OTcl methods in C++

- Hook point

- Tcl method unknown{}

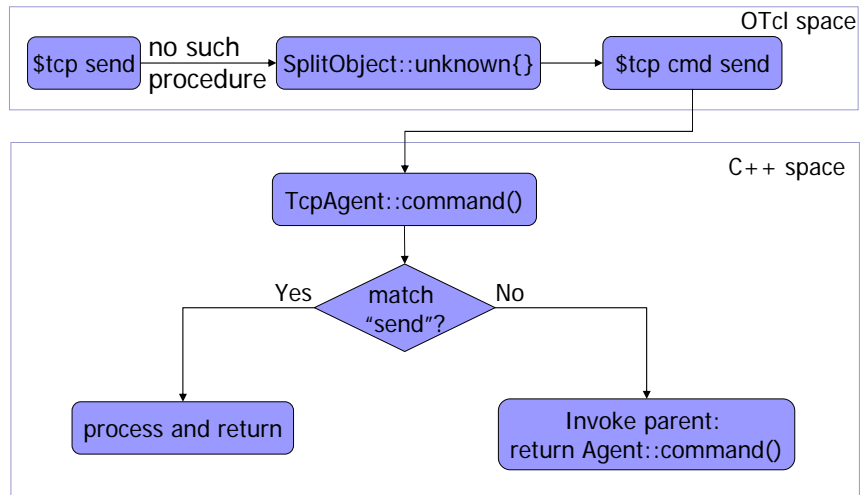
- OTcl method cmd{}

- Send all arguments after cmd{} call to TclObject::command()

- Use Tcl::resultf() in C++ to pass back results

46

Class TclObject: Command



47

Class TclObject: Command

```

■ OTcl
set tcp [new Agent/TCP]
$tcp advance 100
■ C++
int TcpAgent::command(int argc,
                      const char*const* argv) {
    if (argc == 3) {
        if (strcmp(argv[1], "advance") == 0) {
            int newseq = atoi(argv[2]);
            .....
            return TCL_OK;
        }
    }
    return (Agent::command(argc, argv));
}

```

48

TclCL: Class Tcl

- Class Tcl encapsulates the instance of the OTcl interpreter
 - It provides methods in C++ to access and communicate with the interpreter
- Usage
 - Obtain a reference to the OTcl instance
 - Invoke OTcl procedure
 - Obtain or pass back OTcl evaluation results
 - Return success/failure code to OTcl

49

Class Tcl: Example

```

■ C++ (app.cc)
Tcl& tcl = Tcl::instance();
if (argc == 2) {
    if (strcmp(argv[1], "agent") == 0) {
        tcl.resultf("%s", agent->name());
        return TCL_OK;
    } else if (strcmp(argv[1], "start") == 0) {
        tcl.evalf("[%s info class] info instprocs",
                 name_);
        sprintf(result, " %s ", tcl.result());
        ...
    }
    tcl.error("unknown command");
}

```

50

TclICL: Summary

- Class TclObject
 - Unified interpreted (OTcl) and compiled (C++) class hierarchies
 - Seamless access (procedure call and variable access) between OTcl and C++
- Class TclClass
 - Mechanism that makes TclObject work
- Class Tcl
 - Primitives to access OTcl interpreter

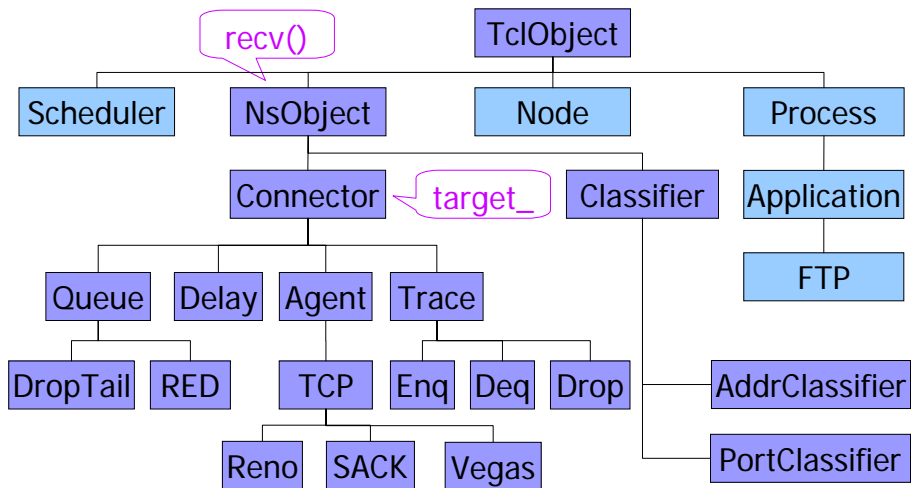
51

Network Simulator ns-2

Part II: Wired Network

52

Class Hierarchy



53

Simulation Elements

- Create the event scheduler (simulator)
- [Setup tracing]
- Create network topology
- [Setup routing]
- [Insert error modules/network dynamics]
- Create connection (transport)
- Create traffic (application)
- Start the scheduler
- Post-process data

54

Event Scheduler

- Create event scheduler
 - `set ns [new Simulator]`
- Schedule events (OTcl)
 - `OTcl: $ns at <time> <TCL_command>`
 - `C++: Scheduler::schedule(h,e, delay)`
- Obtain simulation time
 - `OTcl: $ns now`
 - `C++: Scheduler::clock()`
- Start scheduler
 - `$ns run`
 - The last line of your OTcl script

55

Trace

- Trace packets on all links
 - `$ns trace-all [open nstr.out w]`

```

<event> <t> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <uid>
+ 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

```

 - `$ns namtrace-all [open namtr.out w]`
- Turn on tracing on specific links
 - `$ns trace-queue $n0 $n1`
 - `$ns namtrace-queue $n0 $n1`
- Output trace to `/dev/null` if not desired

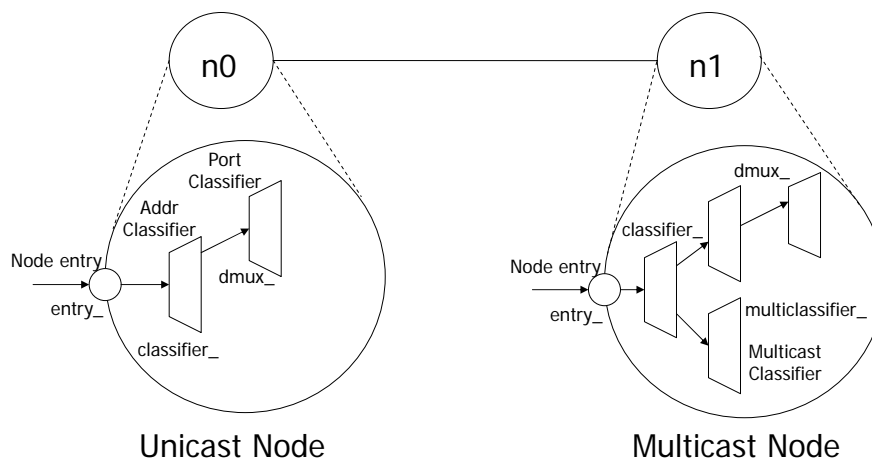
56

Network Topology

- Nodes
 - `set n0 [$ns node]`
 - `set n1 [$ns node]`
- Links and queues
 - `$ns duplex-link $n0 $n1 \`
`<bandwidth> <delay> <queue>`
 - `bandwidth: bind_bw(), delay: bind_time()`
 - `queue: DropTail, RED, CBQ, FQ, ...`
 - `Link delay = f(bandwidth, delay)`
= packet transmission time + propagation delay

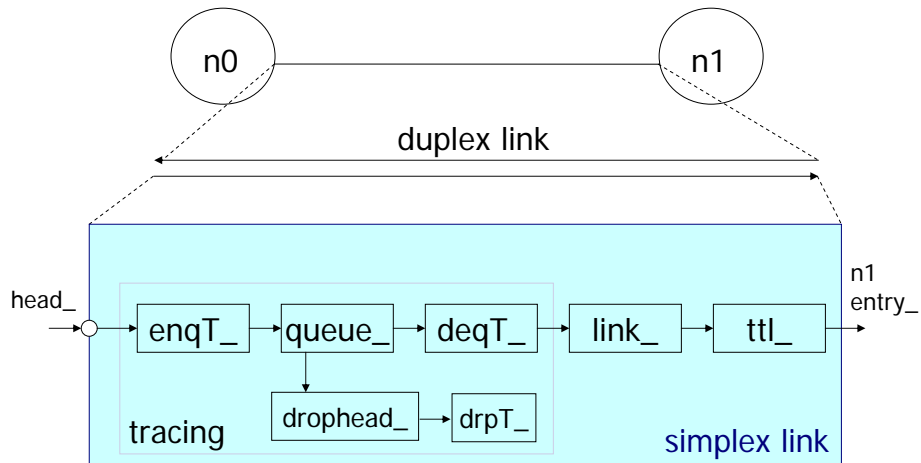
57

Network Topology: Node



58

Network Topology: Link



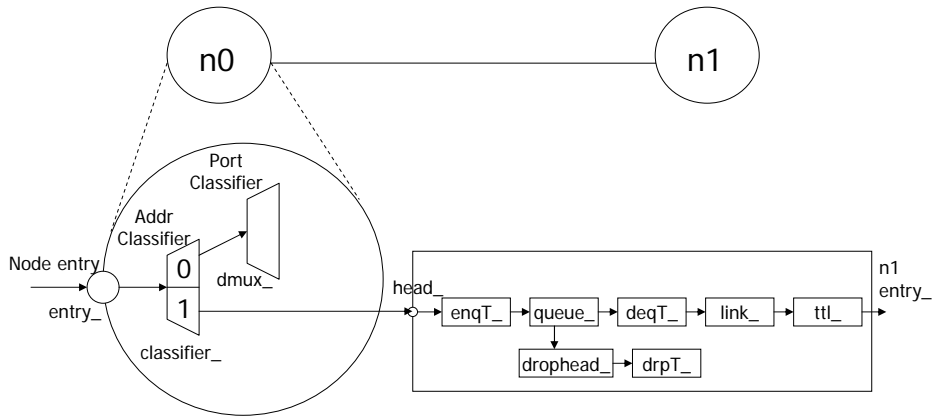
59

Routing

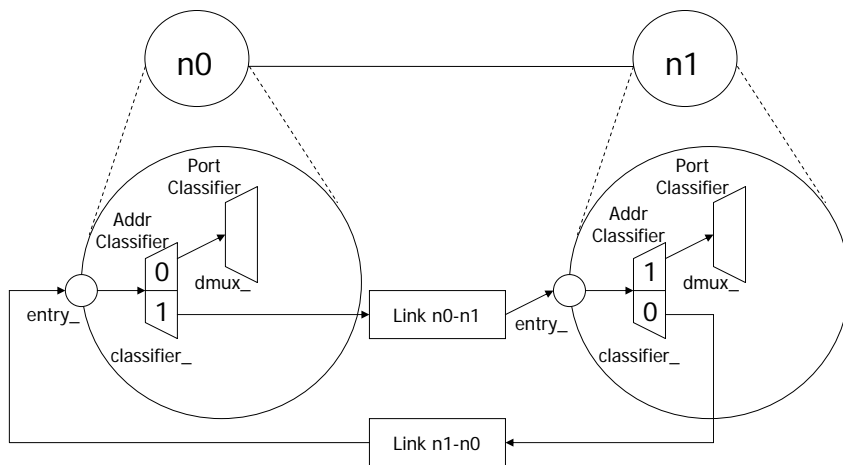
- Unicast routing
 - `$ns rtp proto <type> <nodes>`
 - type: Static (default), Session, DV, LS, Manual
 - nodes: default entire topology
- Default static routing
 - Dijkstra's all-pairs shortest path first algorithm
 - Route calculation is done before simulation starts
- Link cost
 - `$ns cost $n0 $n1 <cost>`
 - default link cost = 1

60

Routing



Routing

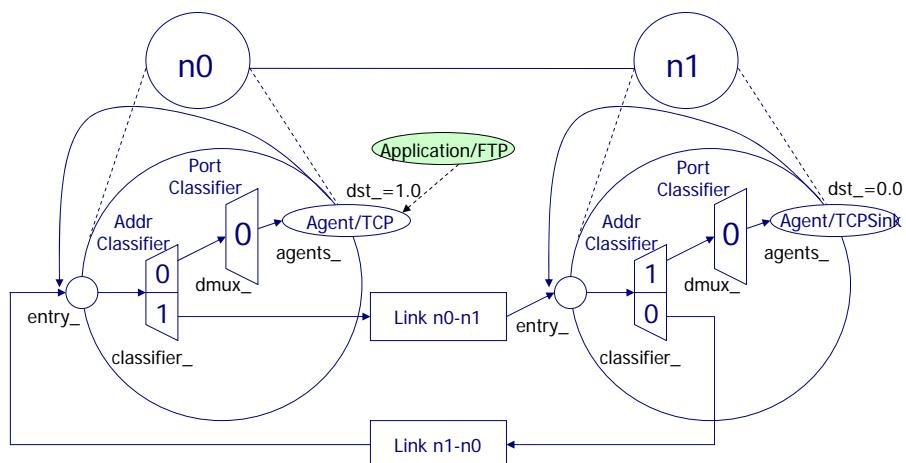


Application

- FTP
 - set ftp [new Application/FTP]
 - \$ftp attach-agent \$tcp
 - \$tcp attach-app FTP
- CBR
 - set cbr [new Application/Traffic/CBR]
 - \$cbr set packetSize_ 1000
 - \$cbr set rate_ 16000
- Start traffic generation
 - \$ns at <time> "\$app start"

65

Application



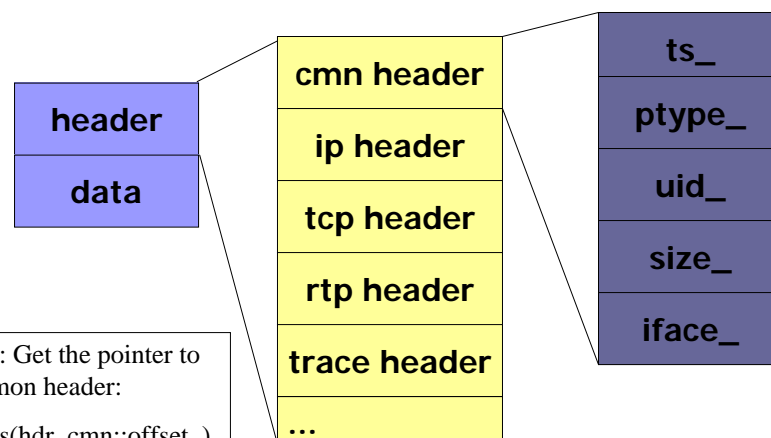
66

Packet

- Packets are events
 - Can be scheduled to “arrive”
- Packets contain header section and data
 - Header section is a cascade of all in-use headers
- Each packet contains a common header
 - packet size (used to compute transmission time)
 - packet type
 - timestamp, uid, ...
- All in-use headers are included when simulation starts
 - Change packet size to reflect header cascading

67

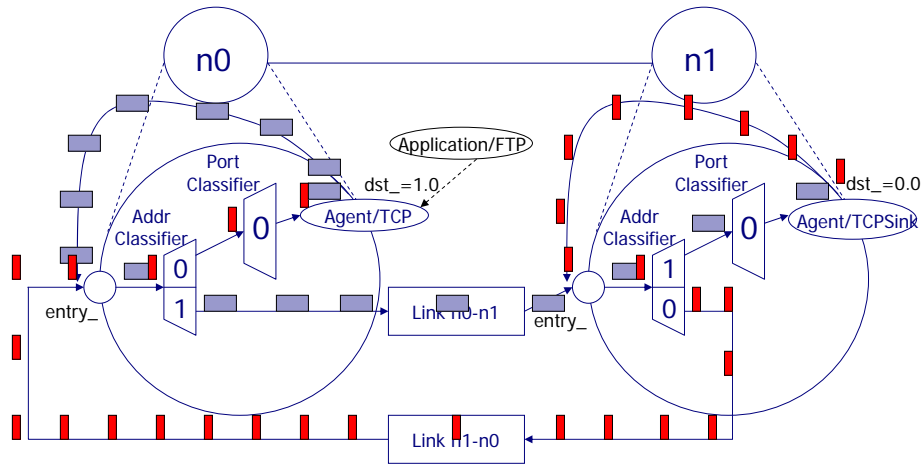
Packet Header



Example: Get the pointer to the common header:
`p->access(hdr_cmn::offset_)`
 or, `HDR_CMN(p)`

68

Packet Flow



69

Simple code (I)

```

set ns [new Simulator]

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

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

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"
$ns at 5.0 "stop"

$ns run

```

Simple.tcl

```

Simulator instproc init args {
    $self create_packetformat
    $self use-scheduler Calendar
    $self set nullAgent_ [new Agent/Null]
    $self set-address-format def
    eval $self next $args
}

```

Run your program →
% ns Simple.tcl

70

Simple code (II)

```

set ns [new Simulator]

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

$ns duplex-link $n0 $n1 1Mb 10ms

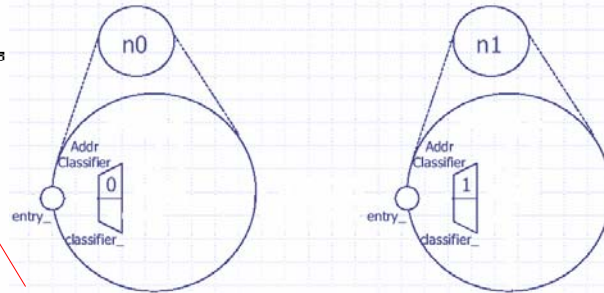
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"
$ns at 5.0 "stop"

$ns run

```



71

Simple code (III)

```

set ns [new Simulator]

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

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

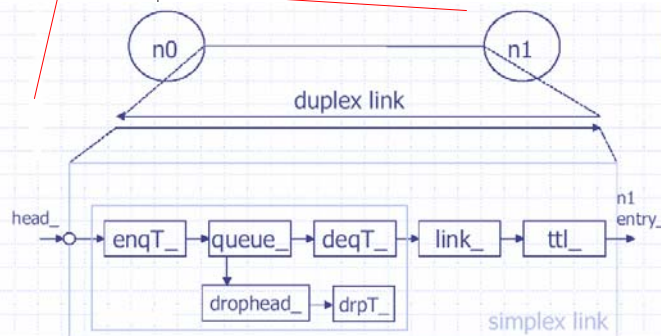
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"
$ns at 5.0 "stop"

$ns run

```



72

Simple code (IV)

```

set ns [new Simulator]

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

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

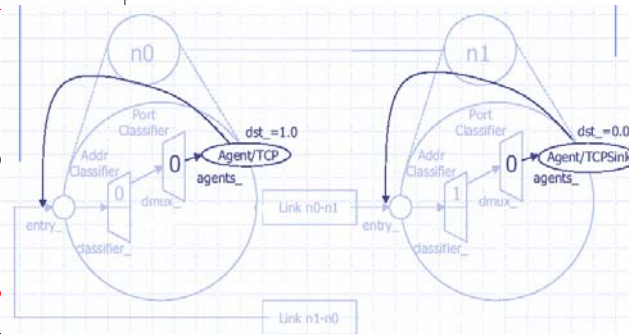
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"
$ns at 5.0 "stop"

$ns run

```



73

Simple code (V)

```

set ns [new Simulator]

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

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

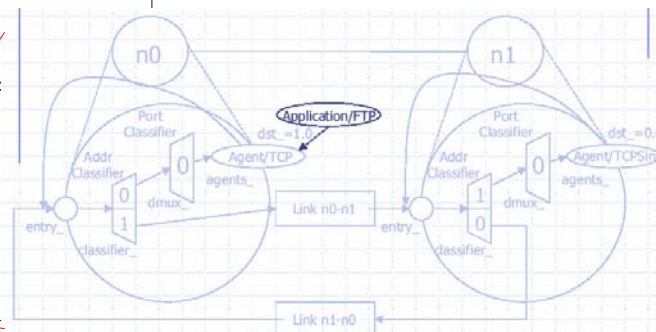
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"
$ns at 5.0 "stop"

$ns run

```



74

Recap

- NsObject: generic receive method `recv()` for packet reception
- Connector: one neighbor `target_`
- Node: collection of classifiers and agents
- Link: encapsulation of queue and delay
- Classifier: packet demultiplexer (routing)
- Agent: protocol endpoint or implementation of routing protocol
- Application: traffic generation

75

Network Simulator ns-2

Part III: Wireless Network

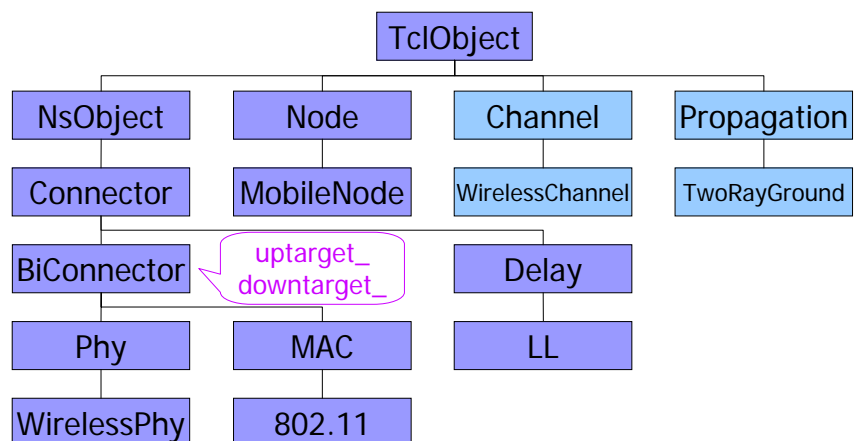
76

Wireless Network

- Wireless network
 - Nodes can move
 - No explicit “links” used to connect nodes
- Wireless network extension
 - Mobile node
 - Wireless channel and propagation model
 - Packet headers
 - Topology and movement
 - Routing and forwarding

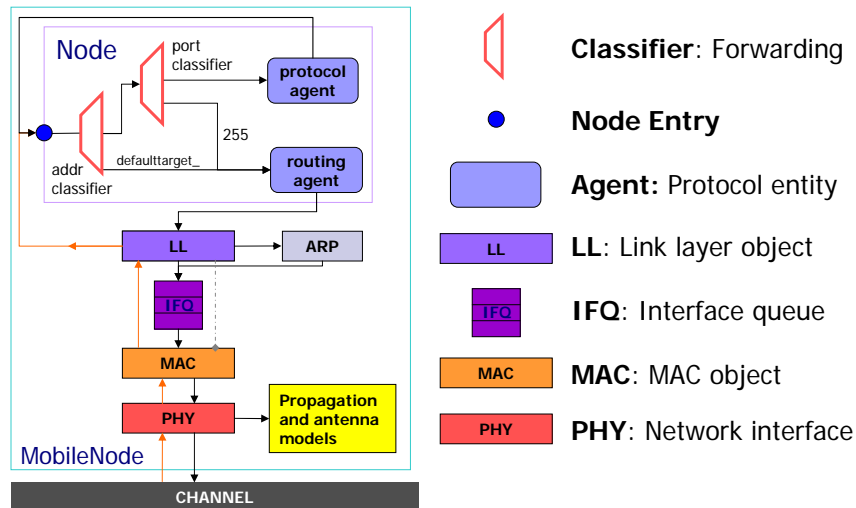
77

Class Hierarchy



78

Mobile Node: Portrait



79

Mobile Node: Components

- Link layer and ARP
 - Same as for LAN, but with a separate ARP module
 - ARP holds only one packet to the same destination
- Interface queue
 - Use callback to allow MAC retransmission
 - Use priority queue to give priority to routing protocol packets
- MAC layer
 - IEEE 802.11
 - RTS/CTS/DATA/ACK for all unicast packets
 - Physical and virtual carrier sense

80

Mobile Node: Components

- Network interface (PHY)
 - Parameters based on DSSS (WaveLAN 914MHz)
 - Interface with antenna and propagation models for packet reception decision
 - Update energy upon transmission and reception
- Radio propagation model
 - Friss-space attenuation ($1/r^2$) at near distance
 - Two-ray ground reflection ($1/r^4$) at far distance
- Antenna
 - Omni-directional antenna with unity gain

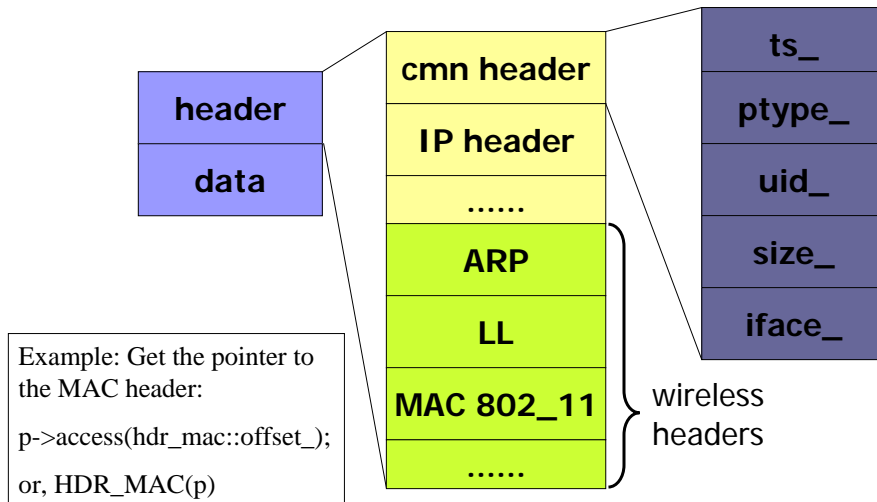
81

Wireless Channel

- Duplicate packets to all mobile nodes attached to the channel except the sender
 - Propagation delay is included
 - Use of multiple channels is possible
- It is the receiver's responsibility (PHY) to decide if it will accept the packet
 - Decision is based on received signal power
 - Each packet will have the transmission power stamped
 - Currently interference from other transmissions is not included in reception decision
 - Collision is handled at individual receiver

82

Wireless Packet Header



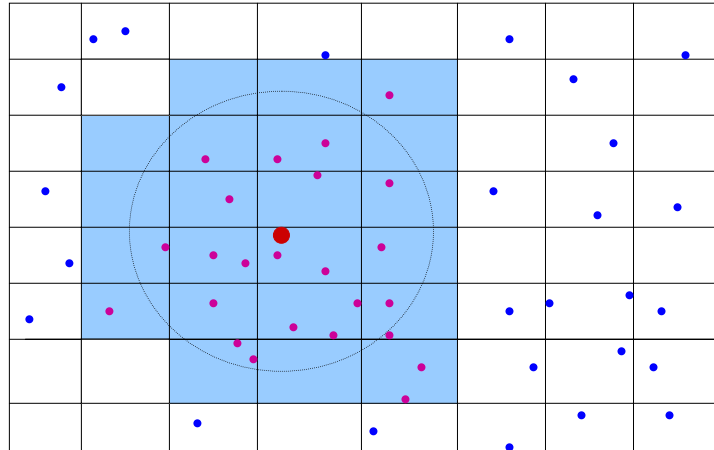
83

Node Movement

- Location
 - Coordinates (x,y,z)
- Movement
 - Waypoint movement model
 - Random destination
 - Random speed [0, maxSpeed]
 - Random pause time or random moving time

84

Network Topology



85

Example: Ad Hoc Network

■ Scenario

- 3 mobile nodes
- Move within a 670m*670m flat topology
- DSR ad hoc routing protocol
- Random waypoint mobility model
- UDP and CBR traffic

86

An Example – Step 1

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

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

# ns trace and nam trace
$ns trace-all [open ns.tr w]
$ns namtrace-all-wireless [open ns.nam w] 670 670
```

87

An Example – Step 2

```
# Create God
set god [create-god 3]
```

- **God: General Operations Director**
 - Keep the number of nodes in the network
 - Called by 802.11 MAC to keep a sequence number cache of all nodes
 - Store an array of the smallest number of hops required to reach one node to another
 - Used for setdest operation

```
$ns at 100.00 "$god set-dist 2 3 1"
```

88

An Example – Step 3

```
# Define how to create a mobile node
$ns node-config \
  -adhocRouting DSR \
  -llType LL \
  -macType Mac/802_11 \
  -ifqLen 50 \
  -ifqType Queue/DropTail/PriQueue \
  -phyType Phy/WirelessPhy \
  -antType Antenna/OmniAntenna \
  -propType Propagation/TwoRayGround \
  -channel [new Channel/WirelessChannel] \
  -topoInstance $topo
-agentTrace ON \
-routerTrace OFF \
-macTrace OFF \
-movementTrace OFF
```

89

Energy Parameters

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

- Node is energy-aware
 - Node status: on / off / sleep
- Pt_ and Pt_consume_

90

An Example – Step 4

```
# Create mobile nodes
for {set i 0} {$i<3} {incr i} {
  set node($i) [$ns node]
  # disable random motion for static network
  $node($i) random-motion 0
}

# Define movement model (if applicable)
source movement-scenario-files

# Define traffic model (if applicable)
source traffic-scenario-files
```

91

Scenario: Movement

- Mobile movement generator
 - `~ns/indep-utils/cmu-scen-gen/setdest/setdest`
 - `setdest -n <numNodes>`
 - `-p <pauseTime> -s <maxSpeed>`
 - `-t <simTime> -x <maxX> -y <maxY>`
- Random movement
 - `$node random-motion 1`
 - `$node start`
 - Change `POSITION_UPDATE_INTERVAL` and `MAX_SPEED` internally

92

A Movement File

```

$node_(0) set X_ 83.4
$node_(0) set Y_ 239.4
$node_(0) set Z_ 0.0
$node_(1) set X_ 257.1
$node_(1) set Y_ 345.4
$node_(1) set Z_ 0.0
$node_(2) set X_ 591.3
$node_(2) set Y_ 199.4
$node_(2) set Z_ 0.0
$ns_ at 33.0 "$node_(0) setdest 89.7 283.5 19.2"
$ns_ at 51.0 "$node_(1) setdest 221.8 80.9 14.9"
$ns_ at 50.0 "$node_(2) setdest 369.5 170.5 3.4"

```

93

Scenario: Traffic

- Traffic pattern generator
 - CBR (UDP) or FTP (TCP) traffic
 - `~ns/indep-utils/cmu-scen-gen/cbrgen.tcl`
`ns cbrgen.tcl [-type cbr|tcp]`
`[-nn nodes] [-seed seed]`
`[-mc connections] [-rate rate]`
- Specify in the OTcl script
 - Same as the wired network scenario

94

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_ 1000
$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)
$ns_ at 20.0 "$cbr_(0) start"

```

95

An Example – Step 5

```

# 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
$ns at 100.0 "$ns nam-end-wireless 100.0"
$ns at 100.0 "$ns halt"

# Start your simulation
$ns run

```

96

Summary

- NS-2 is an open source, discrete event, and packet level network simulator
- NS-2 is written in C++ with OTcl interpreter as a front end
- TclCL provides linkage for class hierarchy, object instantiation, variable binding and command dispatching
- NS-2 provides abundant implementations of protocols used in wired and wireless networks

References

- The *ns* Manual, January 2002
- IEC *ns* workshop slides, June 2000
- First *ns* workshop slides, September 1997
- Wetherall and Lindblad, "Extending Tcl for Dynamic Object-Oriented Programming," Proceedings of the Tcl/Tk Workshop, 1995
- Welch, "Practical Programming in Tcl and Tk", Prentice-Hall, 1995
- Ousterhout, "Tcl and the Tk Toolkit," Addison-Wesley, 1994