NS-2 Tutorial

Part II: Interface: OTCL linkage

In the last talk

- We’ve talked about
  - TCL grammar
  - Basic network simulation
  - Network-wide trace
  - Tips on running simulation

In this talk

- We'll talk about
  - Creating a network component
  - Access OTCL from C++
  - Access C++ from OTCL
  - Class hierarchy and adding a network component
  - Tracing

Creating a network component

Network components?

- *% cat basic_udp.tcl*
  - *set ns [new **Simulator**]*
  - *set n0 [$ns node]*
    set n1 [$ns node]
    *$ns duplex-link $n0 $n1 1Mb 2ms **DropTail***
  - *set udp0 [new **Agent/UDP**]*
    $ns attach-agent $n0 $udp0
    *set cbr0 [new **Application/Traffic/CBR**]*
    $cbr0 attach-agent $udp0
  - *set null0 [new **Agent/Null**]*
    $ns attach-agent $n1 $null0
  - *$ns connect $udp0 $null0*
  - *$ns at 1.0 “$cbr0 start”*
    $ns run

Create new components

- OTCL
- C++
  - With OTCL linkage

Creating pure OTCL class

- Add a new OTCL component
  - Create *new_otcl.tcl*
  - Source the file in *$ns/tcl/lib/ns-lib.tcl*
  - Add the file to NS_TCL_LIB in *Makefile*
  - Recompile the whole ns

Description of OTCL class

- *% cat new_otcl.tcl*
  - *Class NewOtcl*
  - *NewOtcl instproc init {} {*
    $self greeting }
  - *NewOtcl instproc greeting {} {*
    puts “Hello, new OTCL class is created.”}
  - *NewOtcl instproc say {} {*
    puts “This is a dummy class.”}

Compile procedure

- Move *new_otcl.tcl* to *$ns/tcl/lib*
- Add the following in *$ns/tcl/lib/ns-lib.tcl*
  - *source new_otcl.tcl*
- Add the following in *$ns/Makefile*
  - *NS_TCL_LIB =…*
    tcl/lib/new_otcl.tcl
- Recompile
  - *% cd $ns; make*

Running example

- In the interpreter mode
  - *% set a [new NewOtcl]*
    Hello, new OTCL class is created.
    - Function *greeting* is called by *init*
  - *% $a say*
    This is a dummy class.

C++ class?

- Most of components are written in C++
  - Not pure OTCL class
  - OTCL is slower than C++
- How to connect two different spaces?
  - Usually, the user starts from OTCL
  - How to control components written in C++?

OTCL Linkage

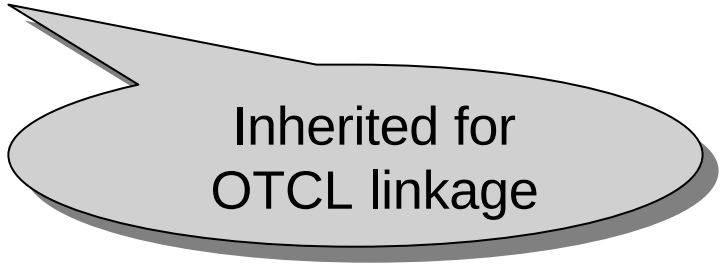
- Important OTCL linkages
  - Class TclClass
  - Class Tcl
  - Class TclObject
- Other linkages
  - Class TclCommand
  - Class EmbeddedTcl
  - Class InstVar

Creating C++ class

- Create a new C++ component
  - Create *new_cpp.{h,cc}*
  - Add the file to OBJ_CC in *Makefile*
  - Recompile the whole ns

Description of C++ class

- *% cat new_cpp.cc*
  - *#include “config.h”*
  - *class NewCpp : public TclObject {*
    public:
    NewCpp();
    };
  - *NewCpp::NewCpp() {*
    printf(“Hello, new C++ class is created.\n”);
    }



Hook for OTCL linkage

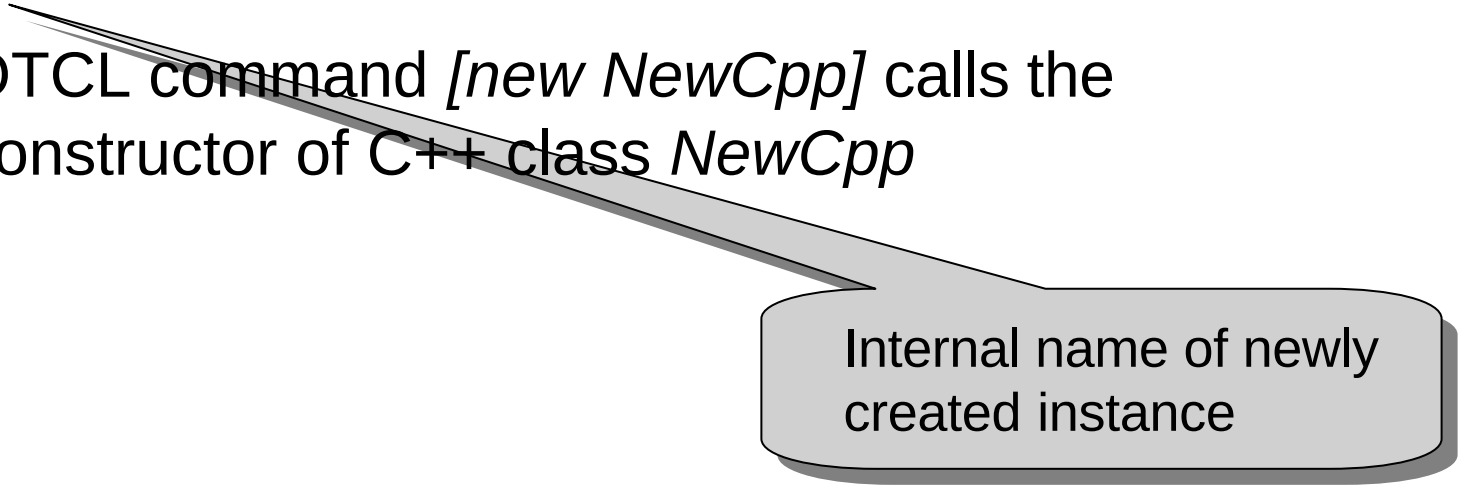
- After class declaration insert the following
  - *static class NewCppClass : public TclClass {*
    public:
    *NewCppClass() : TclClass("**NewCpp**") {}*
    TclObject create(int, const char*const*) {*
    return (new NewCpp());
    }
    } class_newcpp;

Compile procedure

- Move *new_cpp.cc* to *$ns*
- Add the following in *$ns/Makefile*
  - *OBJ_CC = …*
    new_cpp.o
- Recompile
  - *% cd $ns; make*

Running example

- In the interpreter mode
  - *% set a [new NewCpp]*
    Hello, new C++ class is created.
    _o4
    - OTCL command *[new NewCpp]* calls the constructor of C++ class *NewCpp*



Class TclClass

- C++ virtual class
  - Construct interpreted class to mirror compiled class
  - Method to instantiate new compiled objects

Revisit TclClass codes

- *static class NewCppClass : public TclClass {*
  public:
  NewCppClass() : TclClass("NewCpp") {}
  TclObject create(int, const char*const*) {*
  return (new NewCpp());
  }
  } class_newcpp;

Class name in OTCL, not necessary to be the same name of C++ class

Invoke constructor of TclClass

Instantiate new object by calling its constructor

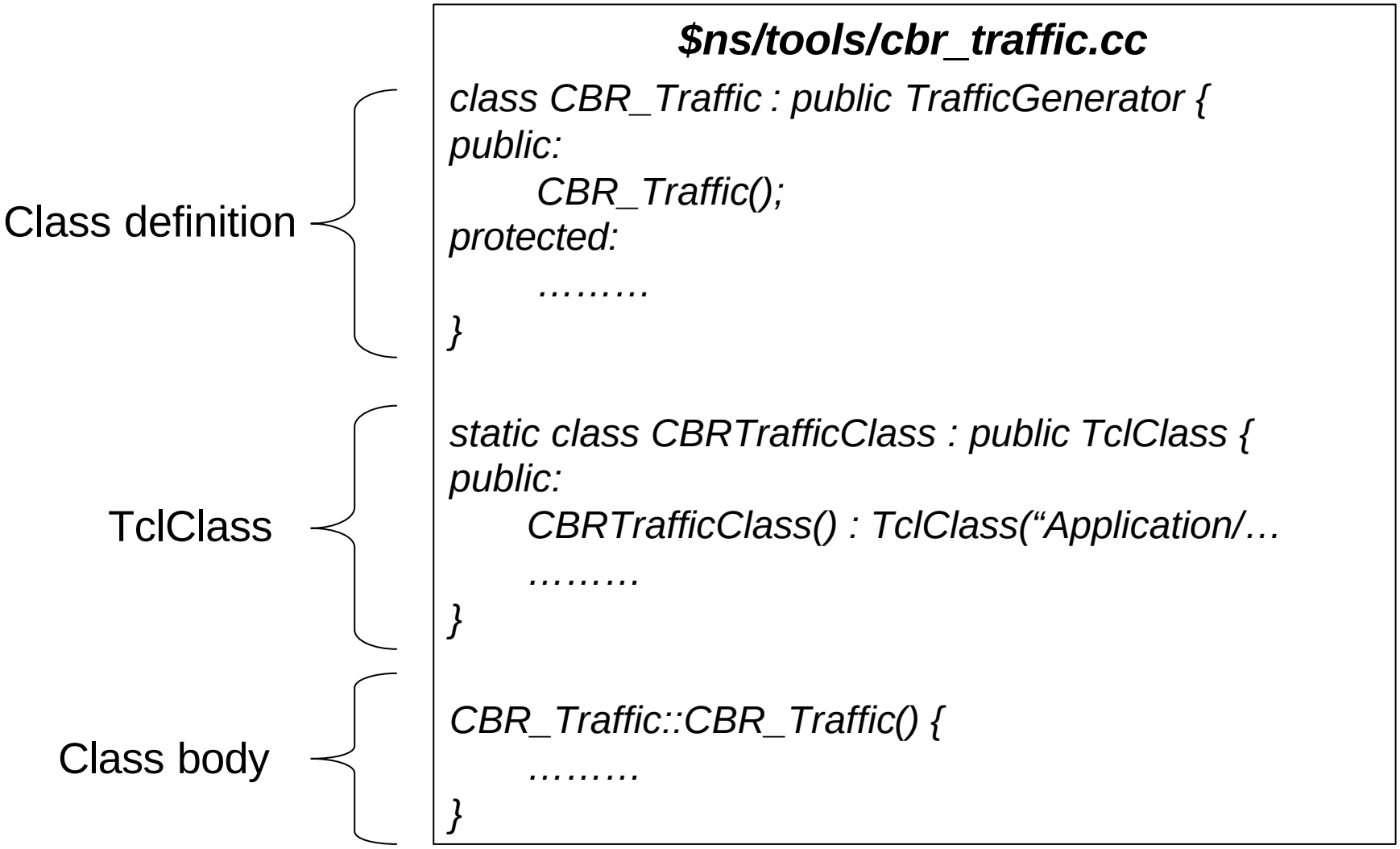
Basic structure of C++ elements

Class definition

Define **TclClass**
-- Interface to OTCL

Class body

Example of C++ elements (CBR Traffic)



Access to OTCL from C++

Communication channel

- Class Tcl
  - Defined in *$ns/../tclcl-1.15/tclcl.h*
  - One Tcl instance is created at the start-up
    - *$ns/../tclcl-1.15/Tcl.cc*

      ……

      Tcl Tcl::instance_;

Important functions

- Obtain reference to the Tcl instance
  - *Tcl& tcl = Tcl::instance();*
- Invoking OTCL procedures
  - *tcl.eval(char*)*
  - *tcl.evalc(const char*)*
  - *tcl.evalf("formatted string…", var1, var2,…)*
- Passing results to/from OTCL
  - *tcl.result(const char*)*
  - *tcl.result()*

Example – tcl.eval

- Modify the constructor in *new_cpp.cc*
  - *% cat new_cpp_tcleval.cc*
  - *NewCpp::NewCpp() {*
    printf("Hello, new C++ class is created.\n");
    Tcl& tcl = Tcl::instance();
    tcl.eval("puts \"We used Class Tcl.\"\n");
    }

- Compile, then
  - *% set a [new NewCpp]*
    Hello, new C++ class is created.
    We used Class Tcl.
    _o4

Example – tcl.result

- Modify the constructor in *new_cpp.cc*
  - *% cat new_cpp_tclresult.cc*
  - *NewCpp::NewCpp() {*

```
float version;
printf("Hello, new C++ class is created.\n");
Tcl& tcl = Tcl::instance();
tcl.eval("ns-version");
version = atof(tcl.result());
printf("The version of this NS-2 is %f\n",version);
}
```

- Compile, then
  - *% set a [new NewCpp]*
    Hello, new C++ class is created.
    The version of this NS-2 is 2.270000.
    _o4

Access C++ from OTCL

OTCL Linkages that we talked about

- Class TclClass
  - Create new (C++) component from OTCL
- Class Tcl
  - Access OTCL from C++

Access C++ from OTCL?

- What do we want to do?
  - Set a variable of C++ component from OTCL
  - Trace a variable of C++ component
  - Execute a function of C++ component from OTCL

Class TclObject

- Variable binding
  - Set/get variable of C++ component from/to OTCL
- Command method
  - Give control to C++

Variable binding

- Special function "*bind*"
  - Show a C++ variable to OTCL
  - Defined in *$ns/../tclcl-1.15/Tcl.cc*
    - Create OTCL variable instance
    - Create an instance of *class InstVar* (bound with the OTCL variable instance)
    - Initialize OTCL variable instance

Function 'bind'

- Interface
  - *bind (const char* var, double* val)*
  - *bind_bw (cont char* var, double* val)*
    - For bandwidth, understand '*M*', '*K*',…
  - *bind_time (const char* var, double)*
    - For time, understand '*ms*', '*us*',…
  - *bind_bool (const char* var, int)*
    - For boolean, understand '*T*', '*F*',…

Usually, binding in the constructor

- *% cat new_cpp_bind.cc*
  - *class NewCpp : public TclObjecdt {*

    …….

    int store_int_;

    };
  - *NewCpp::NewCpp() {*

    printf(“Hello, new C++ class is created.\n”);

    bind(“store_”, &store_int_);

    }

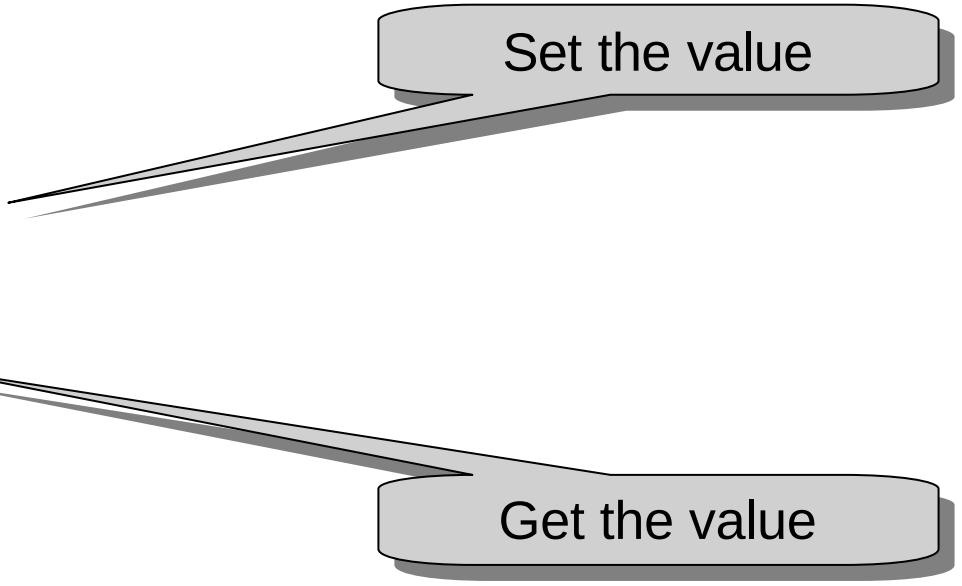
Set/get the variable from OTCL

- Compile, then
  - *% set a [new NewCpp]*
    Hello, new C++ class is created.
    *warning: no class variable NewCpp::store_*

    see tcl-object.tcl in tclcl for info about this warning.

_o4

% $a set store_ 10
10
*% $a set store_*
10



Set default value

- Warning
  - The bound variable does not have the default value
- Remove warning
  - Define the default value in *$ns/tcl/lib/ns-default.tcl*
  - *……*
    NewCpp set store_ 1
  - Then, recompile

Summary for binding

- binding
  - Set/get a C++ variable from/to OTCL
- Procedure
  - Call ‘bind’ in the constructor
    - *bind(“store_”, &store_int_)*
  - Set default value in *$ns/tcl/lib/ns-default.tcl*
    - *NewCpp set store_ 1*
  - Recompile

Command method

- Hand control to C++
  - Execute C++ function
  - Set/get C++ variables
  - Communicate with OTCL using Class Tcl

Overriding “command”

- *% cat new_cpp_command.cc*
  - *class NewCpp : public TclObject {*

    ……

    int store_int_;

    *int command(int argc, const char*const* argv);*

    };

  - First defined in *class TclObject*

- *int NewCpp::command(int argc, const char*const* argv) {*

```
    if (argc == 2) {
        ……
    }
    else if (argc == 3) {
        if(strcmp(argv[1], “store”) == 0) {
                store_int_ = atoi(argv[2]);
                return (TCL_OK);
        }
    }
    return (TclObject::command(argc, argv));
}
```

Should return TCL_OK or TCL_ERROR

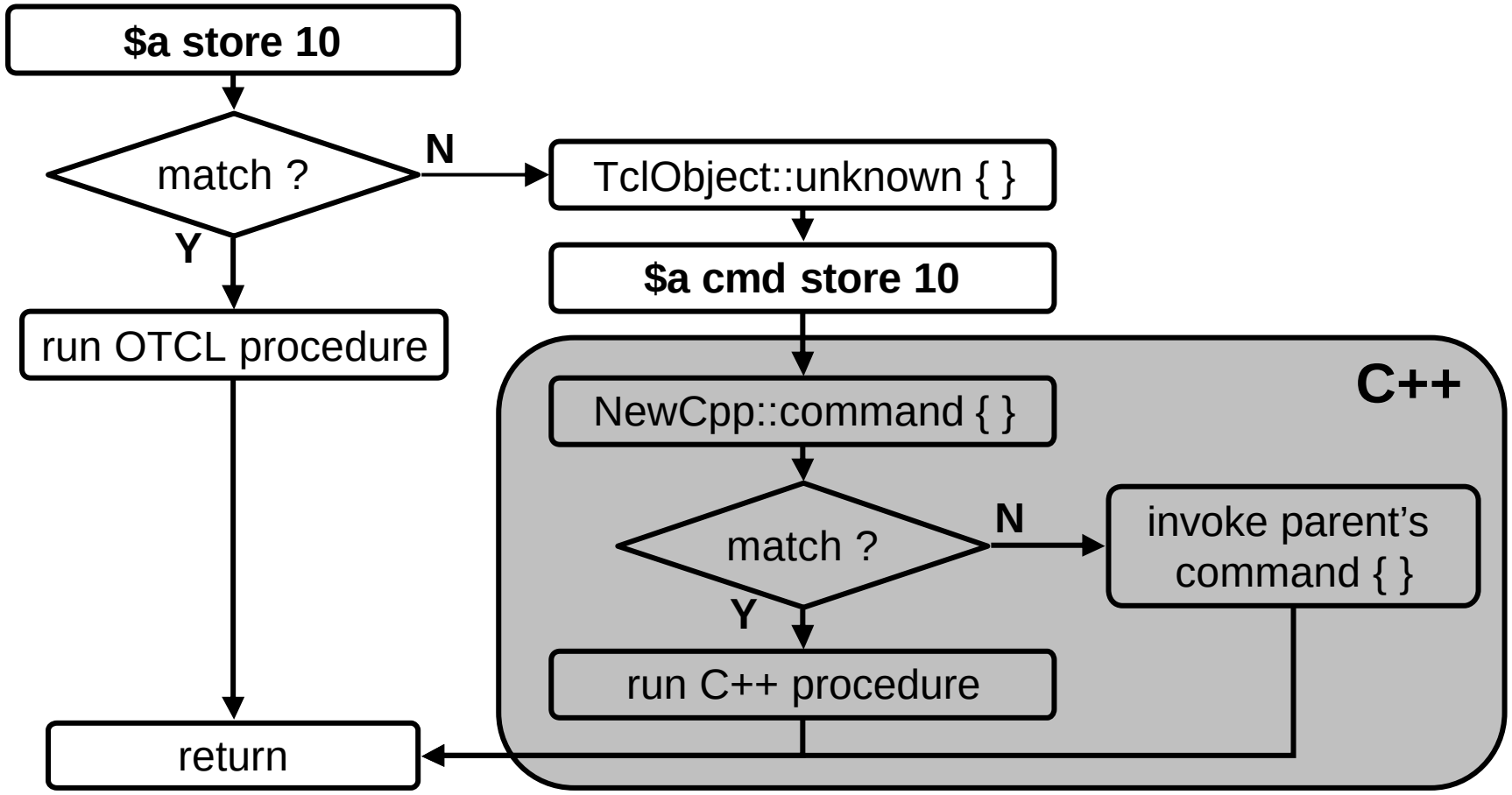
If not found, check parent class

Using command method

- Compile, then
  - *% set a [new NewCpp]*
    Hello, new C++ class is created.
    _o4
    % $a store 10
    % $a store
    10

  - No default value
  - Can do multiple jobs
  - Can execute C++ function

Common path of command



Summary of OTCL linkage

- We talked about
  - Class TclClass
    - Create a new C++ component
  - Class Tcl
    - Access OTCL from C++
    - Using OTCL instance *Tcl::instance()*
  - Class TclObject
    - Access C++ from OTCL
    - Binding variable and command method

Revisit basic structure of C++ elements

Class definition

Define **TclClass**
Connection to “constructor”

Class body

TclObject: binding method
Connection to “variables”

TclObject: command method
Connection to “functions”

Other OTCL linkages

- Class TclCommand
  - For top-level command such as *ns-version*
- Class EmbeddedTcl
  - We did this ! (in pure OTCL class)
- Class InstVar
  - Internal interface used for binding variable

Class TclCommand

- Used for top-level commands
  - such as *ns-version, ns-random*
  - Defined in *$ns/common/misc.cc*
    - other top-level commands
    - You can put a new top-level command
  - "*say_hello*" example in the ns-document

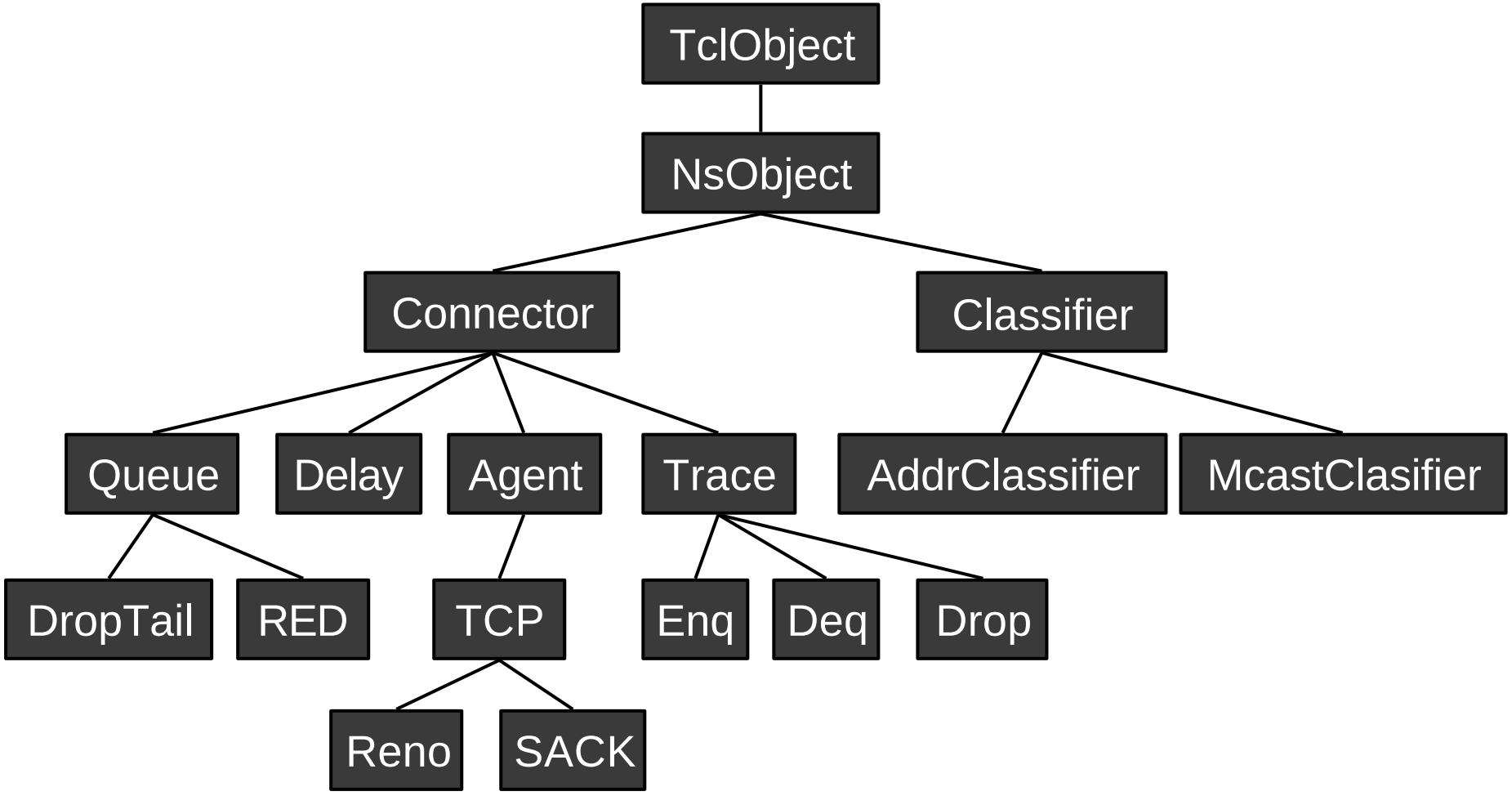
Add in $ns/common/misc

- *class say_hello : public TclCommand {*
  public:
  say_hello() : TclCommand("hi") {}
  *int command(int argc, const char*const* argv) {*
  printf("Hello World\n");
  return TCL_OK;
  }
  };
- *void init_misc(void) { …*
  (void) new say_hello;
  }

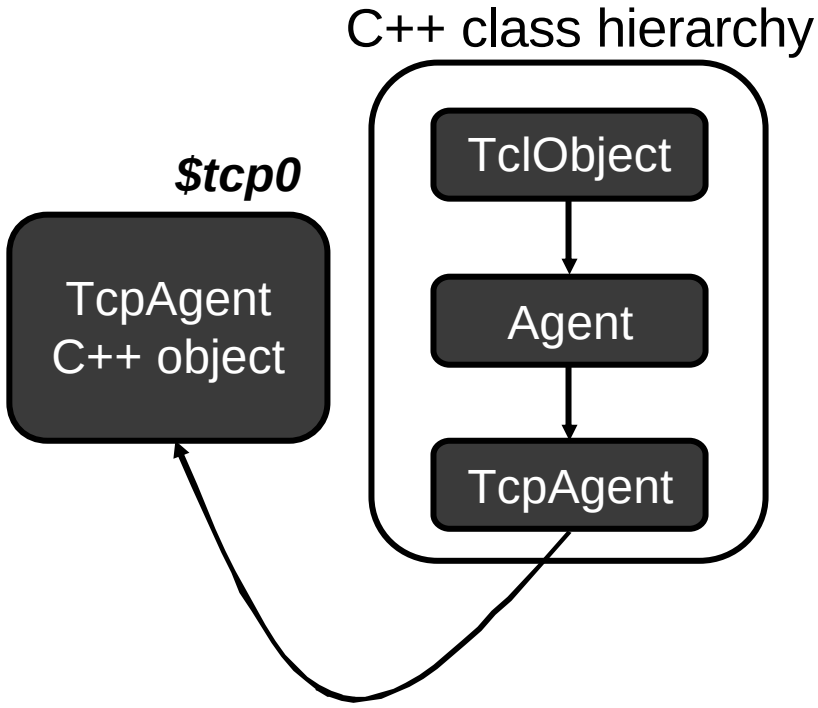
- Compile, then
  - *% hi*
    Hello, World

Class hierarchy and adding a network component

Hierarchical structure of C++ class



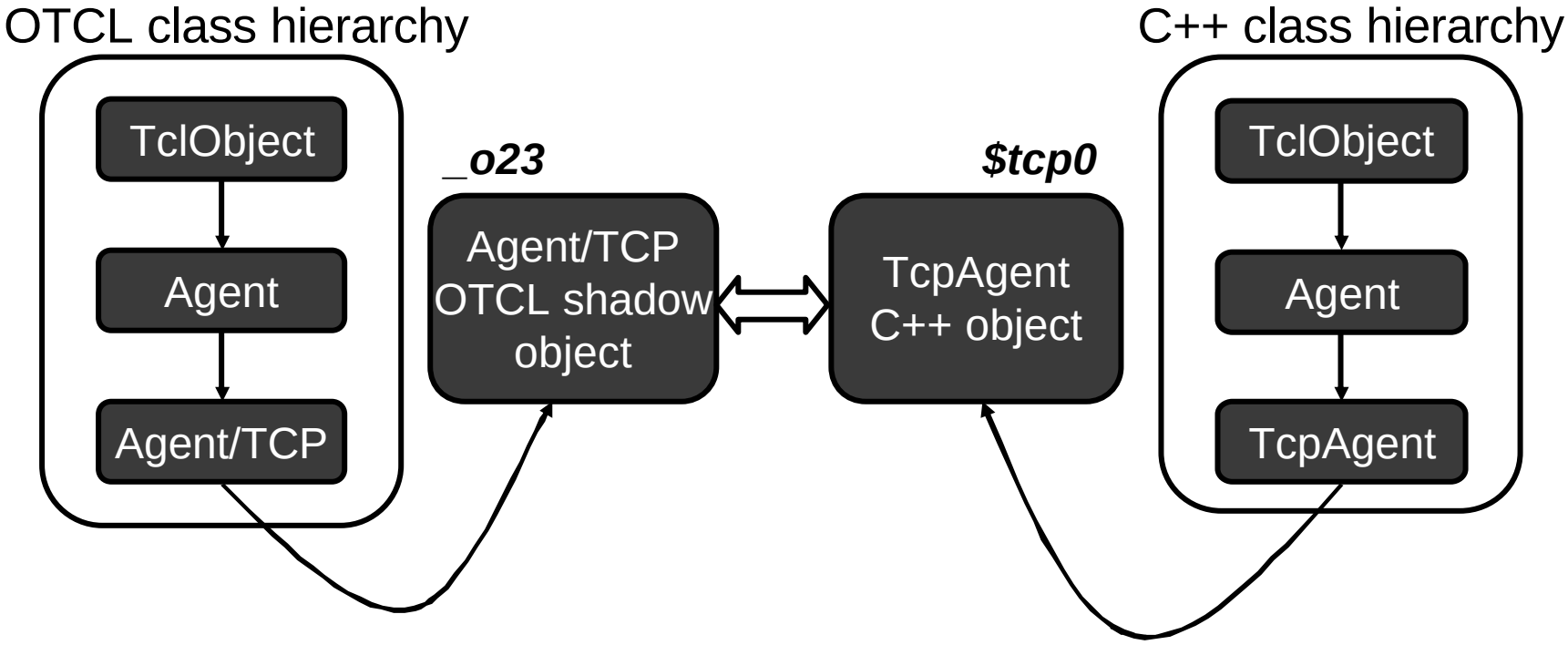
C++ class hierarchy



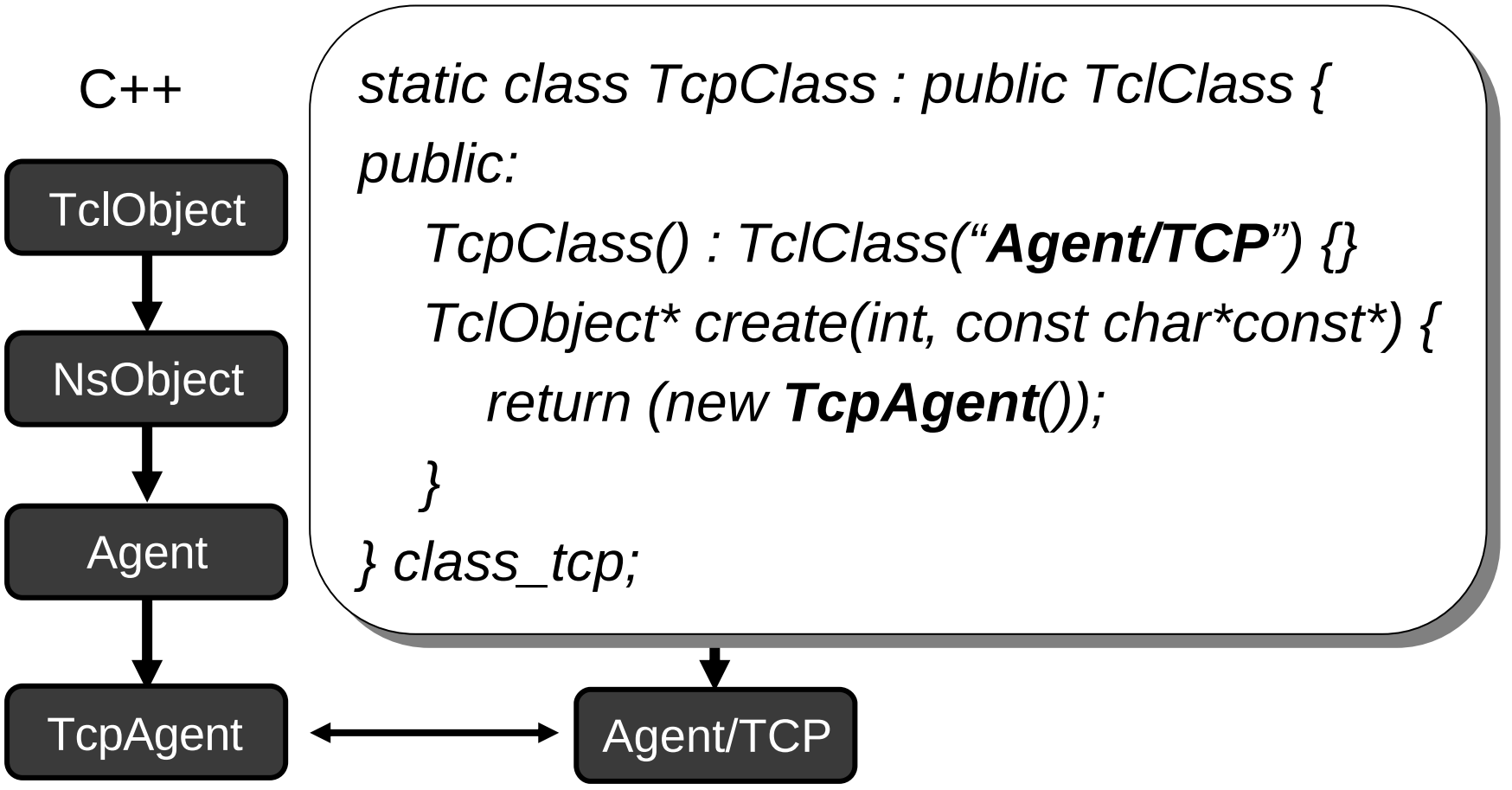
Split object

- Shadowing
  - When an object in the class TclObject is created by the user from within the interpreter, an equivalent shadow object is created.
  - Two objects are closely associated
    - Using Class TclObject and Class TclClass

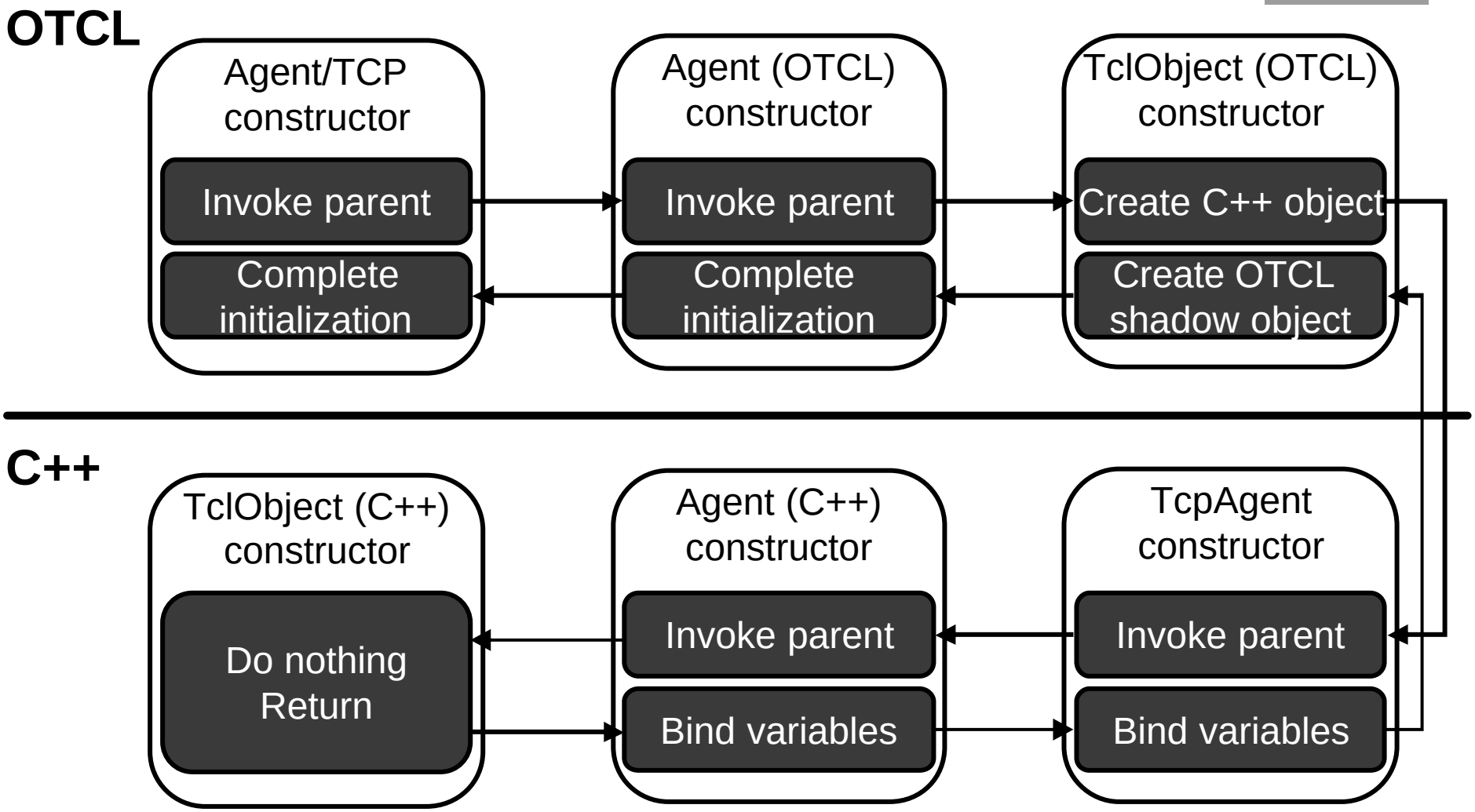
Shadowing


OTCL class hierarchy
TclObject
Agent
Agent/TCP
_o23
Agent/TCP
OTCL shadow
object
$tcp0
TcpAgent
C++ object
C++ class hierarchy
TclObject
Agent
TcpAgent

Shadowing with hierarchy



Creating objects



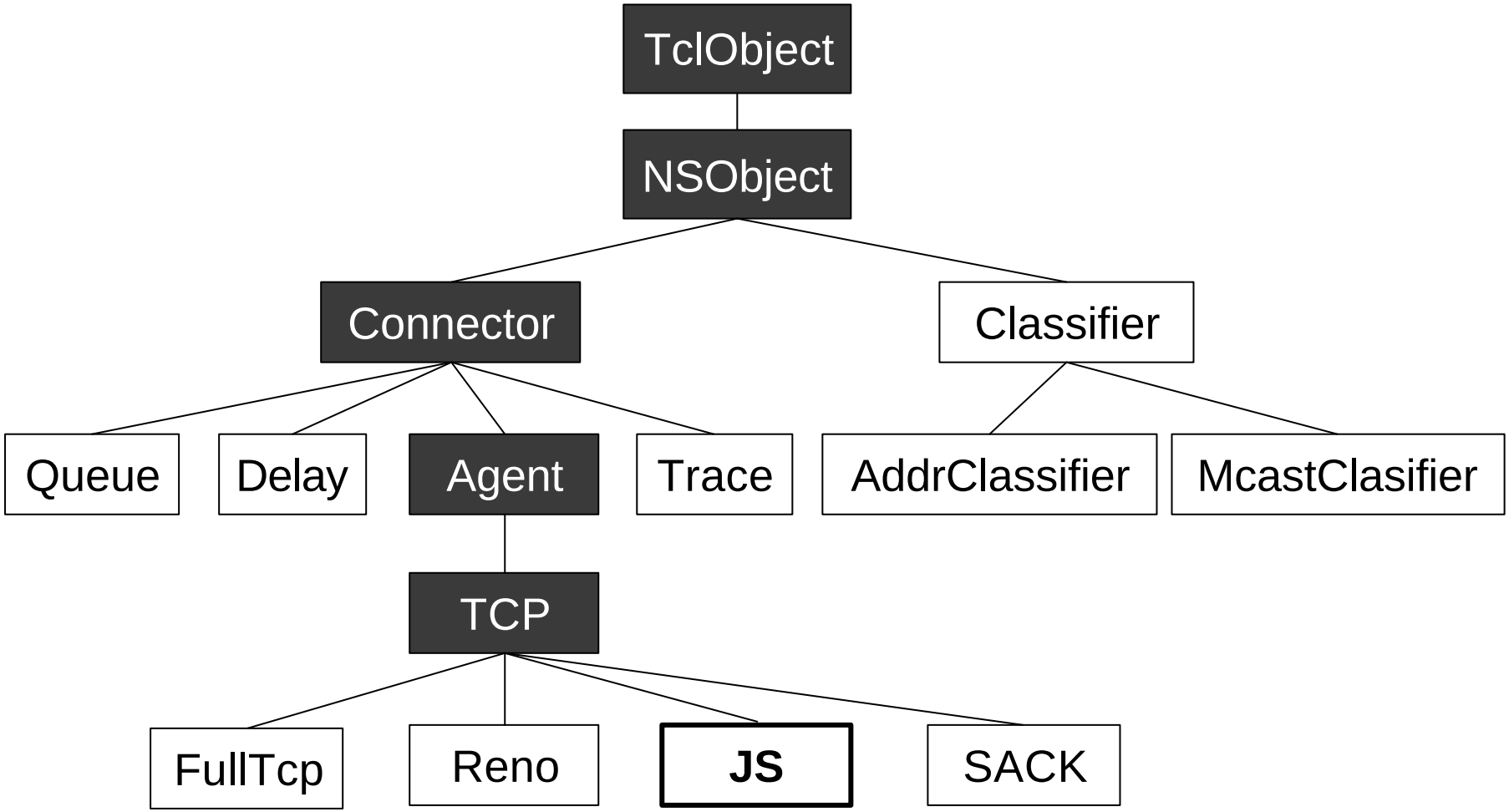
Adding network component

- Procedure
  - Decide position in hierarchy
  - Create new packet header if necessary
  - Create C++ class inheriting parent
  - Define OTCL linkage
  - Write OTCL code if necessary
  - Build (compile) and use it

Create a New TCP Variant

- TCP Jump Start Example
  - TCP large initial window option
  - Shown in Padma Haldar's presentation
  - Purpose
    - Wide-open transmission window at start-up
    - Change ***cwnd_ = 1*** to ***cwnd_ = MAXWIN_***

Position in hierarchy



Declaration

- *% cat $ns/tcp/tcp-js.cc*
  - *include “tcp.h”*
  - *class JSTcpAgent : public TcpAgent {*
    public:
    JSTcpAgent();
    virtual void set_initial_window();
    // override function in TcpAgent
    private:
    int maxwin_;
    }
- Can be separated into *tcp-js.h*

Linkage using Class TclClass

- *% cat $ns/tcp/tcp-js.cc*
  - [Declaration]
  - *static class JSTcpClass : public TclClass {*
    public:
    JSTcpClass():TclClass("Agent/TCP/JS") { }
    TclObject create(int, const char* const*) {*
    return (new JSTcpAgent());
    }
    } class_tcpjs;

Variable binding

- *% cat $ns/tcp/tcp-js.cc*
  - [Declaration]
  - [TclClass]
  - *JSTcpAgent::JSTcpAgent() {*
    bind("MAXWIN_", &maxwin_);
    }

Override “set_initial_window”

- Function in *$ns/tcp/tcp.cc*
  - *TcpAgent::set_initial_window() { cwnd_ = 1.0; }*

- *$ns/tcp/tcp-js.cc*
  - [Declaration]
  - [TclClass]
  - [Variable binding]
  - *void JSTcpAgent::set_initial_window() {*
    cwnd_ = maxwin_;
    }

Setting default value

- *% cat $ns/tcl/lib/ns-default.tcl*
  - *# Agent/TCP/JS*
    Agent/TCP/JS set MAXWIN_ 1

Modify Makefile

- *% cat $ns/Makefile*
  - *…*
    tcp/tcp.o ***tcp/tcp-js.o*** *tcp/tcp-sink.o*

Compile

- Move to *$ns*
  - *% make*

Using in OTCL

- *% cat basic_tcpjs_trace.tcl*
  - *set init_win [lindex $argv [expr [lsearch $argv ……]*
  - *set tcp [new **Agent/TCP/JS**]*
    $tcp set MAXWIN_ $init_win

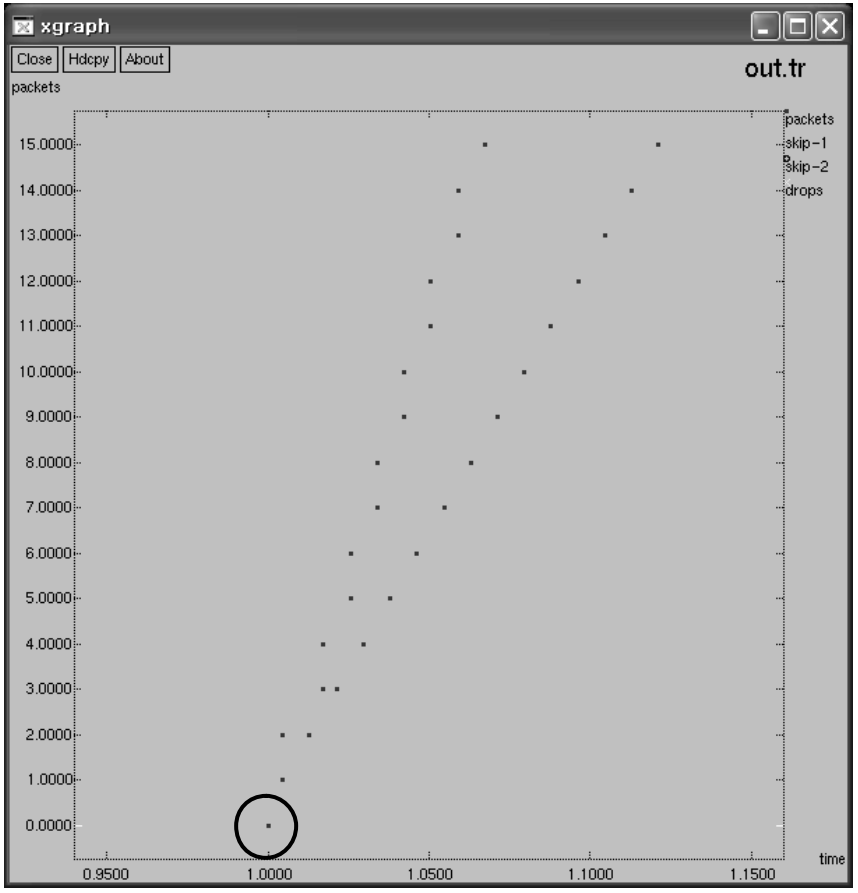
- Run
  - *% ns basic_tcpjs_trace.tcl –win 10*

Confirming result

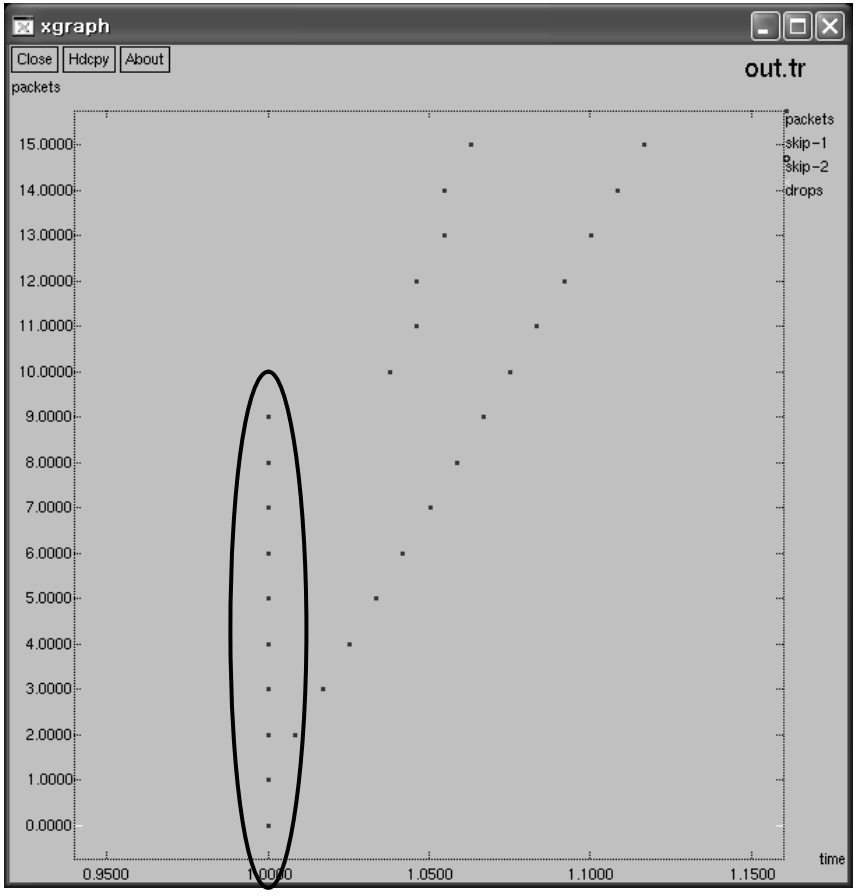
- Does it work?
  - Run simulations
  - Convert trace output using *raw2xg*
  - Compare using *xgraph*

Xgraph output

% ns basic_tcpjs_trace.tcl –win 1



% ns basic_tcpjs_trace.tcl –win 10



Tracing

Network-wide tracing

- Network-wide tracing
  - trace-all – tracing at all links
  - namtrace-all – tracing for animator
- Limitations
  - Our interests can be internal variables of object rather than packets on links
  - Network-wide tracings are time- and space-consuming

Tracing using bound variable

- Periodic probing in OTCL
- Variable Tracing support

Periodic probing

- Self-calling of OTCL procedure
- Variable should be visible in OTCL
  - Bound variable or OTCL variable

Example of periodic probing

- *% cat tcp_probe1.tcl*
  - *proc probe { } {*

```
global ns tcp0
set now [$ns now]
set cwnd [$tcp0 set cwnd_ ]
puts “congestion window = $cwnd (time = $now)”

$ns at [expr $now + 1] “probe”
```

    }
  - *$ns at 1.001 “probe”*

Bound variable in tcp.cc

Self-calling

Initial start-up

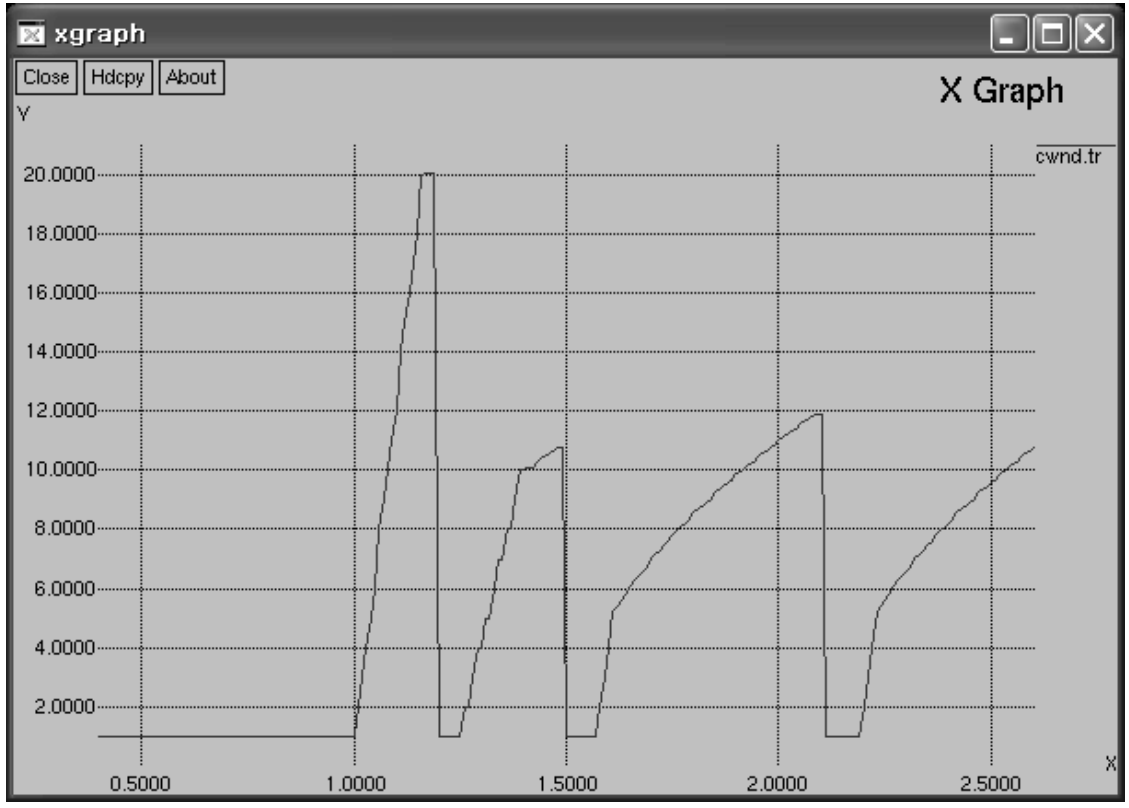
Running

- *% ns tcp_probe1.tcl*
  congestion window = 1 (time = 1.0009999)
  congestion window = 24.5399 (time = 2.0009)
  congestion window = 29.0237 (time = 3.0009)
  congestion window = 32.9013 (time = 4.0009)

Generate xgraph output

- *tcp_probe2.tcl*
  - Interval of probing = 0.01 sec
  - Xgraph output
    - *puts "$now $cwnd"*
  - Reduce queue size to cause packet drops
- Running
  - *% ns tcp_probe2.tcl > cwnd.tr*

- *% xgraph cwnd.tr*



Variable Tracing support

- Automatically record whenever the value of traced variable changes
  - Variable must be visible in OTCL
  - Variable must belongs to trace class
    - *TracedVar, TracedInt, TracedDouble*
  - Variable must be traced by general or its own tracer

Required in C++ class

- Declaration in trace classes
  - Trace classes are defined in *$ns/../tclcl-1.15/tracedvar.{h,cc}*
  - *TracedVar, TracedInt, TracedDouble*

- TCP *cwnd_* example
  - *% cat $ns/tcp/tcp.h | grep cwnd_*

    ……

    *TracedDouble cwnd_*

- Variable should be bound
  - To be visible in OTCL

- TCP *cwnd_* example
  - *% cat $ns/tcp/tcp.cc* | *grep cwnd_* | *grep bind*

    ……

    bind(“cwnd_”, &cwnd_);

Using Tracer

- Class Tracer
  - Defined in *$ns/trace/trace.{h,cc}*
- TCP *cwnd_* example
  - *% cat tcp_tracer.tcl*
  - *set tracer_ [new Trace/Var]*
    $tracer_ attach [open cwnd.raw.tr w]
    *$tcp0 trace cwnd_ $tracer_*

Running example

- *% ns tcp_tracer.tcl*
- *% tail cwnd.raw.tr*

  ……
  f t4.9912 a_o28 ncwnd_ v9.78154
  f t4.99952 a_o28 ncwnd_ v9.88378

  - *f*: trace type (Trace/Var)
  - *t*: time
  - *a*: name of owner of trace
  - *n*: name of traced variable
  - *v*: value

Converting to xgraph format

- *% cat traceout.tcl*
  - *set tracefile [open cwnd.raw.tr r]*

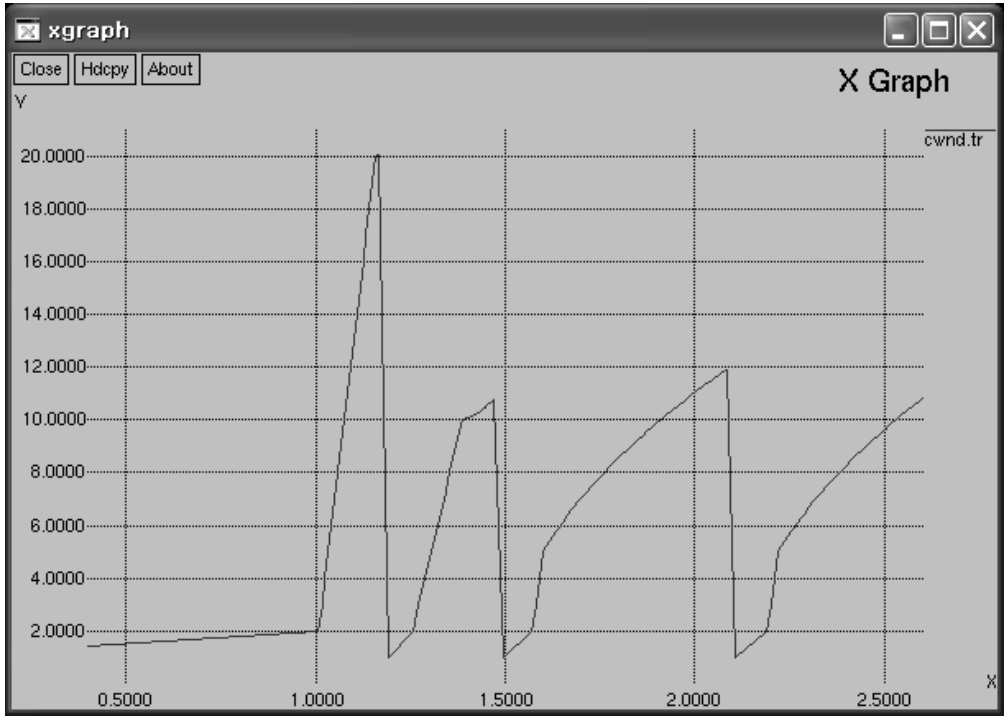
```
while {[gets $tracefile line]} {
    if {[eof $tracefile] == 1} {
        break
    }
    set time [lindex $line 1]
    set value [lindex $line 4]
    puts “[string range $time 1 [string length $time]]
  [string range $value 1 [string length $value]]”
}
close $tracefile
```

Converting output

- *% ns traceout.tcl*

  ……..
  4.9912 9.78154
  4.99952 9.88378

- *% ns traceout.tcl > cwnd.tr*
- *% xgraph cwnd.tr &*

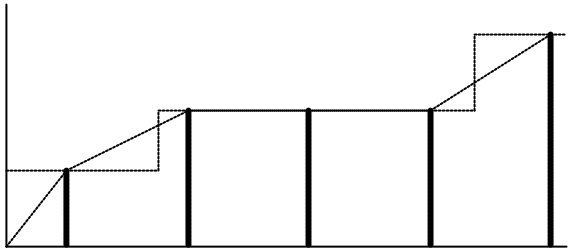


Summary of Variable Tracing

- In C++
  - Declare the variable in trace class
  - Bind the variable
- In OTCL
  - Create a tracer instance of Trace/Var
  - Attach the tracer to file for output
  - Place trace command
- After running
  - Convert the output accordingly

Periodic probing vs. Variable tracing

- Periodic probing
  - Sample the variable
  - Controlled interval
  - Good for time averaging



- Variable tracing
  - Show detailed changes
  - May be time/space-consuming
  - Good for event averaging