Corso di Reti di Calcolatori I

Network simulation with ns-3

Outline of presentation

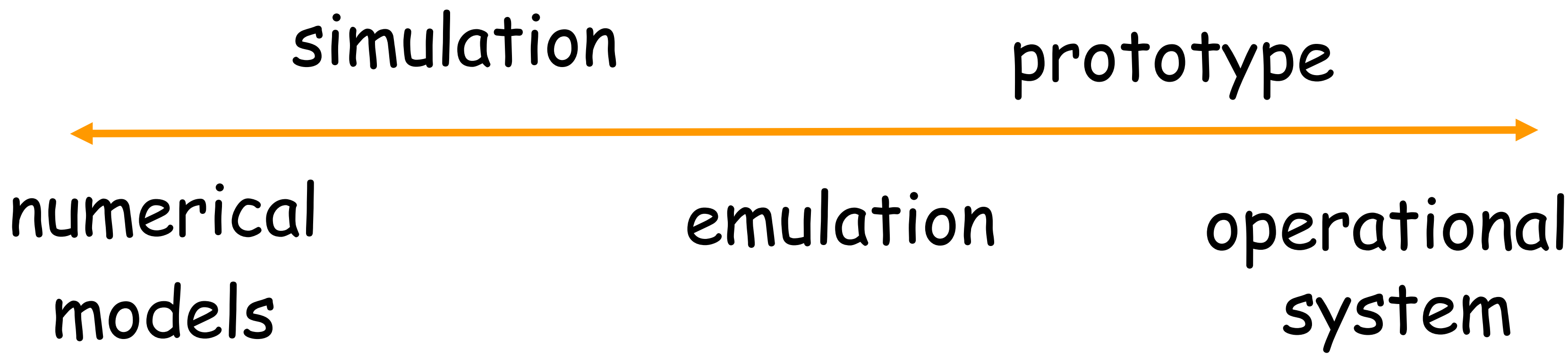
- Breve introduzione al concetto di simulazione di computer networks
- Tutorial su ns-3
  - Introduzione
  - Architettura
  - Lettura di codice

Simulation of computer networks

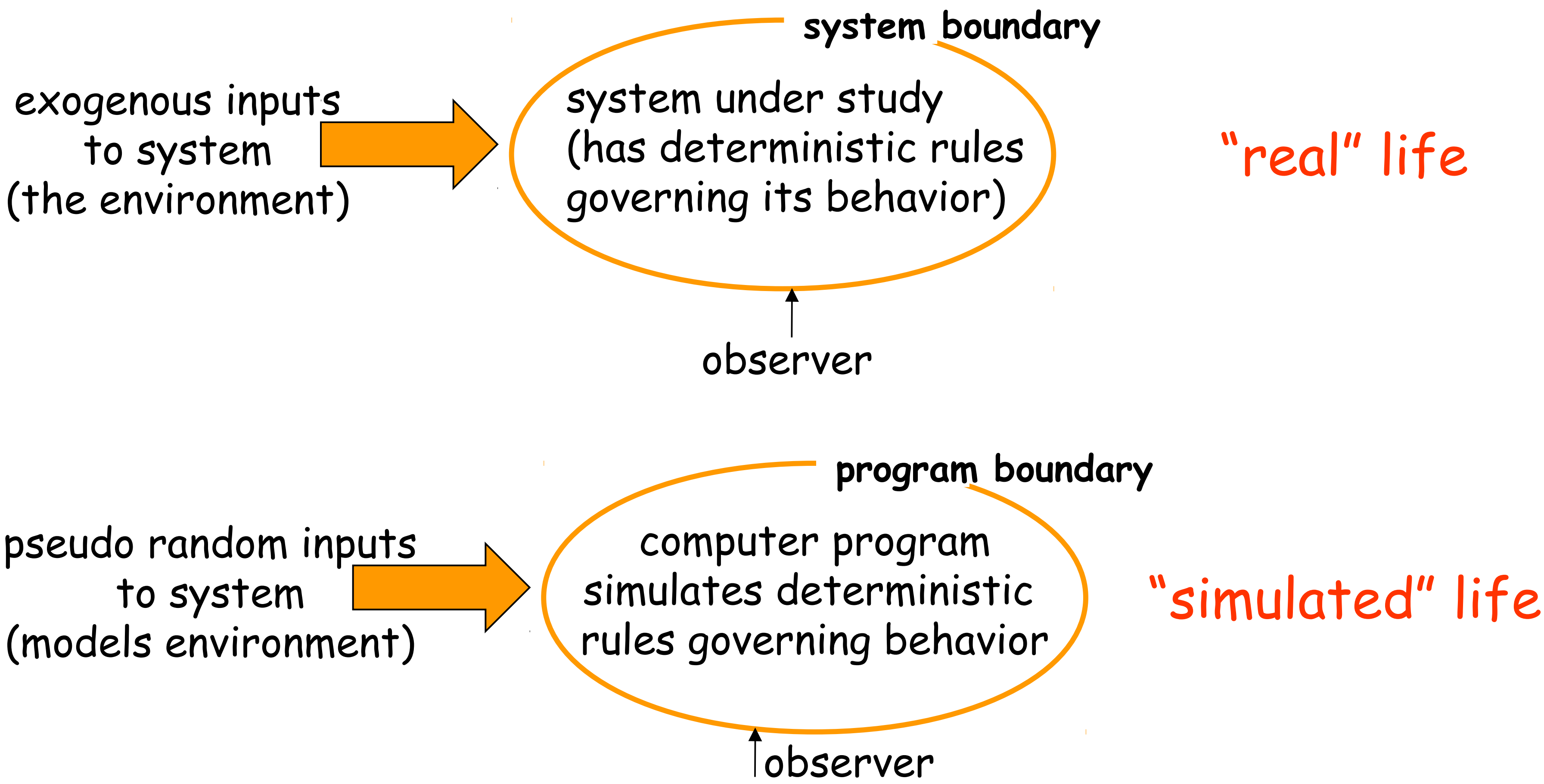
To evaluate protocols and distributed algorithms for computer networks the following alternatives are possible:

1) Small scale experimental testbeds (in the laboratory)
2) Testbed sperimentali su media scala (*wide area*), PlanetLab
3) Sistemi di network emulation
4) Ambienti di simulazione generali per reti di calcolatori
   - ns-3, GLOMOSIM, OPNET, NCTUns, …
1) Strumenti di simulazione sviluppati *ad hoc*
2) Modelli matematici del sistema

The evaluation spectrum



What is simulation?



Why Simulation?

- *goal:* study system *performance, operation*
- real-system not *available, is complex/costly or dangerous* (eg: space simulations, flight simulations)
- quickly evaluate design *alternatives* (eg: different system configurations)
- evaluate *complex functions* for which closed form formulas or numerical techniques not available
- Need of complete control over the inputs of the system

Requisiti per un simulatore di reti

- Astrazione
- Generazione di scenari (topologie, pattern di traffico, ...)
- Programmabilità
- Estendibilità
- Disponibilità di un’ampia gamma di moduli di protocolli riutilizzabili, affidabili e validati
- Possibilità di modificare protocolli esistenti
- Visualizzazione dei risultati
- Emulazione

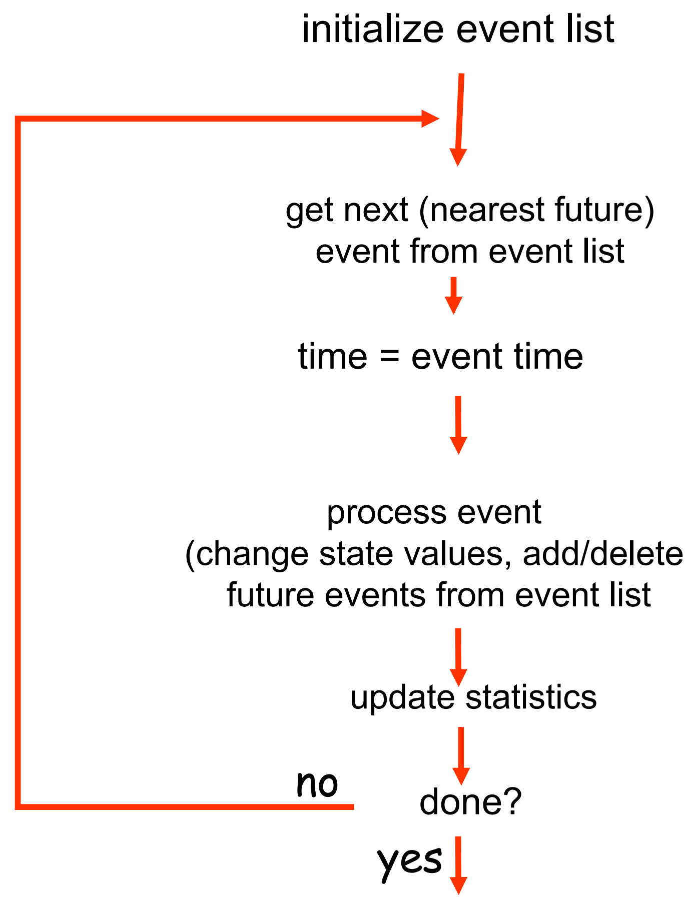
Programming a simulation

What 's in a simulation program?

- *simulated time:* internal (to simulation program) variable that keeps track of simulated time
- *system "state":* variables maintained by simulation program define system "state"
  - e.g., may track number (possibly order) of packets in queue, current value of retransmission timer
- *events:* points in time when system changes state
  - each event has associated *event time*
    - e.g., arrival of packet to queue, departure from queue
    - precisely at these points in time that simulation must take action (change state and may cause new future events)
  - model for time between events (probabilistic) caused by external environment

Discrete Event Simulation

- simulation program maintains and updates list of future events: event list
- simulator structure:



Need:

- well defined set of events
- for each event: simulated system action, updating of event list

Things to remember about Discrete Event Simulation

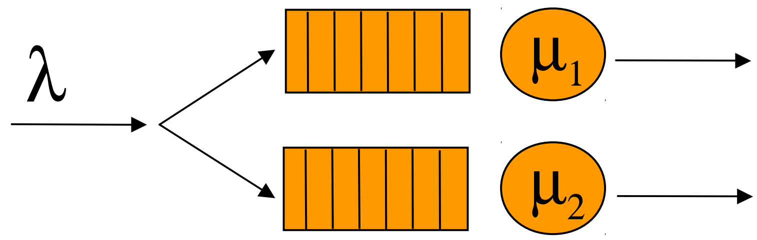
- The programming model revolves around "events" (eg: packet arrivals):
  - Events trigger particular sub-routines
    - Huge "switch" statement to classify events and call appropriate subroutine
    - The subroutine may schedule new events! (cannot schedule events for past, I.e., events are causal)
    - Rarely you might introduce new event *types*
- Events have associated with them:
  - Event type, event data structures (eg: packet)
  - Simulation time when the event is scheduled
- Key event operations: *Enqueue* (I.e. schedule a event)
  - Dequeue is handled by the simulation engine

Discrete Event Simulation: Scheduler

- Purpose: maintain a notion of simulation time, schedule events. A.k.a: “simulation engine”
- Simulation time ≠ Real time
  - A simulation for 5 sec of video transmission *might* take 1 hour!
- Events are sorted by simulation time (not by type!): priority queue or heap data structure
  - After all subroutines for an event have been executed, control is transferred to the simulation engine
  - The simulation engine schedules the next event available at the same time (if any)
  - Once all the events for current time have been executed, simulation time is advanced and nearest future event is executed.
  - Simulation time = time of currently executing event

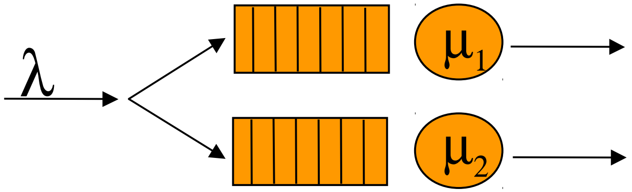
Simulation: example

- packets arrive (avg. interrarrival time: 1/ λ) to router (avg. execution time 1/μ) with two outgoing links
- arriving packet joins link i with probability ϕ_i



- state of system: size of each queue
- system events:
  - packet arrivals
  - service time completions

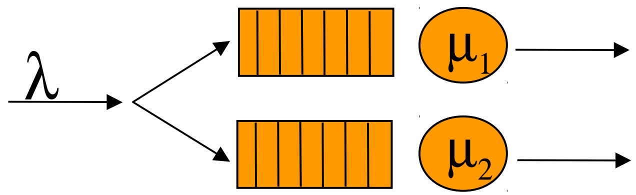
Simulation: example



Simulator actions on *arrival* event

- choose a link
  - if link idle {place pkt in service, determine service time (random number drawn from service time distribution) add future event onto event list for pkt transfer completion, set number of pkts in queue to 1}
  - if buffer full {increment # dropped packets, ignore arrival}
  - else increment number in queue where queued
- create event for next arrival (generate interarrival time) stick event on event list

Simulation: example



Simulator actions on *departure* event

- remove event, update simulation time, update performance statistics
- decrement counter of number of pkts in queue
- If (number of jobs in queue > 0) put next pkt into service – schedule completion event (generate service time for put)

Ns-3 tutorial: assumptions

- Some familiarity with C++ programming language
- Some familiarity with Unix Network Programming (e.g., sockets)
- Some familiarity with discrete-event simulators

Ns-3 features

- open source licensing (GNU GPLv2) and development model
- Python scripts or C++ programs
- alignment with real systems (sockets, device driver interfaces)
- alignment with input/output standards (pcap traces, ns-2 mobility scripts)
- testbed integration as a priority
- modular, documented core
- Easy to modify, extend
- Ns-3 is not an extension of ns-2

Resources

Web site:

http://www.nsnam.org

Mailing list:

http://mailman.isi.edu/mailman/listinfo/ns-developers

Tutorial:

http://www.nsnam.org/docs/tutorial/tutorial.html

Code server:

http://code.nsnam.org

Wiki:

http://www.nsnam.org/wiki/index.php/Main_Page

Resources (2)

- Most of the ns-3 API is documented with Doxygen

  http://www.nsnam.org/doxygen/index.html

Outline of the tutorial

- Introduction to ns-3
- Architecture of ns-3
- Ns-3 logging and tracing
- Ns-3 examples

Introduction to ns-3

- ns-3 is written in C++
- Bindings in Python
- ns-3 uses the waf build system
- simulation programs are either C++ executables or python scripts

the waf build system

- Waf is a Python-based framework for configuring, compiling and installing applications.
  - It is a replacement for other tools such as Autotools, Scons, CMake or Ant
  - http://code.google.com/p/waf/
- To download the simulator
  - hg clone http://code.nsnam.org/ns-3-allinone
- Steps to build the simulator
  - cd ns-3-dev
  - ./waf [-d optimized|debug] configure [--enable-modules=netanim]
  - make -> ./waf
  - make test -> ./waf check (run unit tests)

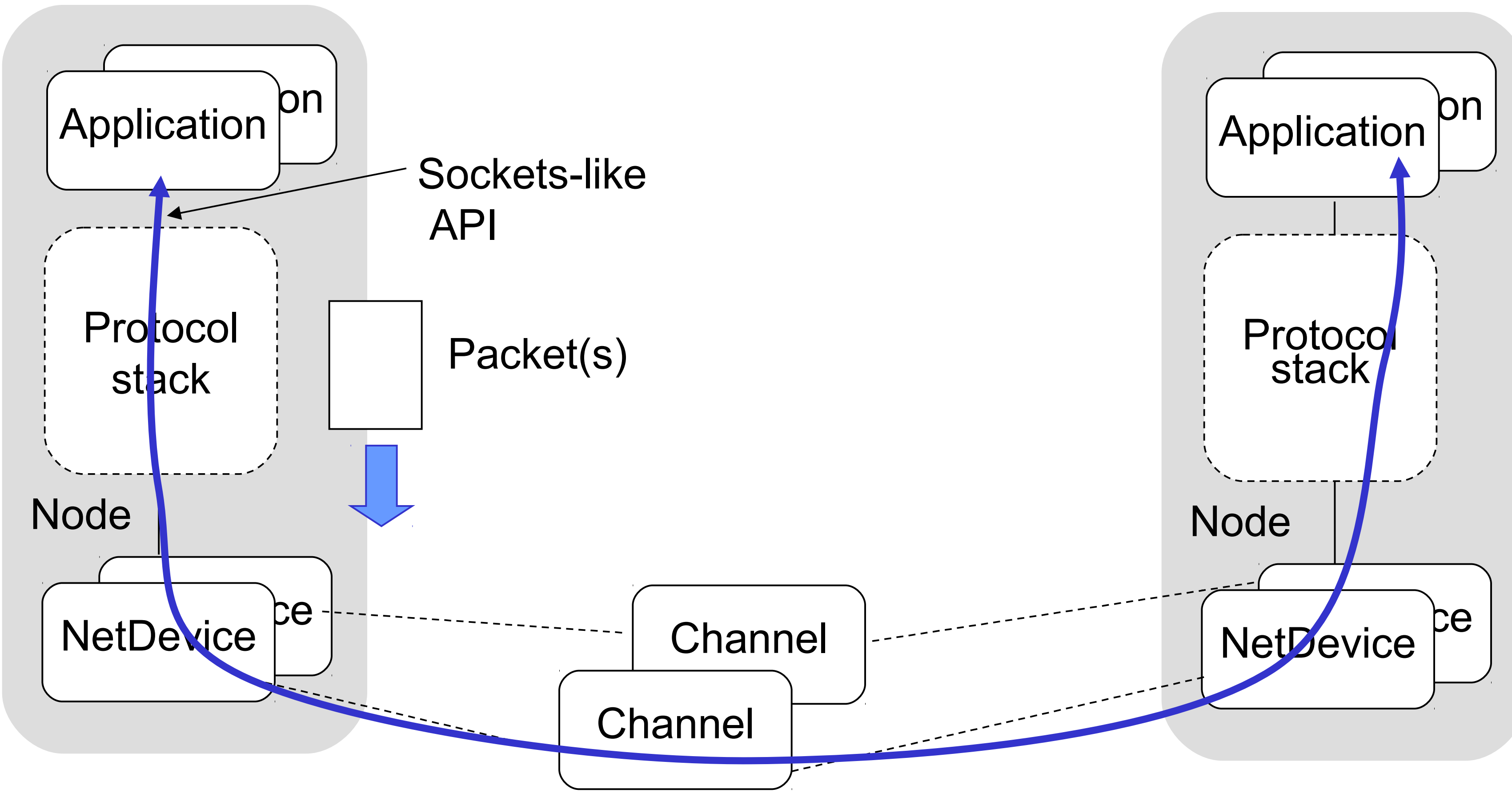
waf key concepts

- Can run programs through a special waf shell; e.g.
  - `./waf --run simple-point-to-point`
  - (this gets the library paths right for you)

Outline of the tutorial

- Introduction to ns-3
- Architecture of ns-3
- Ns-3 logging and tracing
- Ns-3 examples

Architecture: the basic model



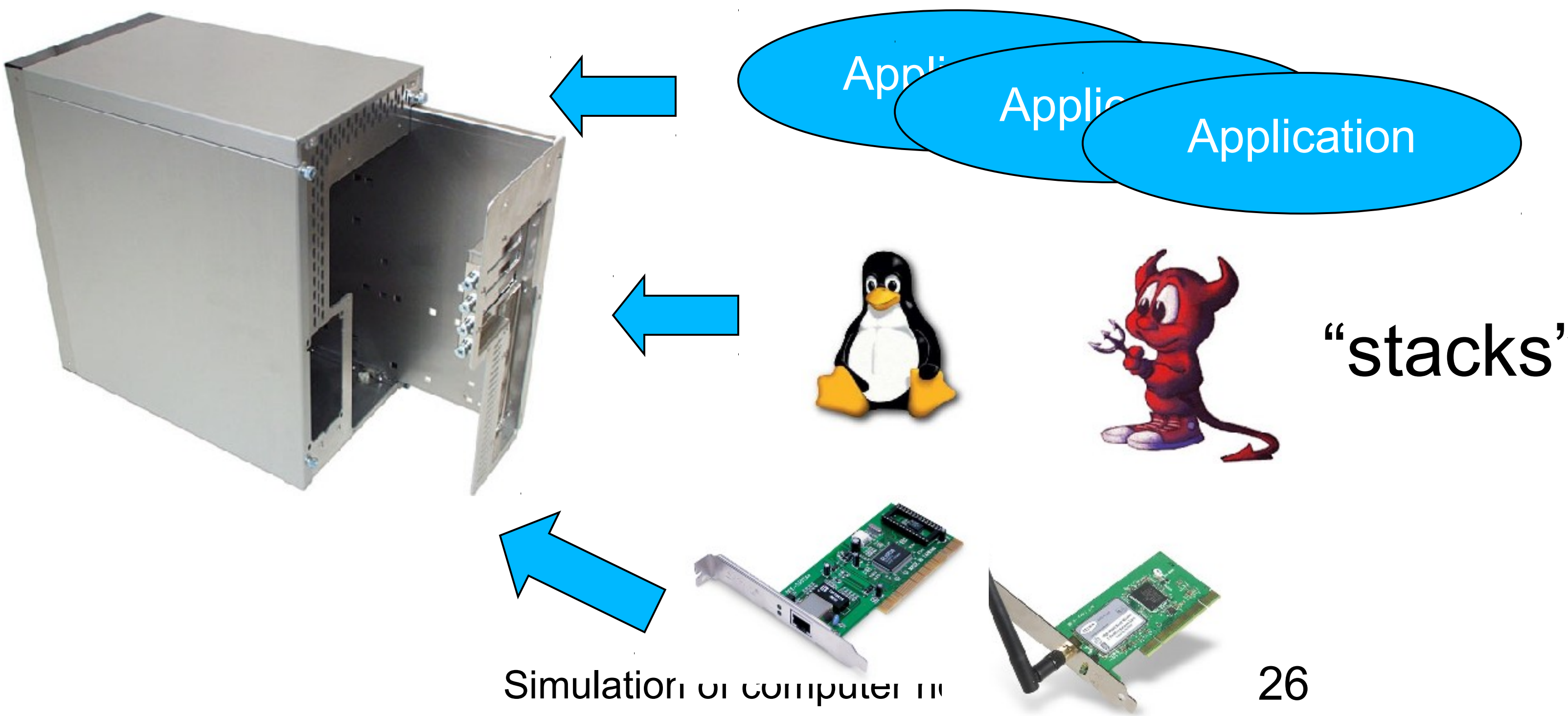
Fundamentals

Key objects in the simulator are Nodes, Packets, and Channels

Nodes may contain Applications, “stacks”, and NetDevices

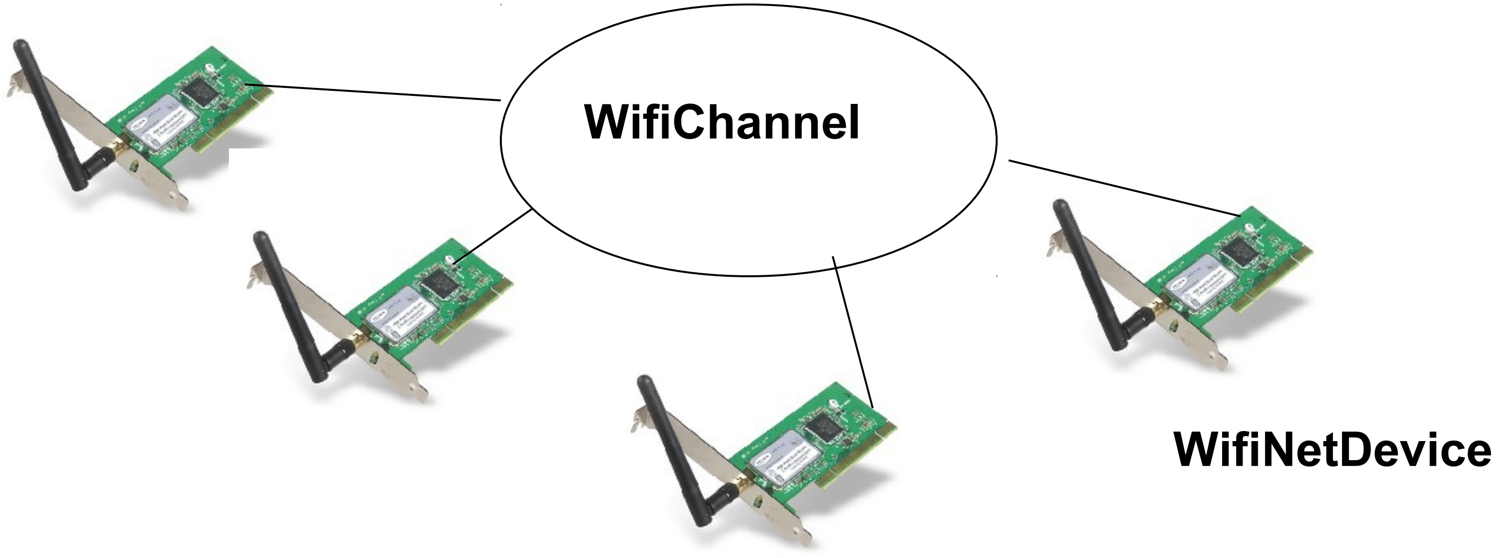
Node basics

A Node is a husk of a computer to which applications, stacks, and NICs are added



NetDevices and Channels

NetDevices are strongly bound to Channels of a matching type



Nodes are architected for multiple interfaces

Node basics

Two key abstractions are maintained:

1) applications use an (asynchrounous for the moment) sockets API

   – Based on the BSD Socket API

2) the boundary between IP and layer 2 mimics the boundary at the device-independent sublayer in Linux

   i.e., Linux Packet Sockets

Ns-3 packets

- each network packet contains a byte buffer, a list of tags, and metadata
  - **buffer:** bit-by-bit (serialized) representation of headers and trailers
  - **tags:** set of arbitrary, user-provided data structures (e.g., per-packet cross-layer messages, or flow identifiers)
  - **metadata:** describes types of headers and and trailers that have been serialized
    - optional-- disabled by default

Ns-3 packets (2)

- Each type of header is represented by a subclass of ns3::Header
- to add a new header, subclass from Header, and write your Serialize() and Deserialize() methods
  - how bits get written to/from the Buffer
- Similar for Packet Tags

Simulation basics

- Simulation time moves discretely from event to event
- A simulation scheduler orders the event execution
- Simulation::Run() gets it all started
- Simulation stops at specific time or when events end (Simulation::Stop())

Ns-3 object metadata system

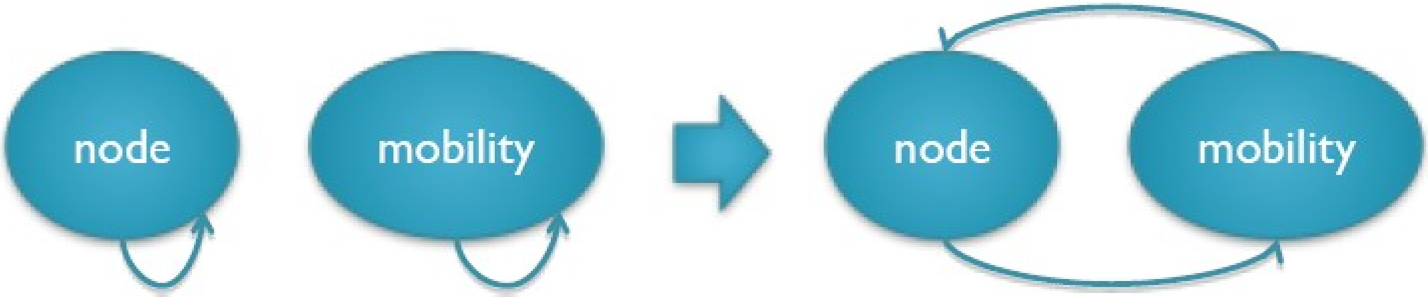
- ns-3 is, at heart, a C++ object system
- ns-3 objects that inherit from base class ns3::Object get several additional features
  - dynamic run-time object aggregation
  - an attribute system
  - smart-pointer memory management

Object aggregation

- You can aggregate objects to one another at run-time
  - Avoids the need to modify a base class to provide pointers to all possible connected objects
- Object aggregation is planned to be the main way to create new Node types (rather than subclassing Node)

Object aggregation example

- How aggregation works




```
node->AggregateObject (mobility);
```

- How to access on aggregated object

```
Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
```

Attributes

- An Attribute represents a value in our system
- An Attribute can be connected to an underlying variable or function
  - e.g. TcpSocket::m_cwnd;
  - or a trace source

Attributes (2)

- What would users like to do?
  - Set a default initial value for a variable
  - Set or get the current value of a variable
  - Know what are all the attributes that affect the simulation at run time
  - Initialize the value of a variable when a constructor is called
- The attribute system is a unified way of handling these functions

How to handle attributes?

- The traditional C++ way:
  - export attributes as part of a class's public API
  - walk pointer chains (and iterators, when needed) to find what you need
  - use static variables for defaults
- The attribute system provides a more convenient API to the user to do these things

The traditional C++ way

```
class Foo {
public:
  void SetVar1 (uint32_t value);
  uint32_t GetVar1 (void);
  static void SetInitialVar1(uint32_t value);
  void SetVar2 (uint32_t value);
  uint32_t GetVar2 (void);
  static void SetInitialVar2(uint32_t value);
  ...
private:
  uint32_t m_var1;  // document var1
  uint32_t m_var2;  // document var2
  static uint32_t m_initial_var1;
  static uint32_t m_initial_var2;
}

Foo::Foo() : m_var1(Foo::m_initial_var1), m_var2(Foo::m_initial_var2){
}
```

To modify an instance of Foo, get the pointer somehow, and use the public accessor functions
To modify the default values, modify the statics
Default values may be available in a separate framework (e.g. ns-2 Bind())

Navigating the attributes

- Attributes are exported into a string-based namespace, with filesystem-like paths
  - namespace supports regular expressions
- Examples:
  - Nodes with identifiers 1, 3, 4, 5, 8, 9, 10, 11:
    - "/NodeList/[3-5]|[8-11]"
  - UdpL4Protocol object instance aggregated to matching nodes:
    - "/NodeList/[3-5]|[8-11]/$UdpL4Protocol"
  - UdpL4Protocol object instances of all nodes:
    - "/NodeList/*/$UdpL4Protocol"
  - EndPoints which match the SrcPort=1025 specification:
    - "/EndPoints/*:SrcPort=1025"

Creating references in the namespace

- Names::Add ("server", serverNode);
- Names::Add ("server/eth0", serverDevice);
- ...
- Config::Set ("/Names/server/eth0/TxQueue/MaxPackets", UintegerValue (25));

Attributes: details

- Attributes also can be referred to without the paths
  - e.g. `"YansWifiPhy::TxGain"`
- A Config class allows users to manipulate the attributes
  - e.g.: Set a default initial value for a variable
  - (Note: this replaces DefaultValue::Bind())

```
Config::SetDefault ("YansWifiPhy::TxGain", Double (1.0));
```

Attribute ↑ ("YansWifiPhy::TxGain")

Value ↑ (Double (1.0))

Attributes: details

- Set or get the current value of a variable
  - Here, one needs the path in the namespace to the right instance of the object

```
Config::SetAttribute("/NodeList/5/DeviceList/
3/YansWifiPhy/TxGain", Double(1.0));
```

```
Double d =
Config::GetAttribute("/NodeList/5/NetDevice/3
/YansWifiPhy/TxGain");
```

- Users can get Ptrs to instances also, and Ptrs to trace sources, in the same way

How is all this implemented (overview)

```
class Foo: public Object
{
public:
  static TypeId GetTypeId (void);
private:
  uint32_t m_var1;  // document var1
  uint32_t m_var2;  // document var2
}

Foo::Foo() {
}

TypeId Foo::GetTypeId (void)
{
  static TypeId tid = TypeId("Foo")
  .AddConstructor<Foo> ();
  .AddAttribute ("m_var1", "document var1",
    UInteger(3),
    MakeUIntegerAccessor (&Foo::m_var1),
    MakeUIntegerChecker<uint32_t> ())
  return tid;
}
```

A real TypeId example

```
TypeId
RandomWalk2dMobilityModel::GetTypeId (void)
{
  static TypeId tid = TypeId ("RandomWalkMobilityModel")
    .SetParent<MobilityModel> ()
    .SetGroupName ("Mobility")
    .AddConstructor<RandomWalk2dMobilityModel> ()
    .AddAttribute ("bounds",
                   "Bounds of the area to cruise.",
                   Rectangle (0.0, 0.0, 100.0, 100.0),
                   MakeRectangleAccessor (&RandomWalk2dMobilityModel::m_bounds),
                   MakeRectangleChecker ())
    .AddAttribute ("time",
                   "Change current direction and speed after moving for this delay.",
                   Seconds (1.0),
                   MakeTimeAccessor (&RandomWalk2dMobilityModel::m_modeTime),
                   MakeTimeChecker ())
    .AddAttribute ("distance",
                   "Change current direction and speed after moving for this distance.",
                   Seconds (1.0),
                   MakeTimeAccessor (&RandomWalk2dMobilityModel::m_modeTime),
                   MakeTimeChecker ())
```

Object system: smart pointers

- ns-3 uses reference-counting **smart pointers** at its APIs to limit memory leaks
  - Or “pass by value” or “pass by reference to const” where appropriate
- A “smart pointer” behaves like a normal pointer (syntax) but does not lose memory when reference count goes to zero
- Use them like built-in pointers:

```
Ptr<MyClass> p = CreateObject<MyClass> ();
p->method ();
```

CreateObject<> ();

- CreateObject<> is a wrapper for operator new.
- Why not just, e.g., Node * node = new Node()?
  - You have to manage the memory allocation and deallocation
- ns3::Object are created on the heap using CreateObject<> (), which returns a smart pointer; e.g.

```
Ptr<Node> rxNode = CreateObject<Node> ();
```

Create<> ();

- What is Create<> ()?
- Create<> provides some smart pointer help for objects that use ns3::Ptr<> but that do not inherit from Object.
- Principally, class ns3::Packet

```
Ptr<Packet> p = Create<Packet>
(data,size);
```

Non-default constructors

- The attribute system allows you to also pass them through the CreateObject<> constructor.
- This provides a *generic* non-default constructor for users (any combination of parameters), e.g.:

```
Ptr<YansWifiPhy> phy = CreateObject<YansWifiPhy>
("TxGain", Double (1.0));
```

How to parse command line arguments

- **To configure the from the command line ns-3 provides the *Command* facility**

  - ```
    int   main (int argc, char *argv[])
     {
       ...
       CommandLine cmd;
       cmd.Parse (argc, argv);
       ...
    }
    ```

- **The snippet of code above enables the setting of all the attributes in the ns-3 attributes namespace**

  - ./waf --run "scratch/first --ns3::PointToPointNetDevice::DataRate=5Mbps"
    ```

How to parse command line arguments (2)

- Custom global properties can be set from the command line as well
  - int main(int argc, char * argv[]){

```
...
uint32_t nPackets = 1;
CommandLine cmd;
cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
cmd.Parse (argc, argv);
...
}
```

- How to set the property on the command line
  - ./waf --run "scratch/first --nPackets=2"

Helpers objects

- Provide simple 'syntactical sugar' to make simulation scripts look nicer and easier to read for network researchers
- Helpers make it easier to repeat the same operations on a set of resources (e.g. nodes, interfaces)
- The settings are applied to the helper and used to perform the operations on the resources
- Each function applies a single operation on a "set of same objects”

Helper Objects (2)

- InternetStackHelper
- MobilityHelper
- OlsrHelper
- ... Each model provides a helper class
- What does this apply to?
  - NodeContainer: vector of Ptr<Node>
  - NetDeviceContainer: vector of Ptr<NetDevice>

Tutorial outline

- Introduction to ns-3
- Architecture of ns-3
- Ns-3 logging and tracing
- Ns-3 examples

Ns-3 logging

- ns-3 has a built-in logging facility to stderr
- Features:
  - Multiple log levels like syslog
  - can be driven from shell environment variables
  - Function and call argument tracing
- Intended for debugging, but can be abused to provide tracing
  - It is not guaranteed that format is unchanging

Ns-3 logging example

- **Logging facilities**
  - NS_LOG_ERROR -- Log error messages;
  - NS_LOG_WARN -- Log warning messages;
  - NS_LOG_DEBUG -- Log relatively rare, ad-hoc debugging messages;
  - NS_LOG_INFO -- Log informational messages about program progress;
  - NS_LOG_FUNCTION -- Log a message describing each function called;
  - NS_LOG_LOGIC -- Log messages describing logical flow within a function;
  - NS_LOG_ALL -- Log everything.
  - NS_LOG_UNCOND() – Log unconditionally (irrespective to the debug settings);
- **How to activate logging messages:**
  - From the shell
    - export NS_LOG=”WifiNetDevice:YansWifiChannel” (Unix)
    - export NS_LOG=”*” (Unix)
    - export NS_LOG=UdpEchoClientApplication=level_all
  - In the code
    - LogComponentEnable("Class", LOGGING_LEVEL);
    - LogComponentEnable ("UdpEchoExample", LOG_LEVEL_INFO);

Tracing model

- Tracing is a structured form of simulation output
  - tracing format should be relatively static across simulator releases
- Example (from ns-2):

```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600
r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602
+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
```

- Needs vary widely

Crude tracing

```
#include <iostream>
...
int main ()
{
   ...
   std::cout << "The value of x is " << x <<
  std::endl;
   ...
}
```

Slightly less crude

```
#include <iostream>
...
int main ()
{
  ...
  NS_LOG_UNCOND ("The value of x is " << x);
  ...
}
```

Simple ns-3 tracing

- these are wrapper functions/classes
- see examples/mixed-wireless.cc

```
#include "ns3/ascii-trace.h"



 AsciiTrace asciitrace ("mixed-wireless.tr");
 asciitrace.TraceAllQueues ();
 asciitrace.TraceAllNetDeviceRx ();
```

Simple ns-3 tracing (pcap version)

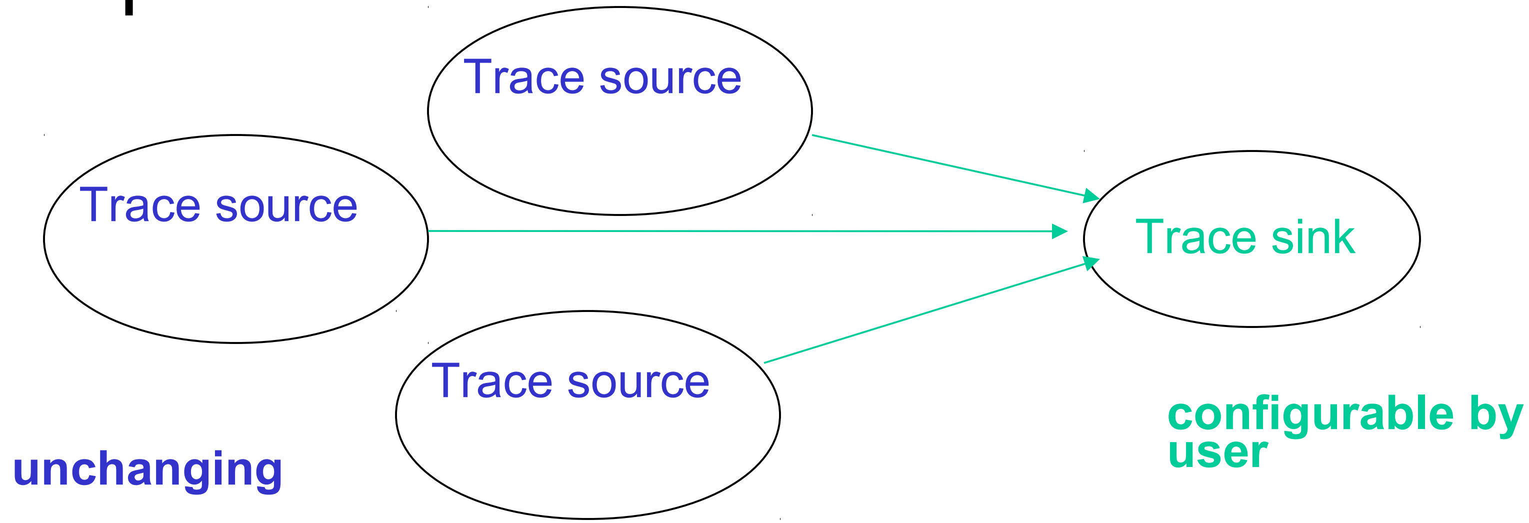
- these are wrapper functions/classes
- see examples/mixed-wireless.cc

```
#include "ns3/pcap-trace.h"


  PcapTrace pcaptrace ("mixed-wireless.pcap");
  pcaptrace.TraceAllIp ();
```

Ns-3 tracing model

- Fundamental #1: decouple trace sources from trace sinks
- Fundamental #2: prefer standard trace outputs for built-in traces



Tracing overview

- Simulator provides a set of pre-configured trace sources
  - Users may edit the core to add their own
- Users provide trace sinks and attach to the trace source
  - Simulator core provides a few examples for common cases
- Multiple trace sources can connect to a trace sink

Multiple levels of tracing

- Highest-level: Use built-in trace sources and sinks and hook a trace file to them
- Mid-level: Customize trace source/sink behavior using the tracing namespace
- Low-level: Add trace sources to the tracing namespace
  - Or expose trace source explicitly

High-level of tracing

- High-level: Use built-in trace sources and sinks and hook a trace file to them

```
// Also configure some tcpdump traces; each interface will be traced
// The output files will be named
// simple-point-to-point.pcap-<nodeId>-<interfaceId>
// and can be read by the "tcpdump -r" command (use "-tt" option to
// display timestamps correctly)
PcapTrace pcaptrace ("simple-point-to-point.pcap");
pcaptrace.TraceAllIp ();
// Ascii format
std::ofstream ascii;
ascii.open ("myfirst.tr");
PointToPointHelper::EnableAsciiAll (ascii);
```

High level of tracing (2)

```
//Su Helper di interfaccia
...
void EnablePcap (std::string prefix, Ptr<NetDevice> nd, bool promiscuous =
false, bool explicitFilename = false);
void EnablePcap (std::string prefix, std::string ndName, bool promiscuous =
false, bool explicitFilename = false);
void EnablePcap (std::string prefix, NetDeviceContainer d, bool promiscuous =
false);
void EnablePcap (std::string prefix, NodeContainer n, bool promiscuous = false);
void EnablePcap (std::string prefix, uint32_t nodeid, uint32_t deviceid, bool
promiscuous = false);
void EnablePcapAll (std::string prefix, bool promiscuous = false);
```

Reading pcap files

- pcap files can be read by means of
  - Wireshark
  - Tcpdump

```
// Example of tcpdump usage
tcpdump -nn -tt -r simple-point-to-point.pcap
reading from file myfirst-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.514648 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

Mid-level of tracing

- Mid-level: Customize trace source/sink behavior using the tracing namespace

```
static void CwndTracer (uint32_t oldval, uint32_t newval)
{
   NS_LOG_INFO ("Moving cwnd from " << oldval << " to " <<
newval);
}
 Config::ConnectWithoutContext
("/NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow"
, MakeCallback (&CwndTracer));
```

Low-level of tracing

- Define your own trace sources
- For specific advanced needs
- Need to modify the core of ns-3
  - Can be defined in custom-elements
  - Easy to apply to any existing object attribute

An additional trace method: statistics

- Avoid large trace files
- Collect statistics of the simulation
- Reuse tracing framework
- One similar approach: ns-2-measure project
  - http://info.iet.unipi.it/~cng/ns2measure/
  - Static “Stat” object that collects samples of variables based on explicit function calls inserted into the code
  - Graphical front end, and framework for replicating simulation runs
- FlowMon is currently available
  - http://telecom.inescn.pt/~gjc/flowmon-presentation.pdf

Ns-3 tutorial

- Introduction to ns-3
- Architecture of ns-3
- Logging and tracing
- <u>Ns-3 examples</u>

examples/ directory

- examples/ contains other scripts with similar themes

```
$ ls
csma-broadcast.cc                  simple-point-to-point.cc
csma-multicast.cc                  tcp-large-transfer-errors.cc
csma-one-subnet.cc                 tcp-large-transfer.cc
csma-packet-socket.cc              tcp-nonlistening-server.cc
mixed-global-routing.cc            tcp-small-transfer-oneloss.cc
simple-alternate-routing.cc        tcp-small-transfer.cc
simple-error-model.cc              udp-echo.cc
simple-global-routing.cc           waf
simple-point-to-point-olsr.cc      wscript
```

First example (udp-echo)

- We simulate a simple network with 2 nodes connected through an ethernet link
- One node send a (udp) packet to a second node
- The second node receives the packet and send a copy of the packet back

udp-echo

- First step: create nodes
  - *NodeContainer n*
  - *n.Create (4);*
- Install the fundamental TCP/IP entities on the nodes
  - *InternetStackHelper internet;*
  - *internet.Install (n);*

Udp-echo (2)

- Create the local network (i.e. ethernet network)
  - *CsmaHelper csma;*
  - *csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));*
  - *csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));*
  - *csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));*
  - *NetDeviceContainer d = csma.Install (n);*

Udp-echo (3)

- Assign the addresses to the nodes' interfaces
  - *Ipv4AddressHelper ipv4;*
  - *ipv4.SetBase ("10.1.1.0", "255.255.255.0");*
  - *Ipv4InterfaceContainer i = ipv4.Assign (d);*
  - *serverAddress = Address(i.GetAddress (1));*

udp-echo

- Install the server application on node 1
  - *uint16_t port = 9; // well-known echo port number*
  - *UdpEchoServerHelper server (port);*
  - *ApplicationContainer apps = server.Install (n.Get (1));*
  - *apps.Start (Seconds (1.0));*
  - *apps.Stop (Seconds (10.0));*

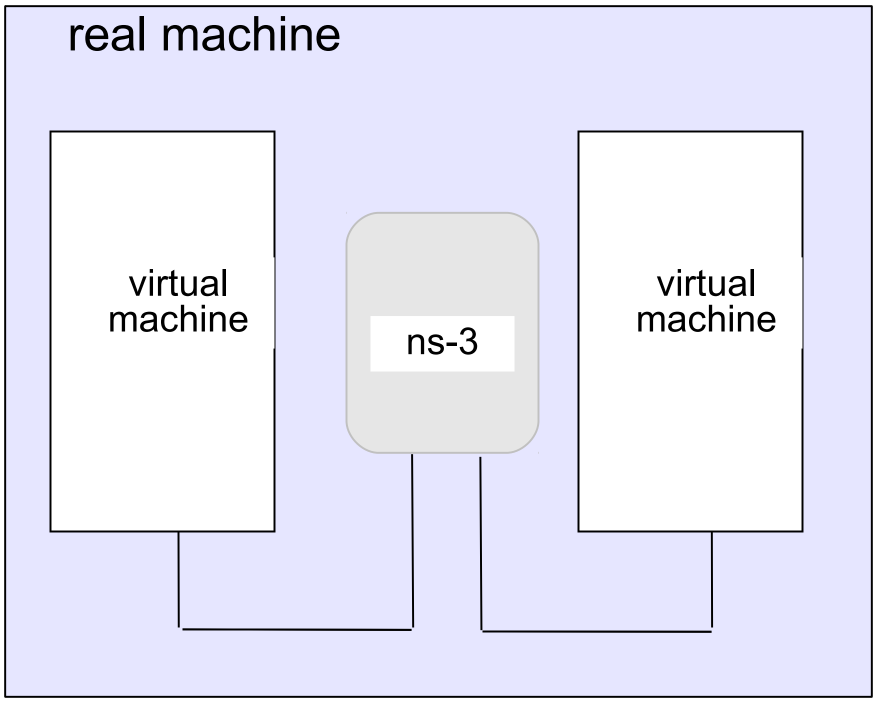
udp-echo

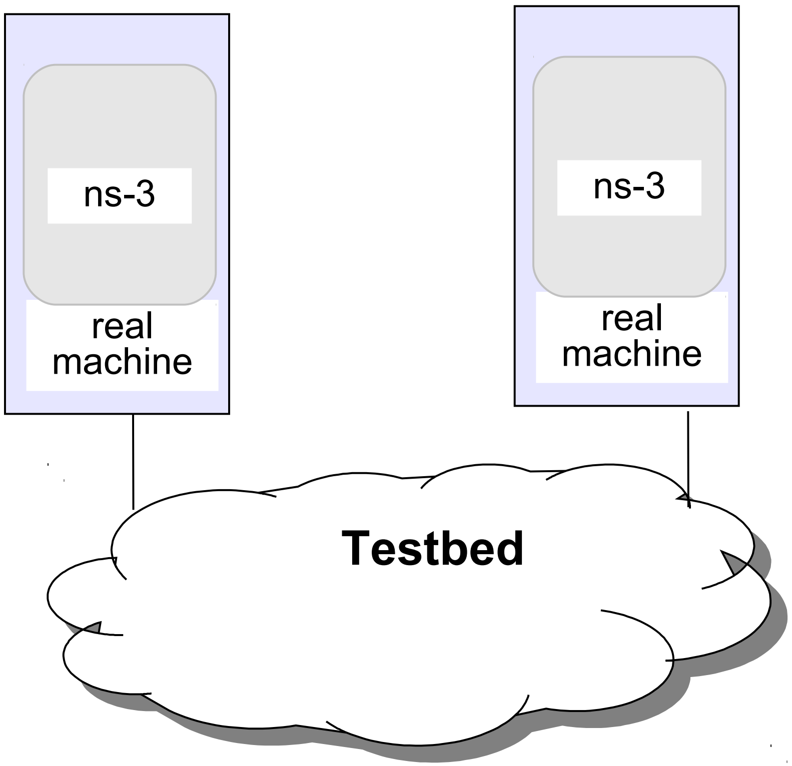
- Install the client application on node 0
  - *uint32_t packetSize = 1024;*
  - *uint32_t maxPacketCount = 1;*
  - *Time interPacketInterval = Seconds (1.);*
  - *UdpEchoClientHelper client (serverAddress, port);*
  - *client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));*
  - *client.SetAttribute ("Interval", TimeValue (interPacketInterval));*
  - *client.SetAttribute ("PacketSize", UintegerValue (packetSize));*
  - *apps = client.Install (n.Get (0));*
  - *apps.Start (Seconds (2.0));*
  - *apps.Stop (Seconds (10.0));*

udp-echo

- Enable netanim logs
  - [#include "ns3/netanim-module.h"]
  - AnimationInterface anim ("udp-echo.xml");
- Enable pcap logs
  - AsciiTraceHelper ascii;
  - csma.EnableAsciiAll (ascii.CreateFileStream ("udp-echo.tr"));
  - csma.EnablePcapAll ("udp-echo", false);
- Start the simulation
  - Simulator::Run ()

One more thing: ns-3 used for emulation





1) ns-3 interconnects virtual machines

2) testbeds interconnect ns-3 stacks

Summary

- ns-3 is an emerging simulator to replace ns-2
- Consider ns-3 if you are interested in:
  - Modular architecture
    - Easily extendable
  - More faithful representations of real computers and the Internet
  - Integration with testbeds
  - A powerful low-level API
  - Python scripting