ns-3 Training

Session 8: Monday 3:30pm

ns-3 Annual Meeting
May 2014

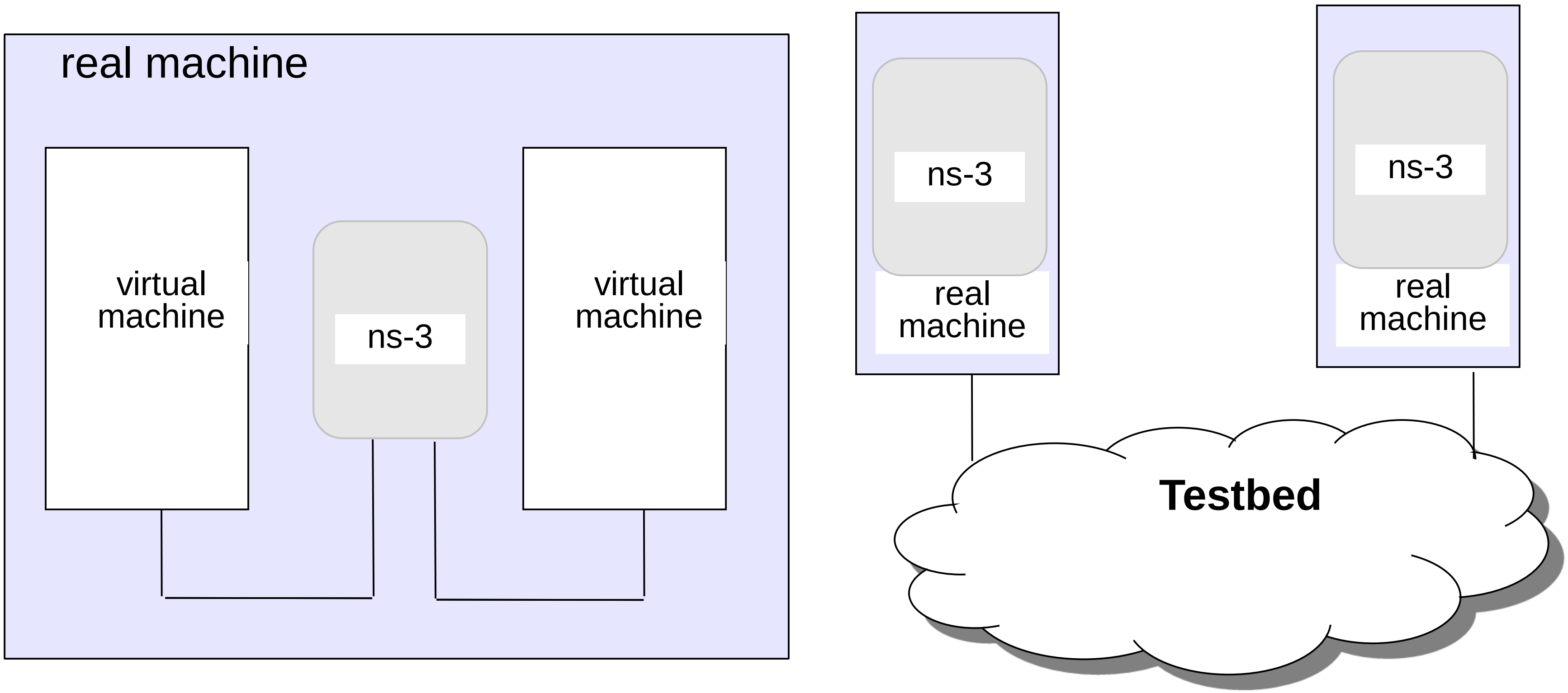
Outline

- Emulation modes
  - Tap Bridge
  - FdNetDevice
- Direct Code Execution (DCE)
  - Applications
  - Linux Kernel
  - DCE Cradle

Emulation support

- Support moving between simulation and testbeds or live systems
- A real-time scheduler, and support for two modes of emulation
- Linux is only operating system supported
- Must run simulator in real time
  - `GlobalValue::Bind (“SimulatorImplementationType”, StringValue (“ns3::RealTimeSimulatorImpl”));`
- Must enable checksum calculations across models
  - `GlobalValue::Bind (“ChecksumEnabled”, BooleanValue (true));`
- Must run as root, or with the --enable-sudo

ns-3 emulation modes



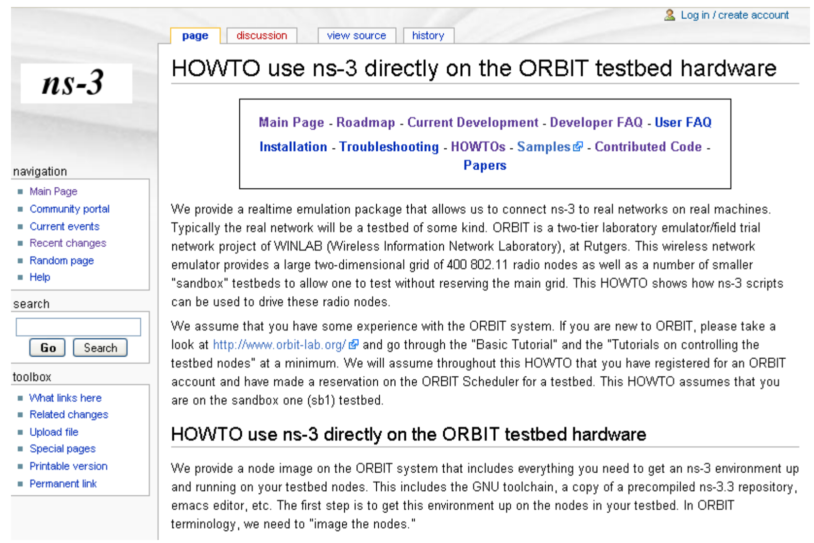
1) ns-3 interconnects real or virtual machines

2) testbeds interconnect ns-3 stacks

Various hybrids of the above are possible

Example use case: testbeds

- Support for use of Rutgers WINLAB ORBIT radio grid



Log in / create account

page | discussion | view source | history

ns-3

navigation
- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help

search

Go | Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

HOWTO use ns-3 directly on the ORBIT testbed hardware

Main Page - Roadmap - Current Development - Developer FAQ - User FAQ
Installation - Troubleshooting - HOWTOs - Samples - Contributed Code - Papers

We provide a realtime emulation package that allows us to connect ns-3 to real networks on real machines. Typically the real network will be a testbed of some kind. ORBIT is a two-tier laboratory emulator/field trial network project of WINLAB (Wireless Information Network Laboratory), at Rutgers. This wireless network emulator provides a large two-dimensional grid of 400 802.11 radio nodes as well as a number of smaller "sandbox" testbeds to allow one to test without reserving the main grid. This HOWTO shows how ns-3 scripts can be used to drive these radio nodes.

We assume that you have some experience with the ORBIT system. If you are new to ORBIT, please take a look at http://www.orbit-lab.org/ and go through the "Basic Tutorial" and the "Tutorials on controlling the testbed nodes" at a minimum. We will assume throughout this HOWTO that you have registered for an ORBIT account and have made a reservation on the ORBIT Scheduler for a testbed. This HOWTO assumes that you are on the sandbox one (sb1) testbed.

HOWTO use ns-3 directly on the ORBIT testbed hardware

We provide a node image on the ORBIT system that includes everything you need to get an ns-3 environment up and running on your testbed nodes. This includes the GNU toolchain, a copy of a precompiled ns-3.3 repository, emacs editor, etc. The first step is to get this environment up on the nodes in your testbed. In ORBIT terminology, we need to "image the nodes."

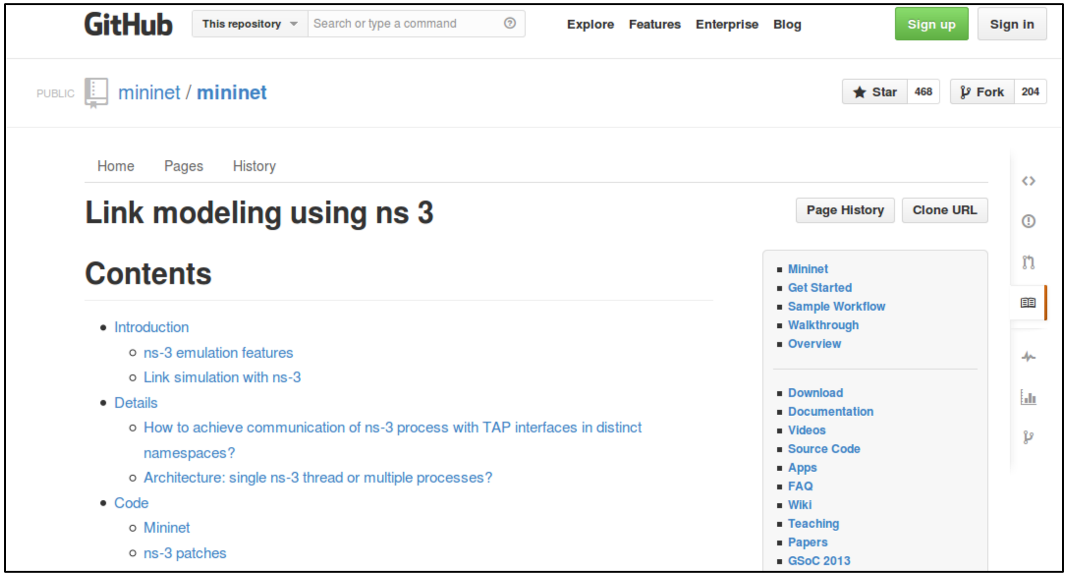
Example use case: PlanetLab

- The PlanetLabFdNetDeviceHelper creates TAP devices on PlanetLab nodes using specific PlanetLab mechanisms (i.e. the vsys system), and associates the TAP device to a FdNetDevice in ns-3.

```
+-------------------------------------+
|         PlanetLab  host             |
+-------------------------------------+
|    ns-3 simulation   |              |
+----------------------+              |
|       ns-3 Node      |              |
|  +----------------+  |              |
|  |    ns-3 TCP    |  |              |
|  +----------------+  |              |
|  |    ns-3 IP     |  |              |
|  +----------------+  |              |
|  |   FdNetDevice  |  |              |
|--+----------------+--+    +------+  |
|       | TAP  |            | eth0 |  |
|       +------+            +------+  |
|     192.168.0.1               |     |
+-------------------------------|-----+
                                |
                                |
                                ------------ (Internet) -----
```

Example use case: mininet

- Mininet is popular in the Software-Defined Networking (SDN) community
- Mininet uses "TapBridge" integration
- https://github.com/mininet/mininet/wiki/Link-modeling-using-ns-3



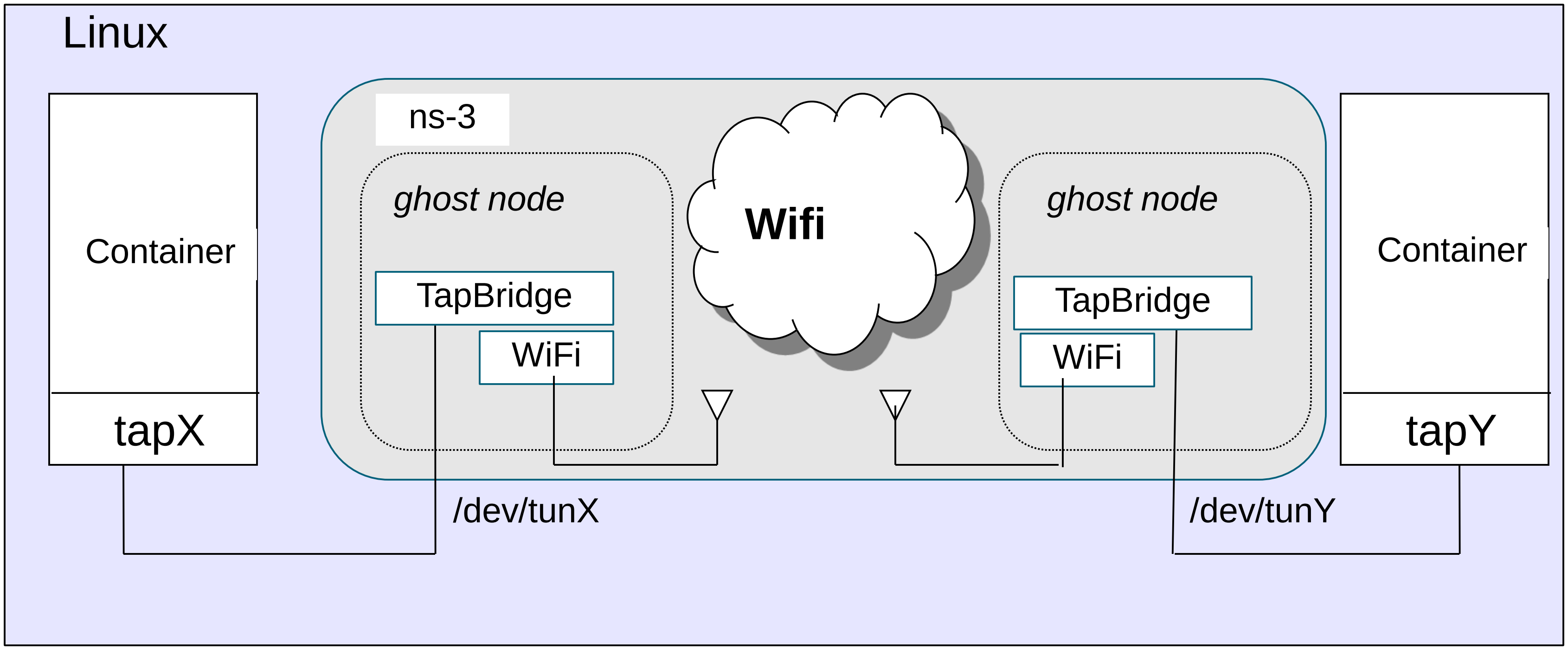
Emulation Devices

NETWORK SIMULATOR

Device models

- File Descriptor Net Device (FdNetDevice)
  - read and write traffic using a file descriptor provided by the user
  - this file descriptor can be associated to a TAP device, to a raw socket, to a user space process generating/consuming traffic, etc.
- Tap Bridge
  - Integrate Tun/Tap devices with ns-3 devices
- EmuNetDevice
  - Deprecated (ns-3.17) in favor of FdNetDevice

“TapBridge": netns and ns-3 integration



Tap device pushed into namespaces; no bridging needed

TapBridge modes

- ConfigureLocal (default mode)
  - ns-3 configures the tap device
  - useful for host to ns-3 interaction
- UseLocal
  - user has responsibility for device creation
  - ns-3 informed of device using “DeviceName” attribute
- UseBridge
  - TapDevice connected to existing Linux bridge

ConfigureLocal

```
+---------+
|  Linux  |
|  host   |                       +-----------+
| ------  |                       |   ghost   |
|  apps   |                       |   node    |
| ------  |                       | --------- |
|  stack  |                       |    IP     |     +-----------+
| ------  |                       |   stack   |     |   node    |
|  TAP    |                       |===========|     | --------- |
| device  | <----- IPC ------>    |    tap    |     |    IP     |
+---------+                       |  bridge   |     |   stack   |
                                  | --------- |     | --------- |
                                  |   ns-3    |     |   ns-3    |
                                  |    net    |     |    net    |
                                  |  device   |     |  device   |
                                  +-----------+     +-----------+
                                       ||                ||
                                  +-----------------------------+
                                  |        ns-3 channel         |
                                  +-----------------------------+
```

ns-3 ensures that Mac addresses are consistent

UseLocal

```
+---------+
|  Linux  |
|  host   |                              +----------+
| ------  |                              |   ghost  |
|  apps   |                              |   node   |
| ------  |                              | -------- |
|  stack  |                              |    IP    |     +----------+
| ------  |                              |   stack  |     |   node   |
|  TAP    |                              |==========|     | -------- |
| device  | <----- IPC ------>           |   tap    |     |    IP    |
| MAC X   |                              |  bridge  |     |   stack  |
+---------+                              | -------- |     | -------- |
                                         |   ns-3   |     |   ns-3   |
                                         |   net    |     |   net    |
                                         |  device  |     |  device  |
                                         |  MAC Y   |     |  MAC Z   |
                                         +----------+     +----------+
                                              ||               ||
                                         +---------------------------+
                                         |        ns-3 channel       |
                                         +---------------------------+
```

Mac X spoofed to Mac Y

UseBridge

```
+----------+
|  Linux   |
| -------  |
|  apps    |
| -------  |                                     +----------+
|  stack   |                                     |  ghost   |
| -------  | +--------+                          |  node    |
| Virtual  | |  TAP   |                          | -------- |
| Device   | | Device | <---- IPC ----->         |   IP     |    +----------+
+----------+ +--------+                          |  stack   |    |   node   |
    ||           ||                              |==========|    | -------- |
+---------------------+                          |   tap    |    |    IP    |
| OS (brctl) Bridge   |                          |  bridge  |    |   stack  |
+---------------------+                          | -------- |    | -------- |
                                                 |  ns-3    |    |   ns-3   |
                                                 |  net     |    |   net    |
                                                 |  device  |    |  device  |
                                                 +----------+    +----------+
                                                     ||              ||
                                                 +---------------------------+
                                                 |        ns-3 channel       |
                                                 +---------------------------+
```

ns-3 devices must support SendFrom()
(i.e. bridging)

TapCsma example

- Demo the TapCsma example

FdNetDevice

- Unified handling of reading/writing from file descriptor
- Three supported helper configurations:
  - EmuFdNetDeviceHelper (to associate the ns-3 device with a physical device in the host machine)
  - TapFdNetDeviceHelper (to associate the ns-3 device with the file descriptor from a tap device in the host machine) (not the same as TapBridge)
  - PlanetLabFdNetDeviceHelper (to automate the creation of tap devices in PlanetLab nodes, enabling ns-3 simulations that can send and receive traffic though the Internet using PlanetLab resource.

EmuFdNetDeviceHelper

- Device performs MAC spoofing to separate emulation from host traffic

```
+----------------------+     +----------------------+
|         host 1       |     |        host 2        |
+----------------------+     +----------------------+
|    ns-3 simulation   |     |                      |
+----------------------+     |         Linux        |
|       ns-3 Node      |     |     Network Stack    |
|  +----------------+  |     |   +----------------+ |
|  |    ns-3 TCP    |  |     |   |       TCP      | |
|  +----------------+  |     |   +----------------+ |
|  |    ns-3 IP     |  |     |   |       IP       | |
|  +----------------+  |     |   +----------------+ |
|  |   FdNetDevice  |  |     |   |                | |
|  |    10.1.1.1    |  |     |   |                | |
|  +----------------+  |     |   +    ETHERNET    + |
|  |   raw socket   |  |     |   |                | |
|--+----------------+--|     |   +----------------+ |
|       | eth0 |       |     |        | eth0 |      |
+-------+------+-------+     +--------+------+------+

        10.1.1.11                     10.1.1.12

            |                             |
            +-----------------------------+
```

PlanetLabFdNetDeviceHelper

- Special case of TapFdNetDeviceHelper where Tap devices configured according to PlanetLab conventions

```
+---------------------------------------+
|          PlanetLab  host              |
+---------------------------------------+
|    ns-3 simulation   |                |
+----------------------+                |
|       ns-3 Node      |                |
|  +----------------+  |                |
|  |    ns-3 TCP    |  |                |
|  +----------------+  |                |
|  |    ns-3 IP     |  |                |
|  +----------------+  |                |
|  |   FdNetDevice  |  |                |
|--+----------------+--+    +-------+   |
|       | TAP  |             | eth0 |   |
|       +------+             +-------+  |
|     192.168.0.1                |      |
+-------------------------------|-----+
                                |
                                |
                                ------------ (Internet) -----
```

ns-3 over host sockets

- Two publications about how to run ns-3 applications over real hosts and sockets
  - "Simulator-agnostic ns-3 Applications", Abraham and Riley, WNS3 2012
  - Gustavo Carneiro, Helder Fontes, Manuel Ricardo, "Fast prototyping of network protocols through ns-3 simulation model reuse", Simulation Modelling Practice and Theory (SIMPAT), vol. 19, pp. 2063–2075, 2011.

Generic Emulation Issues

- Ease of use
  - Configuration management and coherence
  - Information coordination (two sets of state)
    - e.g. IP/MAC address coordination
  - Output data exists in two domains
  - Debugging can be more challenging
- Error-free operation (avoidance of misuse)
  - Synchronization, information sharing, exception handling
    - Checkpoints for execution bring-up
    - Inoperative commands within an execution domain
    - Deal with run-time errors
  - Soft performance degradation (CPU) and time discontinuities

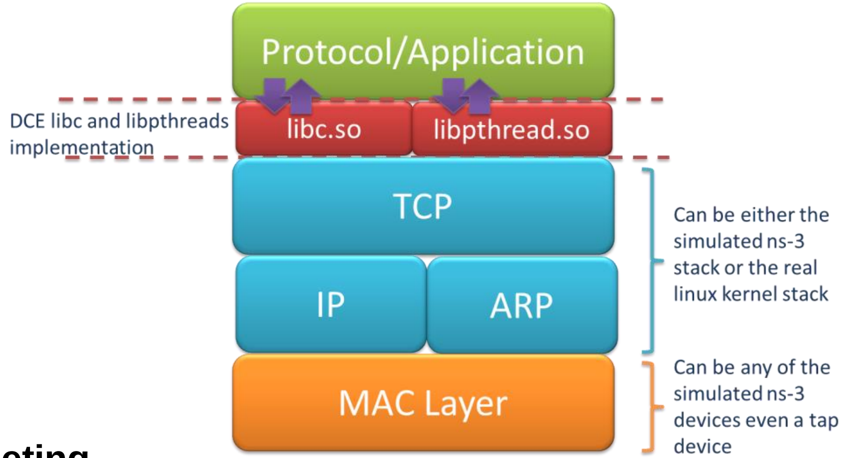
Direct Code Execution

Goals

- Lightweight virtualization of kernel and application processes, interconnected by simulated networks
- Benefits:
  - Implementation realism in controlled topologies or wireless environments
  - Model availability
  - Debugging a whole network within a single process
- Limitations:
  - Not as scalable as pure simulation
  - Tracing more limited
  - Configuration different

Direct Code Execution

- DCE/ns-3 framework requires the virtualization of a series of services
  - Multiple isolated instances of the same protocol on the same machine
- System calls are captured and treated by DCE
- Network stack protocols calls are captured and redirected
- To perform its work DCE re-implement the Linux program loader and parts of *libc* and *libpthread*



Direct Code Execution

- Developed by Mathieu Lacage and Frederic Urbani, INRIA, Hajime Tazaki (University of Tokyo)

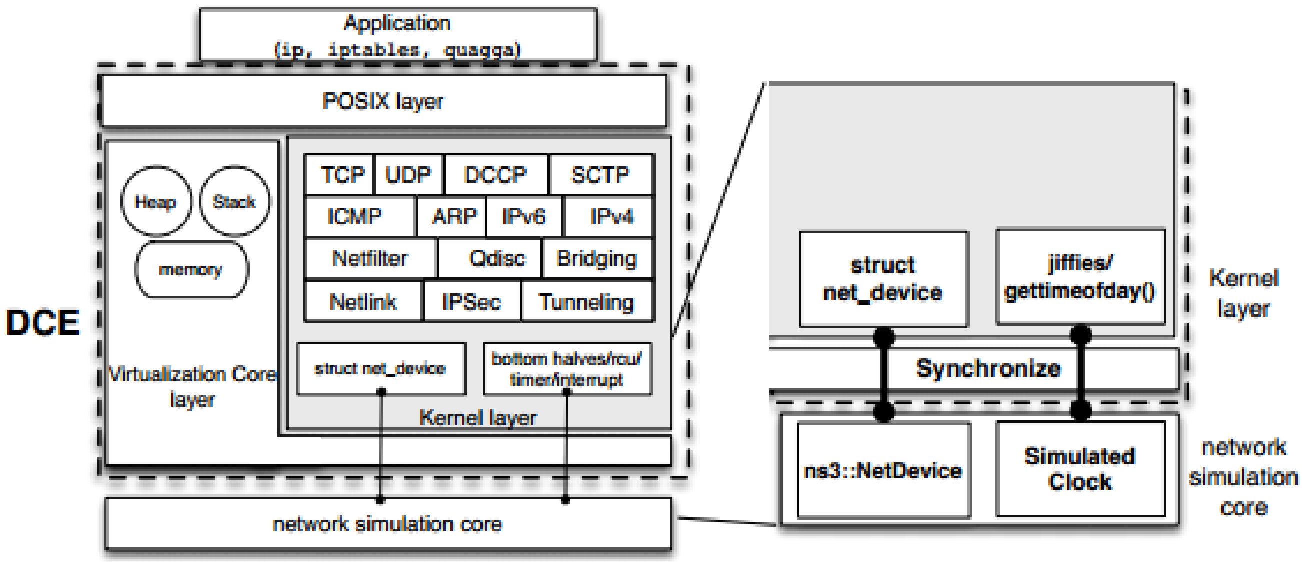


Figure 1: Architecture of Direct Code Execution. Kernel network devices and timers are synchronized with simulated NetDevice and clock.

Figure source: Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments (CONEXT 13)

DCE modes

- DCE modes in context of possible approaches

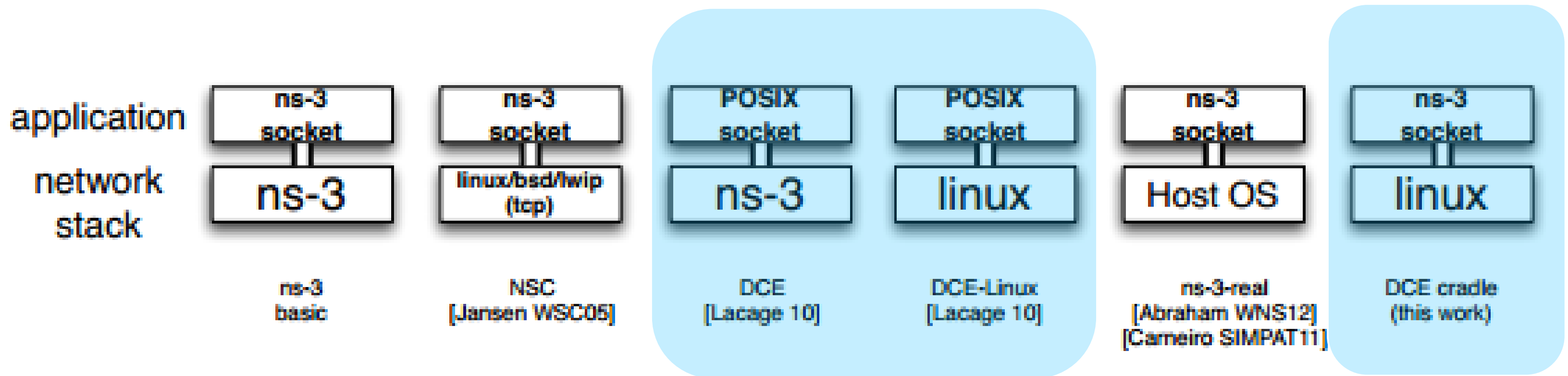


Figure 1: Current possible combinations of network stacks and applications.

Figure source: DCE Cradle: Simulate Network Protocols with Real Stacks for Better Realism, Tazaki et al, WNS3 2013.

Paper references

- Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments
  - Tazaki et al, CONEXT 2013
  - http://hal.archives-ouvertes.fr/docs/00/88/08/70/PDF/con013-hal.pdf
- DCE Cradle: Simulate Network Protocols with Real Stacks for Better Realism
  - Tazaki et al, WNS3 2013
  - http://hal.archives-ouvertes.fr/docs/00/78/15/91/PDF/wns3-2013.pdf

Hands on, what do we need

- **What do you need to start using the framework!**
  - ns-3 – The network simulator
    - http://www.nsnam.org/
  - DCE
    - http://www.nsnam.org/overview/projects/direct-code-execution/
  - Applications
    - iperf, wget, thttpd

  * All software must be re-compiled with –fpic and linked with –pie to generate the code with Position Independent Code (PIC) and permit context switch

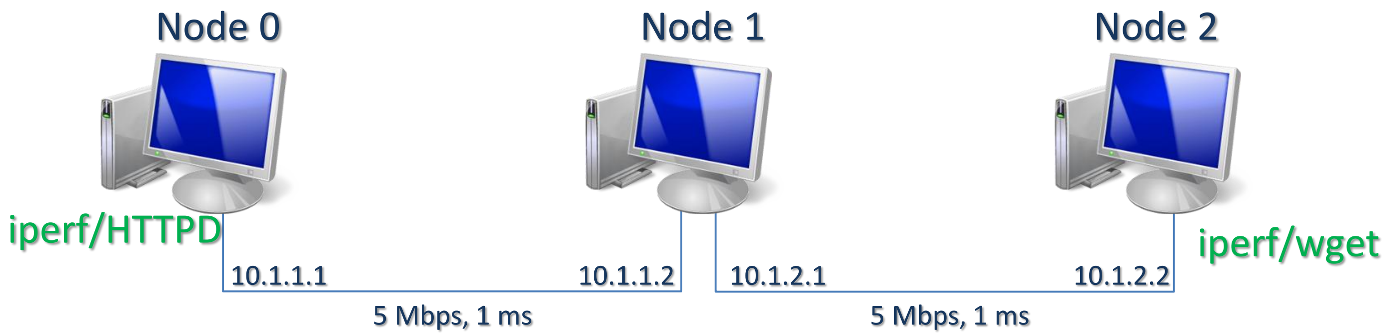
- **To make things easier**
  - Bake – Installation tool
    - http://planete.inria.fr/software/bake/index.html
  - Mercurial – source control management tool
    - http://mercurial.selenic.com/
  - Python – for running bake
    - www.python.org

The plan is

- **The plan is to present**
  - Installation
  - Examples of:
    - iperf with ns-3 stack
    - www server and wget with ns-3 stack
    - iperf with Linux stack

The shared scenario

- The shared scenario is a simple three nodes network



Step by step example
- Installing the required software

* Into a Linux machine

1) > mkdir dce_tutorial; cd dce_tutorial
2) > hg clone http://code.nsnam.org/bake bake
3) > export BAKE_HOME=`pwd`/bake
4) > export PATH=$PATH:$BAKE_HOME
5) > export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
6) > mkdir DCE; cd DCE
7) > bake.py configure -e dce-ns3
8) > bake.py install
9) >. bakeSetEnv.sh

Step by step example
- What we need to do!

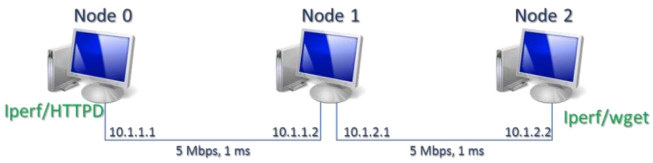
1. Create the nodes
2. Create stack
3. Create devices
4. Set addresses
5. Connect devices
6. Create DCE
7. Configuration the applications to run
8. Set start time for server and client
9. Set simulation time
10. Start simulation

Step by step example
- What we need to do!

1) Create the nodes
2) Create stack
3) Create devices
4) Set addresses
5) Connect devices
6) Create DCE
7) Configuration the applications to run
8) Set start time for server and client
9) Set simulation time
10) Start simulation

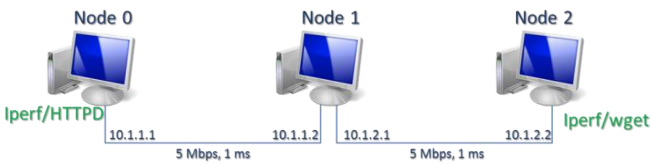
Standard ns-3 procedures
DCE specific

Step by step example
- iperf with ns-3 stack (I)




```
int main (int argc, char *argv[])
{
  // Node Container creation
  NodeContainer nodes;
  nodes.Create (3);

  // Linux stack creation
  InternetStackHelper stack;
  stack.Install (nodes);

  // For real time
  // GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3::RealtimeSimulatorImpl"));
  // GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));

  // Device and channel  creation
  PointToPointHelper p2p;
  p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps“));
  p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));
```

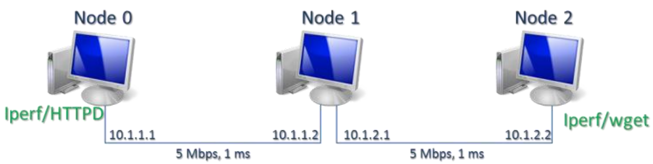
Step by step example
- iperf with ns-3 stack (II)




```
//  Node0-Node1 setup
 Ipv4AddressHelper address;
 address.SetBase ("10.1.1.0", "255.255.255.252"); //  Node0-Node1 addresses

  NetDeviceContainer devices;
 devices = p2p.Install (nodes.Get (0), nodes.Get (1)); // connecting nodes
 Ipv4InterfaceContainer interfaces = address.Assign (devices); //  assign addresses

 //  Node1-Node2 setup
 devices = p2p.Install (nodes.Get (1), nodes.Get (2)); // connecting nodes
 address.SetBase ("10.1.2.0", "255.255.255.252"); //  Node1-Node2 addresses
 interfaces = address.Assign (devices); //  assign addresses

 // setup ip routes
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

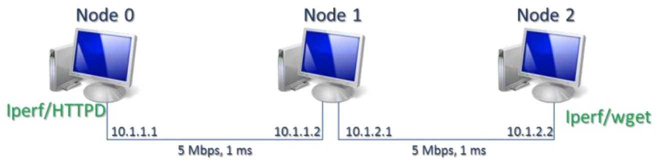
Step by step example
- iperf with ns-3 stack (III)

```
DceManagerHelper dceManager;
dceManager.Install (nodes);

DceApplicationHelper dce;
ApplicationContainer apps;
 dce.SetStackSize (1 << 20); // 1MB stack

dce.SetBinary ("iperf"); // Launch iperf client on node 0
dce.ResetArguments ();  // clean arguments
dce.ResetEnvironment (); // clean environnent
dce.AddArgument ("-c");  // client
dce.AddArgument ("10.1.2.2"); //target machine address
dce.AddArgument ("-i"); // interval
dce.AddArgument ("1");
dce.AddArgument ("--time"); // how long
dce.AddArgument ("10");
 apps = dce.Install (nodes.Get (0)); //install application
apps.Start (Seconds (0.7)); //start at 0.7 simulation time
apps.Stop (Seconds (20)); //stop at 20s        simulation time

dce.SetBinary ("iperf"); // Launch iperf server on node 2
dce.ResetArguments (); // clean arguments
dce.ResetEnvironment (); // clean environnent
dce.AddArgument ("-s"); // server
dce.AddArgument ("-P"); // number of paralell servers
dce.AddArgument ("1");
apps = dce.Install (nodes.Get (2));
apps = dce.Install (nodes.Get (2));
apps.Start (Seconds (0.6));
```



DCE Setup

Step by step example
- iperf with ns-3 stack (IV)



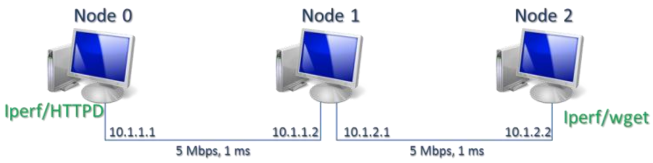

```
 // Simulation stop time
 Simulator::Stop (Seconds (40.0));

 // Run
 Simulator::Run ();

 // Stop
 Simulator::Destroy ();

 return 0;
}
```

Step by step example – iperf, ns-3

- **Generated**
  - elf-cache – program files
  - exitprocs – execution process information
  - files-0 files-2 – execution filesystem

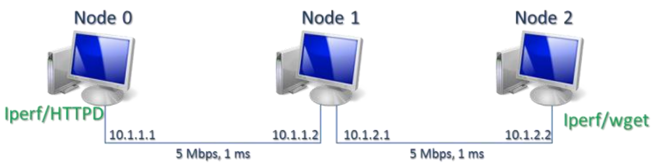
- **files-x**
  - var – “/root” of the machine
  - files-x/var/log/<pid>/
    - cmdline – command executed
    - status – execution information
    - stderr – standard error output
    - stdout – standard output
    - syslog – syslog output

Step by step example
- HTTP with ns-3 stack (I)




```
int main (int argc, char *argv[])
{
  // Node Container creation
  NodeContainer nodes;
  nodes.Create (3);

  // Linux stack creation
  InternetStackHelper stack;
  stack.Install (nodes);

  // For real time
  // GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3::RealtimeSimulatorImpl"));
  // GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));

  // Device and channel  creation
  PointToPointHelper p2p;
  p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps“));
  p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));
```

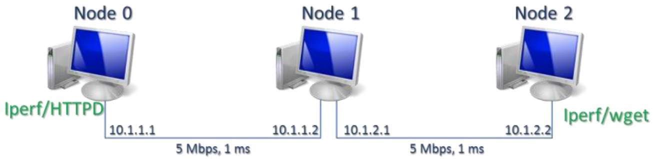
Step by step example
- HTTP with ns-3 stack (II)




```
// Node0-Node1 setup
 Ipv4AddressHelper address;
 address.SetBase ("10.1.1.0", "255.255.255.252"); // Node0-Node1 addresses

  NetDeviceContainer devices;
 devices = p2p.Install (nodes.Get (0), nodes.Get (1)); // connecting nodes
 Ipv4InterfaceContainer interfaces = address.Assign (devices); // assign addresses

 // Node1-Node2 setup
 devices = p2p.Install (nodes.Get (1), nodes.Get (2)); // connecting nodes
 address.SetBase ("10.1.2.0", "255.255.255.252"); // Node1-Node2 addresses
 interfaces = address.Assign (devices); // assign addresses

 // setup ip routes
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

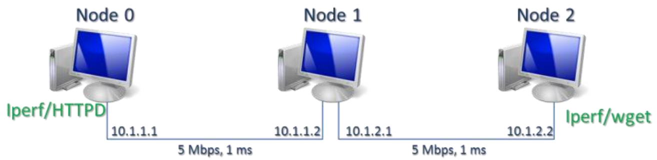
Step by step example
- HTTP with ns-3 stack (III)

```
// Launch the server HTTP
 dce.SetBinary ("thttpd");
 dce.ResetArguments ();  // clean arguments
 dce.ResetEnvironment (); // clean environnent
 dce.SetUid (1);  // Set httpd for super user execution
 dce.SetEuid (1);
 apps = dce.Install (nodes.Get (0)); // install http daemon
 apps.Start (Seconds (1)); // start time

 // Launch the client WGET
 dce.SetBinary ("wget");
dce.ResetArguments ();  // clean arguments
 dce.ResetEnvironment (); // clean environnent
 dce.AddArgument ("-r"); // recursive wget
 dce.AddArgument ("http://10.1.1.1/index.html");
 apps = dce.Install (nodes.Get (2));
 apps.Start (Seconds (2)); // start time
```



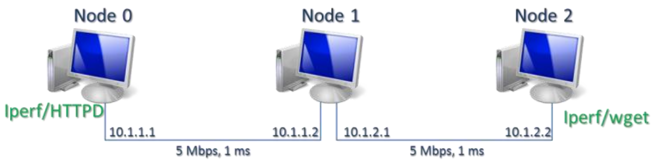
DCE Setup

Step by step example
- HTTP with ns-3 stack (IV)



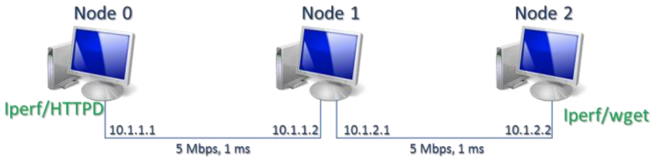

```
 // Simulation stop time
 Simulator::Stop (Seconds (40.0));

 // Run
 Simulator::Run ();

 // Stop
 Simulator::Destroy ();

 return 0;
}
```

Step by step example
- iperf with linux stack (I)




```
int main (int argc, char *argv[])
{
  // Node Container creation
  NodeContainer nodes;
  nodes.Create (3);

  // Linux stack creation
  dceManager.SetNetworkStack ("ns3::LinuxSocketFdFactory", "Library", StringValue ("liblinux.so"));
  LinuxStackHelper stack;
  stack.Install (nodes);

 // For real time
  // GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3::RealtimeSimulatorImpl"));
  // GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));

  // Device and channel  creation
  PointToPointHelper p2p;
  p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps“));
  p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));
```

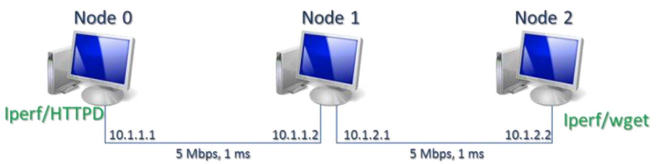
Step by step example
- iperf with linux stack (II)




```
// Node0-Node1 setup
 Ipv4AddressHelper address;
 address.SetBase ("10.1.1.0", "255.255.255.252"); // Node0-Node1 addresses

  NetDeviceContainer devices;
 devices = p2p.Install (nodes.Get (0), nodes.Get (1)); // connecting nodes
 Ipv4InterfaceContainer interfaces = address.Assign (devices); // assign addresses

 // Node1-Node2 setup
 devices = p2p.Install (nodes.Get (1), nodes.Get (2)); // connecting nodes
 address.SetBase ("10.1.2.0", "255.255.255.252"); // Node1-Node2 addresses
 interfaces = address.Assign (devices); // assign addresses

 // setup ip routes
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

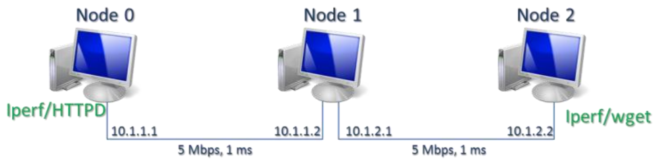
Step by step example
- iperf with linux stack (III)

```
DceManagerHelper dceManager;
dceManager.Install (nodes);

DceApplicationHelper dce;
ApplicationContainer apps;
 dce.SetStackSize (1 << 20); // 1MB stack

dce.SetBinary ("iperf"); // Launch iperf client on node 0
dce.ResetArguments ();  // clean arguments
dce.ResetEnvironment (); // clean environnent
dce.AddArgument ("-c");  // client
dce.AddArgument ("10.1.2.2"); //target machine address
dce.AddArgument ("-i"); // interval
dce.AddArgument ("1");
dce.AddArgument ("--time"); // how long
dce.AddArgument ("10");
 apps = dce.Install (nodes.Get (0)); //install application
apps.Start (Seconds (0.7)); //start at 0.7 simulation time
apps.Stop (Seconds (20)); //stop at 20s        simulation time

dce.SetBinary ("iperf"); // Launch iperf server on node 2
dce.ResetArguments (); // clean arguments
dce.ResetEnvironment (); // clean environnent
dce.AddArgument ("-s"); // server
dce.AddArgument ("-P"); // number of paralell servers
dce.AddArgument ("1");
apps = dce.Install (nodes.Get (2));
apps = dce.Install (nodes.Get (2));
apps.Start (Seconds (0.6));
```



DCE Setup
(Similar to the ns-3 stack one)

Step by step example
- iperf with linux stack (IV)




```
 // Simulation stop time
 Simulator::Stop (Seconds (40.0));

 // Run
 Simulator::Run ();

 // Stop
 Simulator::Destroy ();

 return 0;
}
```