ns-3 Training

Emulation

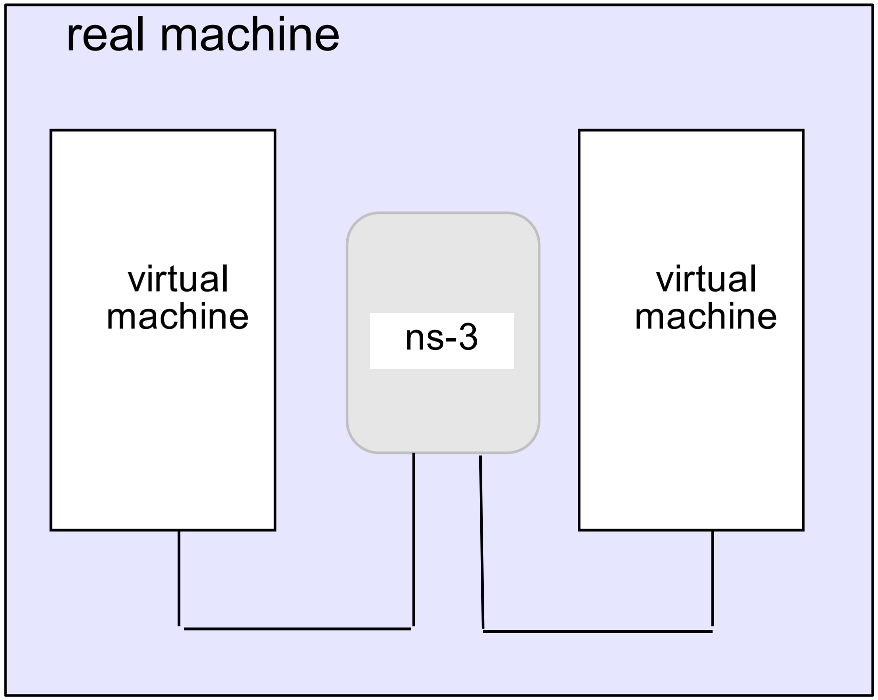
Outline

- Main emulation devices
  - Tap Bridge
  - FdNetDevice
  - NetmapNetDevice (coming soon)

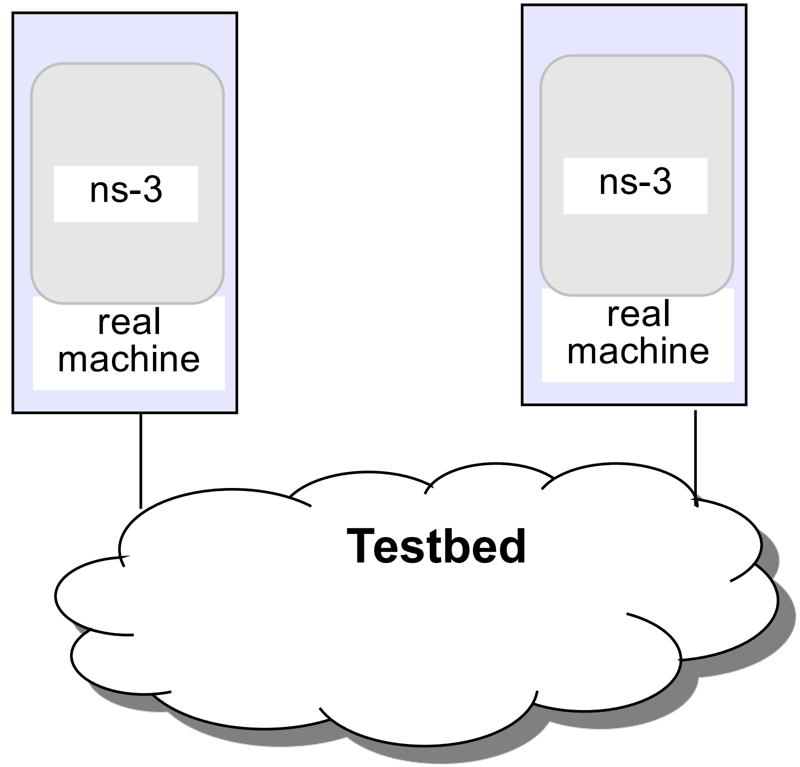
Emulation support

- 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

ns-3 emulation modes



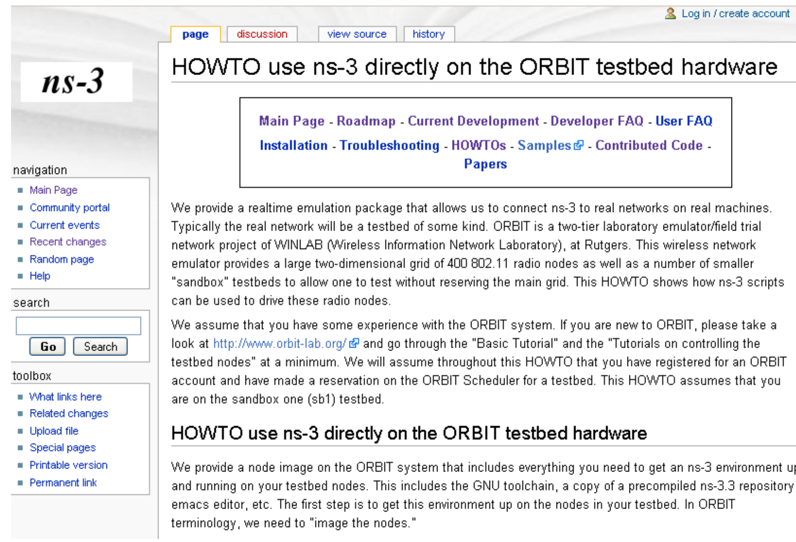
1) ns-3 interconnects real or virtual machines



2) testbeds interconnect ns-3 stacks

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.

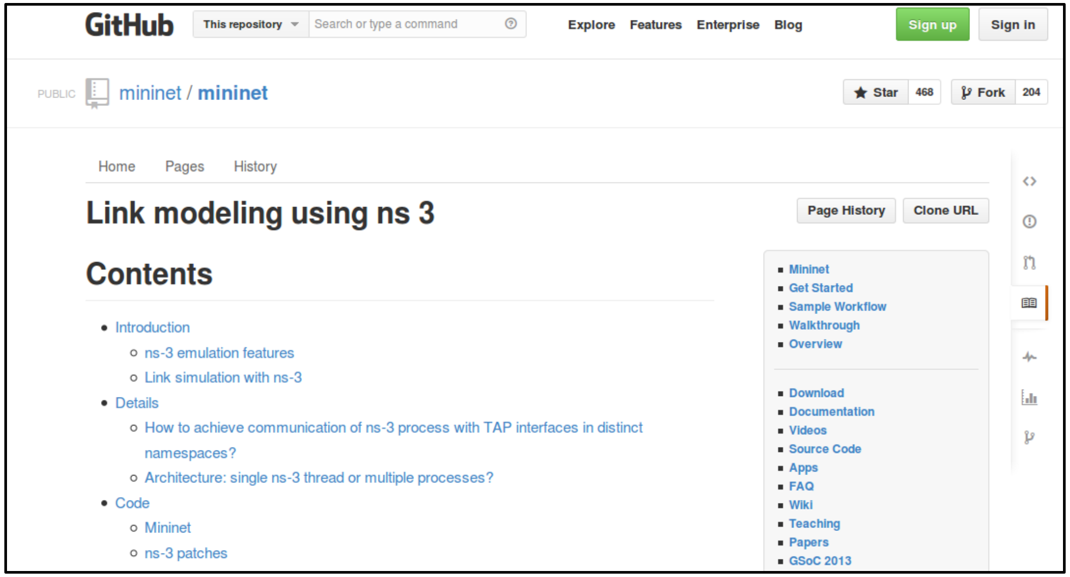
We assume that you have some experience with the ORBIT system. If you are new to ORBIT, please take a look at http://www.orbit-lab.org/ and go through the "Basic Tutorial" and the "Tutorials on controlling the testbed nodes" at a minimum. We will assume throughout this HOWTO that you have registered 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

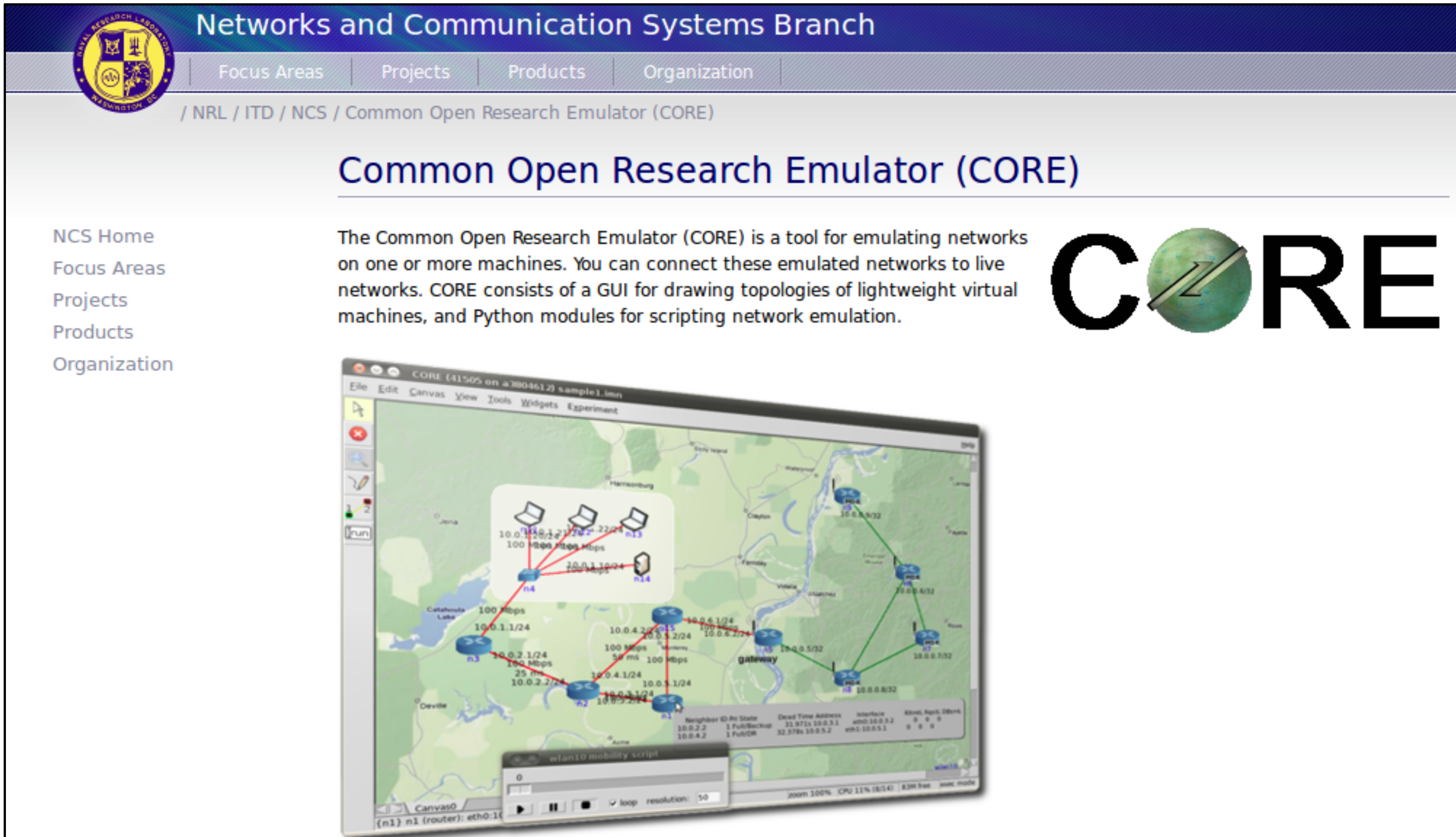
We provide a node image on the ORBIT system that includes everything you need to get an ns-3 environment up and running on your testbed nodes. This includes the GNU toolchain, a copy of a precompiled ns-3.3 repository, emacs editor, etc. The first step is to get this environment up on the nodes in your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

Example use case: 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



CORE emulator



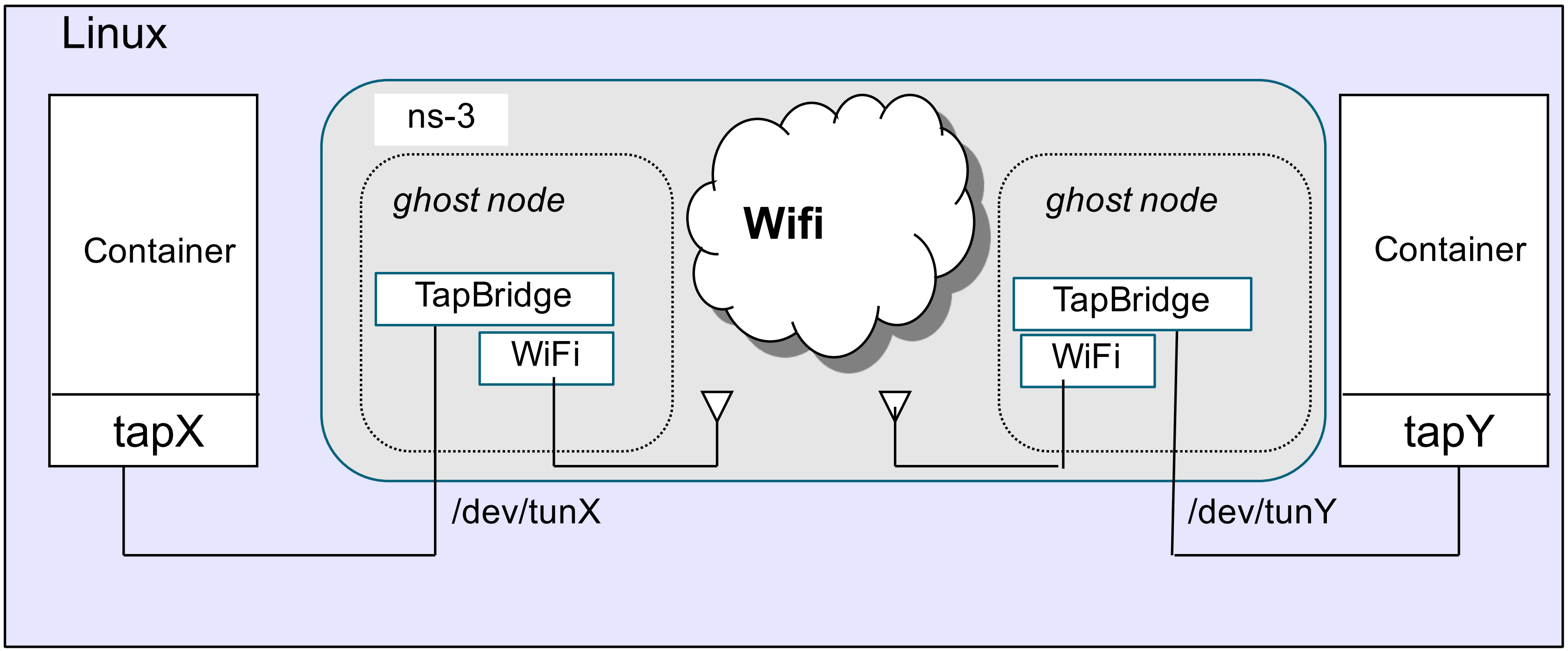
NETWORK SIMULATOR

Emulation Devices

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

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

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

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

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

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

Netmap NetDevice coming soon

- 2017 ESA Summer of Code project by Pasquale Imputato
- Netmap allows ns-3 to gain direct access to the network device, bypassing the host networking stack mapingthe device memory in user space area.
- Support for flow control and BQL (see discussion in traffic control presentation)
- Pasquale reports that the pps achievable with the NetmapNetDevice at lowest layer (i.e., the write method) is up to 1.38 Mpps on the e1000e adapter on the i7 cpu at full frequency of 3.8 GHZ.
- Latency is also reduced due to better flow control

ns-3 Training

Distributed simulation

(some slides/images credit due to Peter Barnes)

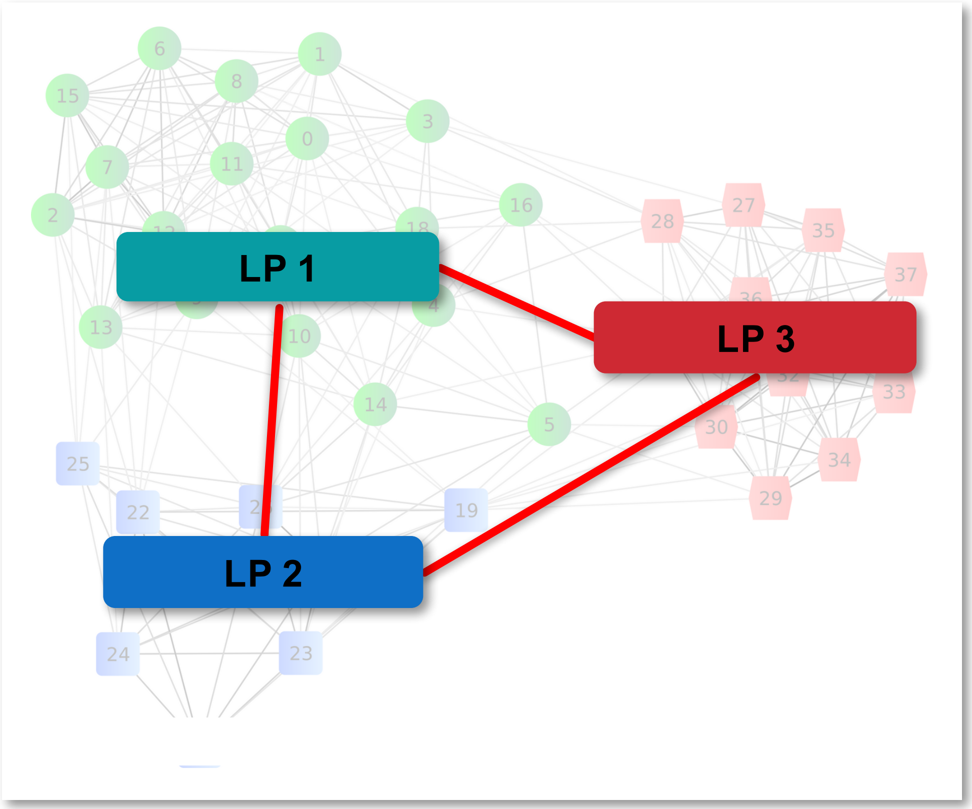
Distributed simulation

- By default, ns-3 runs a single threaded event loop on a single CPU core
  - Multi-core machines or multiple machines not used
- Performance is limited (CPU cycles and memory)
  - ~10^4 packet receives/wall clock second/core *
  - heavyweight applications may consume memory (DCE, routing tables in very large topologies)

* Peter Barnes, WNS3 training 2016

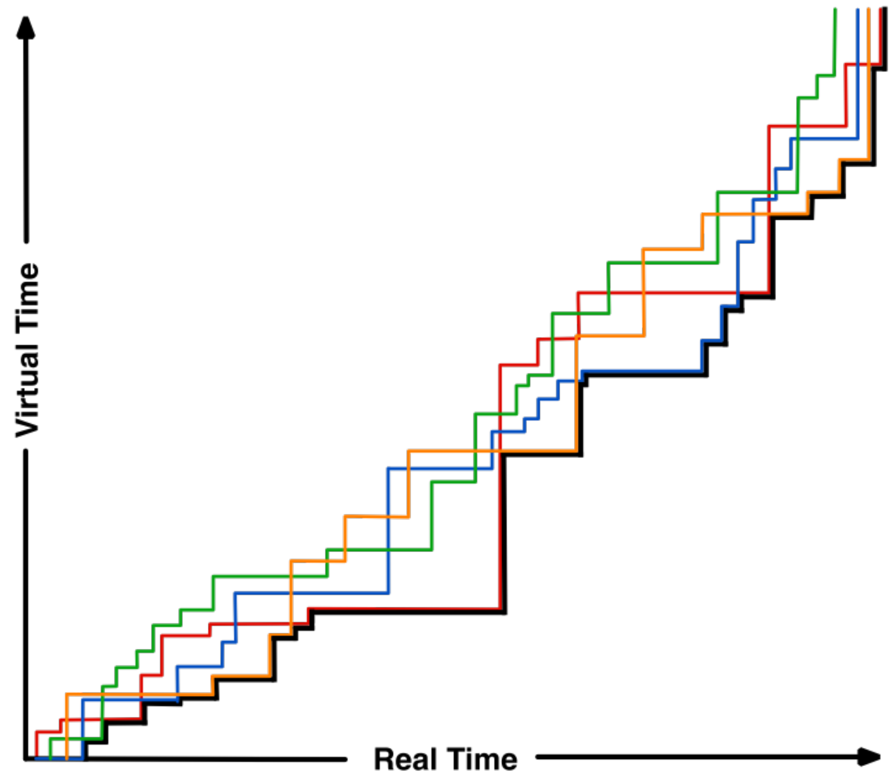
Distributed simulation

- Decompose model into *Logical Processes*
  - Separate objects and event queues
  - Execute independently
  - Events for other LPs become messages
  - ~ MPI Ranks



Virtual time evolution

- Sometimes ahead in virtual time, sometimes behind
- Time evolution constrained by slowest LP
- No causality violations
  - Hallmark of conservative execution



Using MPI in ns-3

- Need to install MPI on the host system (OpenMPI or MPICH libraries)
- Need to pass '--enable-mpi' to './waf configure'
  - Followed by usual build
- Only point-to-point links may be used to separate LPs
  - needs some level of propagation delay to support lookahead in each LP
  - splitting of wireless channels not presently supported
- The ns-3 scenario has to be constructed by assigning nodes to LPs manually

Running with ‘mpirun’

- waf can’t distinguish sequential and parallel
  - Need to specify `mpirun` and number of ranks explicitly

Running Parallel Scripts with `waf` and `mpirun`

```
$ ./waf --run simple-distributed
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.118s)
This simulation requires 2 and only 2 logical processors.
Command [‘build/debug/src/mpi/examples/ns3-dev-simple-distributed-debug'] exited with code 1

#  Multiple ranks on a single computer:
$ ./waf --run simple-distributed --command-template="mpirun -np 2 %s"
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.104s)
At time 1.02264s packet sink received 512 bytes from 10.1.1.1 port 49153 total Rx 512 bytes
At time 1.0235s packet sink received 512 bytes from 10.1.2.1 port 49153 total Rx 512 bytes
At time 1.02437s packet sink received 512 bytes from 10.1.3.1 port 49153 total Rx 512 bytes
At time 1.02524s packet sink received 512 bytes from 10.1.4.1 port 49153 total Rx 512 bytes

# Multiple computers:
$ mpirun –np 2 ./waf –run simple-distributed
```

1. Include mpi-module.h
2. Same topology, split across Point-to-point link

examples/tutorial/third.cc

```
* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
*
*
*
*
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

include "ns3/core-module.h"
include "ns3/point-to-point-module.h"
include "ns3/network-module.h"
include "ns3/applications-module.h"
include "ns3/wifi-module.h"
include "ns3/mobility-module.h"
include "ns3/csma-module.h"
include "ns3/internet-module.h"

/ Default Network Topology
/
/   Wifi 10.1.3.0
/                 AP
/  *    *    *    *
/  |    |    |    |    10.1.1.0
/ n5   n6   n7   n0 -------------- n1   n2   n3   n4
/                   point-to-point  |    |    |    |
/                                   ================
/                                     LAN 10.1.2.0

sing namespace ns3;

S_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

nt
ain (int argc, char *argv[])

 bool verbose = true;
 uint32_t nCsma = 3;
 uint32_t nWifi = 3;
 bool tracing = false;
```

1

2

src/mpi/examples/third-distributed.cc

```
* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

rogram is free software; you can redistribute it and/or modify
er the terms of the GNU General Public License version 2 as
hed by the Free Software Foundation;

* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

include "ns3/core-module.h"
include "ns3/point-to-point-module.h"
include "ns3/network-module.h"
include "ns3/applications-module.h"
include "ns3/wifi-module.h"
include "ns3/mobility-module.h"
include "ns3/csma-module.h"
include "ns3/internet-module.h"
include "ns3/mpi-module.h"

/ Default Network Topology
/ (same as third.cc from tutorial)
/ Distributed simulation, split along the p2p link
/ Number of wifi or csma nodes can be increased up to 250
/                           |
/                  Rank 0   |   Rank 1
/ -----------------------|--------------------------
/   Wifi 10.1.3.0
/                 AP
/  *    *    *    *
/  |    |    |    |    10.1.1.0
/ n5   n6   n7   n0 -------------- n1   n2   n3   n4
/                   point-to-point  |    |    |    |
/                                   ================
/                                     LAN 10.1.2.0

sing namespace ns3;

S_LOG_COMPONENT_DEFINE ("ThirdExampleDistributed");
```

examples/tutorial/third.cc

```
//   |    |    |    |     10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                   point-to-point  |    |    |    |
//                                   ================
//                                     LAN 10.1.2.0

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

int
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  bool tracing = false;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);

  // Check for valid number of csma or wifi nodes
  // 250 should be enough, otherwise IP addresses
  // soon become an issue
  if (nWifi > 250 || nCsma > 250)
    {
      std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::
      return 1;
    }

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
    }
```

src/mpi/examples/third-distributed.cc

```
//                   Rank 0    |    Rank 1
// -------------------------------|----------------------------
                 .3.0
                   AP
               *    *
               |    |     10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                   point-to-point  |    |    |    |
//                                   ================
//                                     LAN 10.1.2.0

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdExampleDistributed");

int
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  bool tracing = false;
  bool nullmsg = false;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);
  cmd.AddValue ("nullmsg", "Enable the use of null-message synchronization",

  cmd.Parse (argc,argv);

  // Check for valid number of csma or wifi nodes
  // 250 should be enough, otherwise IP addresses
  // soon become an issue
  if (nWifi > 250 || nCsma > 250)
    {
      std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::
      return 1;
    }

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

1. Different log component name
2. Command line argument to select Null message

1

2

examples/tutorial/third.cc

src/mpi/examples/third-distributed.cc

1. Condition on NS3_MPI
2. Null message selector
3. Initialize MPI
4. Get rank #, number of ranks
5. Check number of ranks
6. Use symbolic names for each rank
7. Create point-to-point nodes

```
  CommandLine cmd;

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
    }

  NodeContainer p2pNodes;
  p2pNodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer p2pDevices;
  p2pDevices = pointToPoint.Install (p2pNodes);

  NodeContainer csmaNodes;
  csmaNodes.Add (p2pNodes.Get (1));
  csmaNodes.Create (nCsma);

  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);

  NodeContainer wifiStaNodes;
  wifiStaNodes.Create (nWifi);
```

```
  // Sequential fallback values
  uint32_t systemId = 0;
  uint32_t systemCount = 1;

#ifdef NS3_MPI

  // Distributed simulation setup; by default use granted time window algorithm.
  if(nullmsg)
    {
      GlobalValue::Bind ("SimulatorImplementationType",
                         StringValue ("ns3::NullMessageSimulatorImpl"));
    }
  else
    {
      GlobalValue::Bind ("SimulatorImplementationType",
                         StringValue ("ns3::DistributedSimulatorImpl"));
    }

  MpiInterface::Enable (&argc, &argv);

  systemId = MpiInterface::GetSystemId ();
  systemCount = MpiInterface::GetSize ();

  // Check for valid distributed parameters.
  // Must have 2 and only 2 Logical Processors (LPs)
  if (systemCount != 2)
    {
      std::cout << "This simulation requires 2 and only 2 logical processors. <<
      return 1;
    }

#endif // NS3_MPI

  // System id of Wifi side
  uint32_t systemWifi = 0;

  // System id of CSMA side
  uint32_t systemCsma = systemCount - 1;

  NodeContainer p2pNodes;
  Ptr<Node> p2pNode1 = CreateObject<Node> (systemWifi); // Create node with rank
  Ptr<Node> p2pNode2 = CreateObject<Node> (systemCsma); // Create node with rank
  p2pNodes.Add (p2pNode1);
  p2pNodes.Add (p2pNode2);

  PointToPointHelper pointToPoint;
```

examples/tutorial/third.cc

```
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
```

1. Create CSMA nodes on one rank
2. Create Wifi nodes on another rank

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",
```

src/mpi/examples/third-distributed.cc

```
NodeContainer p2pNodes;
Ptr<Node> p2pNode1 = CreateObject<Node> (systemWifi); // Create node with ra
Ptr<Node> p2pNode2 = CreateObject<Node> (systemCsma); // Create node with rank 1
p2pNodes.Add (p2pNode1);
p2pNodes.Add (p2pNode2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma, systemCsma);  // Create csma nodes with rank 1

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi, systemWifi); // Create wifi nodes with rank 0
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",
```

1

2

examples/tutorial/third.cc

```
stack.Install (csmaNodes);
...
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps =
  echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing == true)
  {
    pointToPoint.EnablePcapAll ("third-distributed-wifi");
    phy.EnablePcap ("third-distributed-wifi", apDevices.Get (0));
    csma.EnablePcap ("third-distributed-wifi", csmaDevices.Get (0), true);

    pointToPoint.EnablePcapAll ("third-distributed-csma");
    phy.EnablePcap ("third-distributed-csma", apDevices.Get (0));
```

1. Install devices, addresses and Internet stack everywhere
2. Install applications only on rank-local nodes

src/mpi/examples/third-distributed.cc

```
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);

// If this rank is systemCsma,
// it should contain the server application,
// since it is on one of the csma nodes
if (systemId == systemCsma)
  {
    UdpEchoServerHelper echoServer (9);

    ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma))
    serverApps.Start (Seconds (1.0));
    serverApps.Stop (Seconds (10.0));
  }

// If this rank is systemWifi
// it should contain the client application,
// since it is on one of the wifi nodes
if (systemId == systemWifi)
  {
    UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
    echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
    echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
    echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

    ApplicationContainer clientApps =
      echoClient.Install (wifiStaNodes.Get (nWifi - 1));
    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));
  }

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing == true)
  {
```

1. Enable PCAP tracing on local nodes?
2. Close MPI cleanly

examples/tutorial/third.cc

```cpp
  address.SetBase ("10.1.1.0", "255.255.255.0");

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps =
    echoClient.Install (wifiStaNodes.Get (nWifi - 1));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Stop (Seconds (10.0));

  if (tracing == true)
    {
      pointToPoint.EnablePcapAll ("third-distributed-wifi");
      phy.EnablePcap ("third-distributed-wifi", apDevices.Get (0));
      csma.EnablePcap ("third-distributed-wifi", csmaDevices.Get (0), true);

      pointToPoint.EnablePcapAll ("third-distributed-csma");
      phy.EnablePcap ("third-distributed-csma", apDevices.Get (0));
      csma.EnablePcap ("third-distributed-csma", csmaDevices.Get (0), true);
    }

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

src/mpi/examples/third-distributed.cc

```cpp
    {
      UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
      echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
      echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
      echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

      ApplicationContainer clientApps =
        echoClient.Install (wifiStaNodes.Get (nWifi - 1));
      clientApps.Start (Seconds (2.0));
      clientApps.Stop (Seconds (10.0));
    }

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Stop (Seconds (10.0));

  if (tracing == true)
    {
      // Depending on the system Id (rank), the pcap information
      // traced will be different.  For example, the ethernet pcap
      // will be empty for rank0, since these nodes are placed on
      // on rank 1.  All ethernet traffic will take place on rank 1.
      // Similar differences are seen in the p2p and wireless pcaps.
      if (systemId == systemWifi)
        {
          pointToPoint.EnablePcapAll ("third-distributed-wifi");
          phy.EnablePcap ("third-distributed-wifi", apDevices.Get (0));
          csma.EnablePcap ("third-distributed-wifi", csmaDevices.Get (0), true);
        }
      else // systemCsma
        {
          pointToPoint.EnablePcapAll ("third-distributed-csma");
          phy.EnablePcap ("third-distributed-csma", apDevices.Get (0));
          csma.EnablePcap ("third-distributed-csma", csmaDevices.Get (0), true);
        }
    }

  Simulator::Run ();
  Simulator::Destroy ();

#ifdef NS3_MPI
  // Exit the MPI execution environment
  MpiInterface::Disable ();
#endif

  return 0;
}
```

Script Output–Identical

```
$ ./waf –run third
```

```
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.152s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9
```

```
$ ./waf --run third-distributed \
--command-template="mpirun -n 2 %s --tracing"
```

```
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.050s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9
```