Intelligent Wireless Network Group
Department of Computer Engineering
Faculty of Engineering, Kasetsart University
http://iwing.cpe.ku.ac.th

ns-3 Tutorial (Part III)
Hands-On:
Point-to-point and CSMA - Ethernet

JCSSE 2011's tutorials and workshops
Wednesday, 11 May 2011, 9:00 - 16:00

Time Table

- 09:00 - 10:15 ns-3 Introduction & Installation
- 10:15 - 10.30 Break
- 10:30 - 12:00 Hands-On:
  Point-to-point and CSMA (Ethernet)
- 12:00 - 13:00 Lunch
- 13:00 - 14:15 Hands-On:
  Wireless & Tracing System and Visualizing Results
- 14:15 - 14:30 Break
- 14:30 - 15:30 Demonstation:
  ns-3 protocol stack modification
- 15:30 - 16:00 Q&A

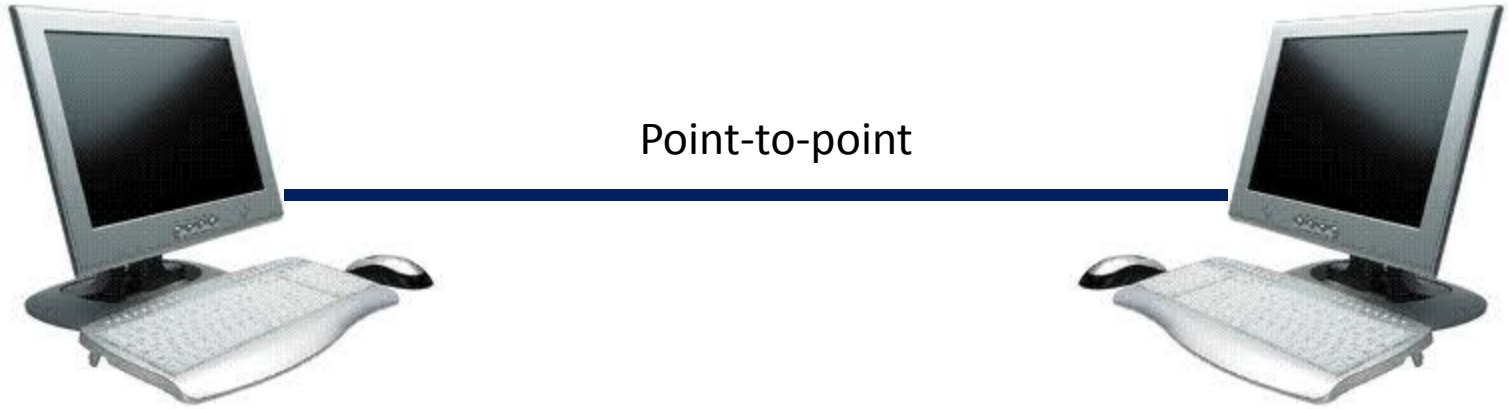
Exploring the directories

```
anan@AP-Moose:~/tarballs/ns-allinone-3.10/ns-3.10$ ls
AUTHORS        examples      ns3              src              VERSION     wutils.py
bindings       LICENSE       README           test.py          waf         wutils.pyc
build          #log.out#     RELEASE_NOTES    testpy-output    waf.bat
CHANGES.html   log.out       samples          testpy.supp      waf-tools
doc            #myfirst.tr#  scratch          utils            wscript
```

- cd examples/tutorial

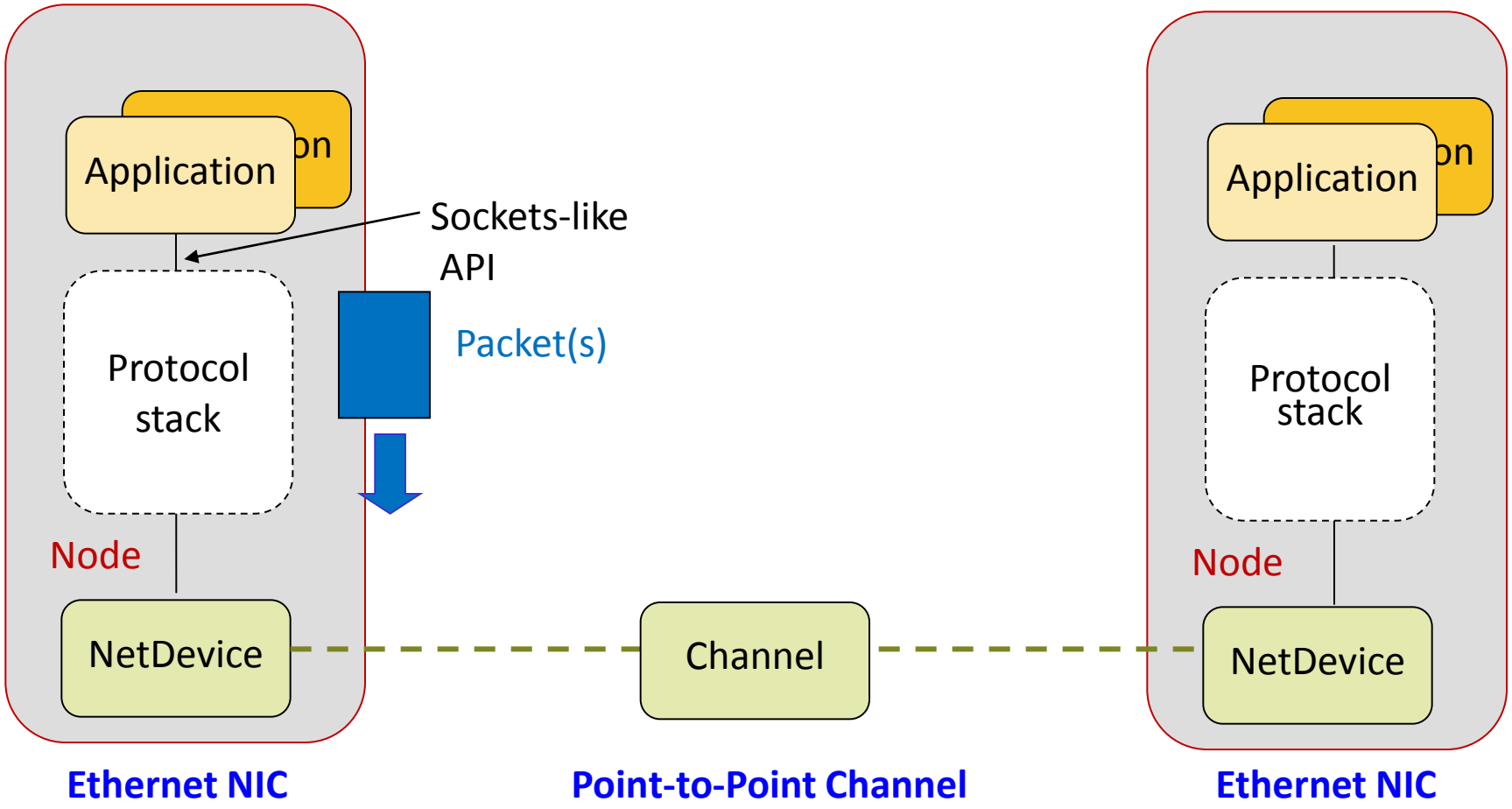
```
anan@AP-Moose:~/tarballs/ns-allinone-3.10/ns-3.10$ cd examples/tutorial/
anan@AP-Moose:~/tarballs/ns-allinone-3.10/ns-3.10/examples/tutorial$ ls
fifth.cc  first.py   hello-simulator.cc  sixth.cc  waf
first.cc  fourth.cc  second.cc           third.cc  wscript
```

Topology



Simple Example: **`first.cc`**

Topology in ns-3 (`first.cc`)



vi examples/tutorial/first.cc

emacs mode line: formatting conventions (coding style)

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

GNU General Public License

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published 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
 */
```

first.cc

Module Includes

```
#include "ns3/core-module.h"
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/helper-module.h"
```

C++ namespace called ns3

```
using namespace ns3;
```

Logging

```
NS_LOG_COMPONENT_DEFINE("FirstScriptExample");
```

first.cc

define a main function

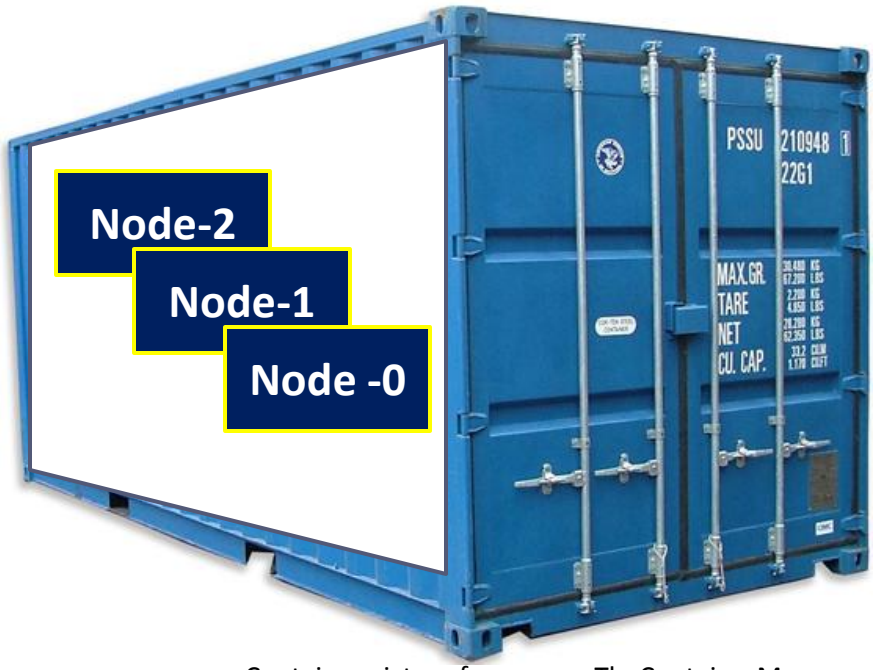
```
int
main (int argc, char *argv[])
{
```

enable two logging components

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

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

NodeContainer

- Provides a convenient way
  - to **create**, **manage** and **access** any Node objects
- Stores pointer to those objects internally
- The nodes as they stand in the script do nothing
  - To construct a topology is to connect our nodes together into a network



Container picture from www.TheContainerMan.com

PointToPointHelper

- To construct a point to point link
- Perform the low-level work required to put the link together
- Typically these two things **`NetDevice`** and **`Channel`** are intimately tied together (cannot expect to interchange)
  - Ethernet devices and CSMA channels
  - Wifi devices and wireless channels

first.cc

create the ns-3 Node objects

```
NodeContainer nodes;
nodes.Create (2);
```

instantiates a PointToPointHelper object on the stack

```
PointToPointHelper pointToPoint;
```

as the “DataRate” when it creates a PointToPointNetDevice object

```
pointToPoint.SetDeviceAttribute ("DataRate",
  StringValue ("5Mbps"));
```

as the “Delay” when it creates a PointToPointChannel object

```
pointToPoint.SetChannelAttribute ("Delay",
  StringValue ("2ms"));
```

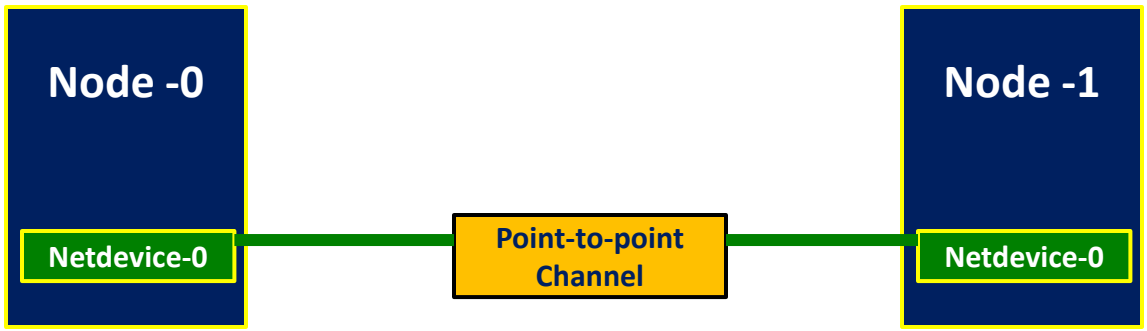
first.cc

create the NetDevice objects

```
NetDeviceContainer devices;
```

Install NICs to Nodes and link Point-to-point

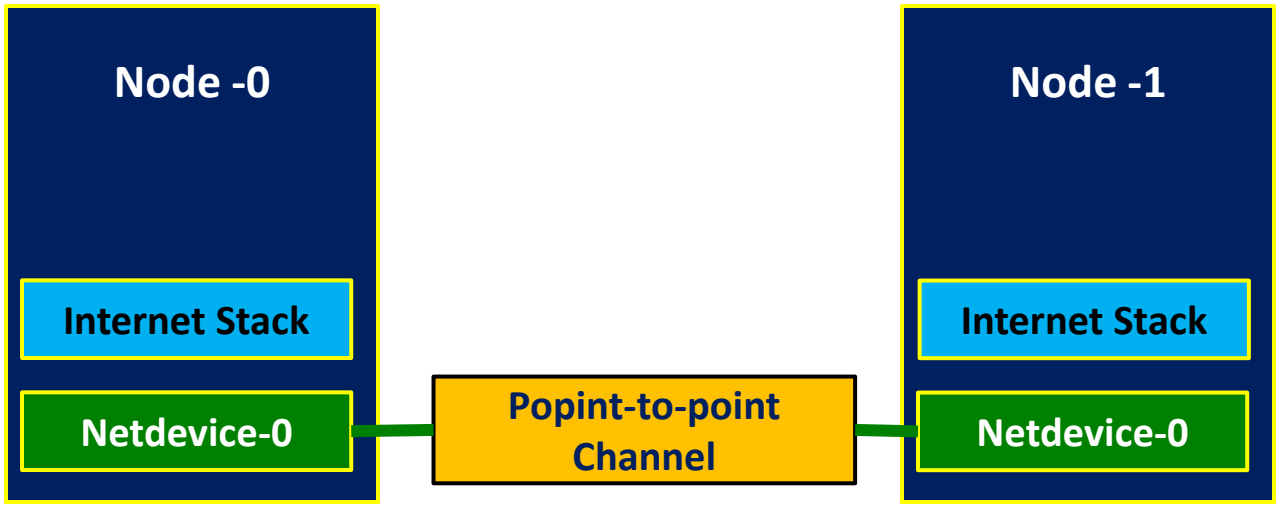
```
devices = pointToPoint.Install (nodes);
```



first.cc

install an Internet Stack
(TCP, UDP, IP, etc.) on each of the nodes in the node container

```
InternetStackHelper stack;
stack.Install (nodes);
```



first.cc

to associate the devices on our nodes with IP addresses

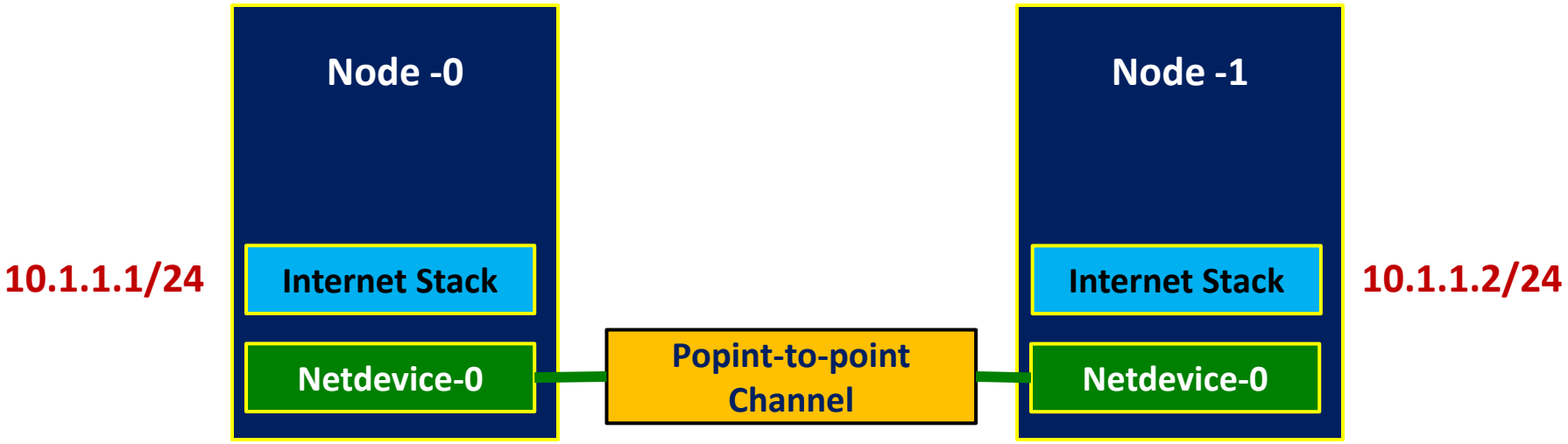
```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0","255.255.255.0");
```

- begin allocating IP addresses
  - from network 10.1.1.0 using the mask 255.255.255.0
  - In our case, 10.1.1.1/24 followed by 10.1.1.2/24

first.cc

performs the actual address assignment

```
Ipv4InterfaceContainer interfaces =
  address.Assign (devices);
```



first.cc

set up a UDP echo server application

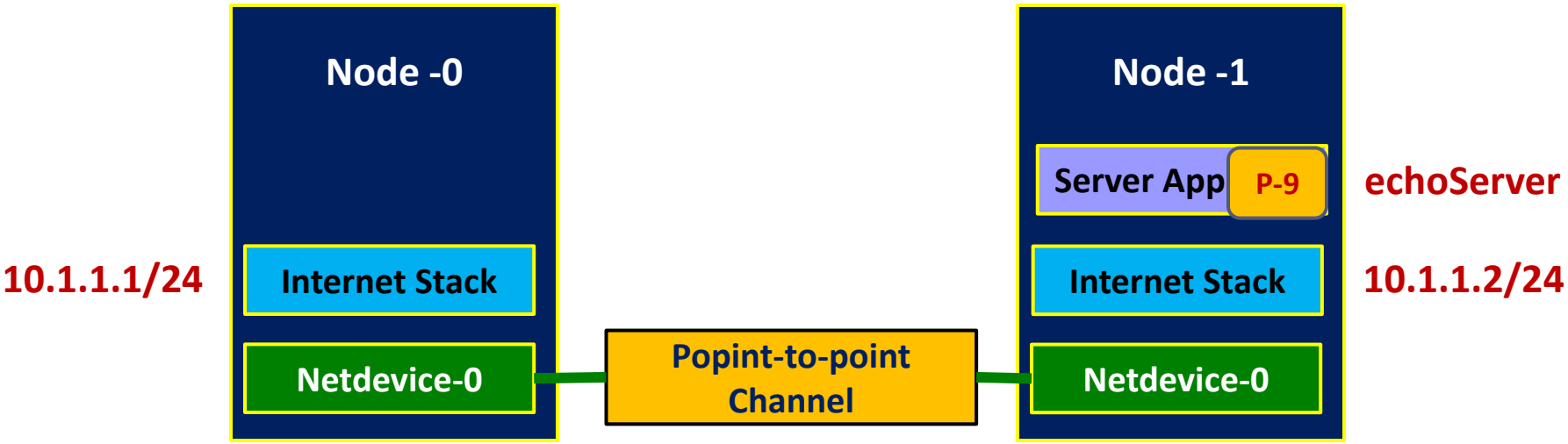
```
UdpEchoServerHelper echoServer (9);
```

- Set up a UDP **echo server application** on one of the nodes
- Require the **port number** as a parameter to the constructor

first.cc

install server application @node(1)

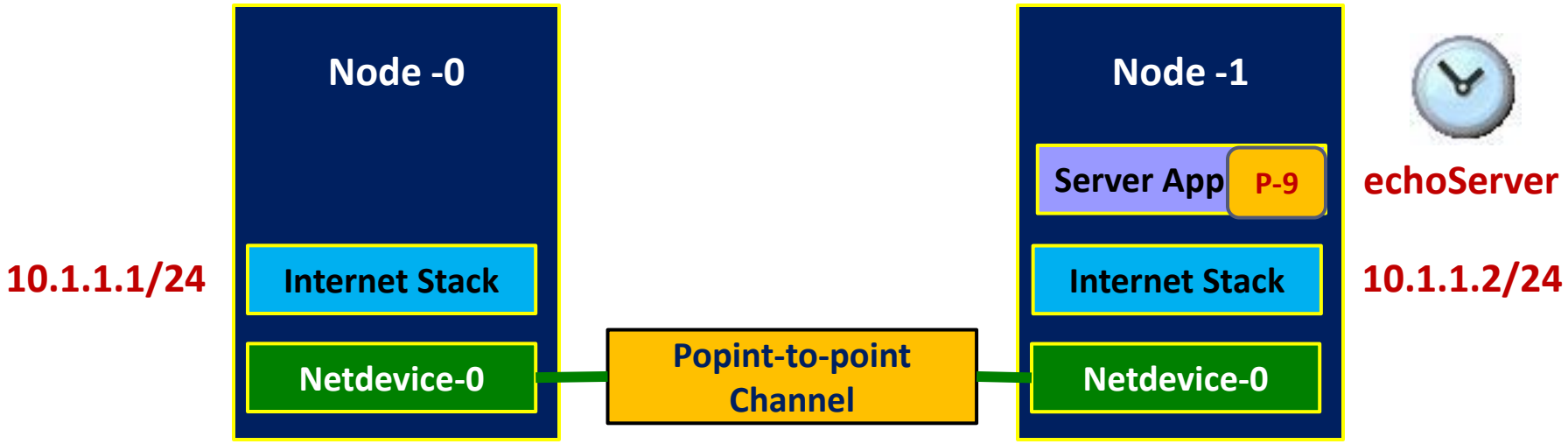
```
ApplicationContainer serverApps =
  echoServer.Install (nodes.Get (1));
```



first.cc

echo server application starts (enable itself) at 1 sec
and to Stop (disable itself) at 10 sec

```
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```



first.cc

Set remote add and port

```
UdpEchoClientHelper echoClient (interfaces.GetAddress(1),9);
```

- Pass parameters (to helper) to set the **Remote Address** and **Remote Port** for echoClient to connect
- Remote: 10.1.1.2 , Port 9

Set echo Client's attributes

```
echoClient.SetAttribute("MaxPackets", UintegerValue(1));
echoClient.SetAttribute("Interval", TimeValue (Seconds(1.)));
echoClient.SetAttribute("PacketSize", UintegerValue(1024));
```

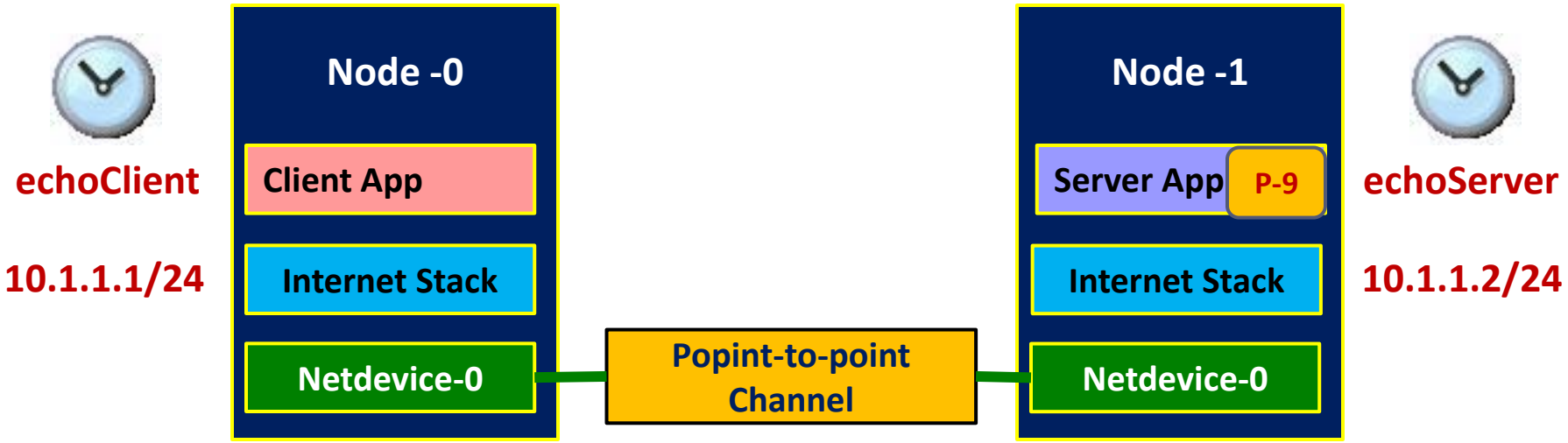
first.cc

install client application @node(0)

```
ApplicationContainer clientApps =
  echoClient.Install (nodes.Get (0));
```

Set echo Client's attributes

```
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```



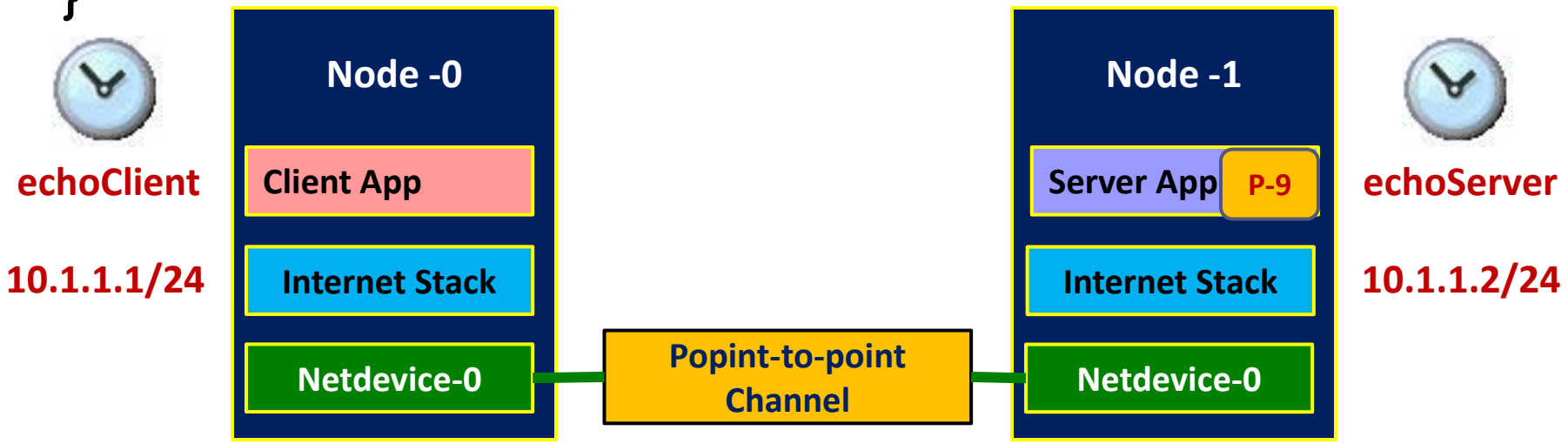
first.cc

to actually run the simulation

```
Simulator::Run ();
```

to clean up

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



Classes

- Container
  - `NodeContainer`
  - `NetDeviceContainer`
  - `Ipv4InterfaceContainer`
  - `ApplicationContainer`
- Helper
  - `PointToPointHelper`
  - `InternetStackHelper`
  - `Ipv4AddressHelper`
  - `UdpEchoServerHelper`
  - `UdpEchoClientHelper`

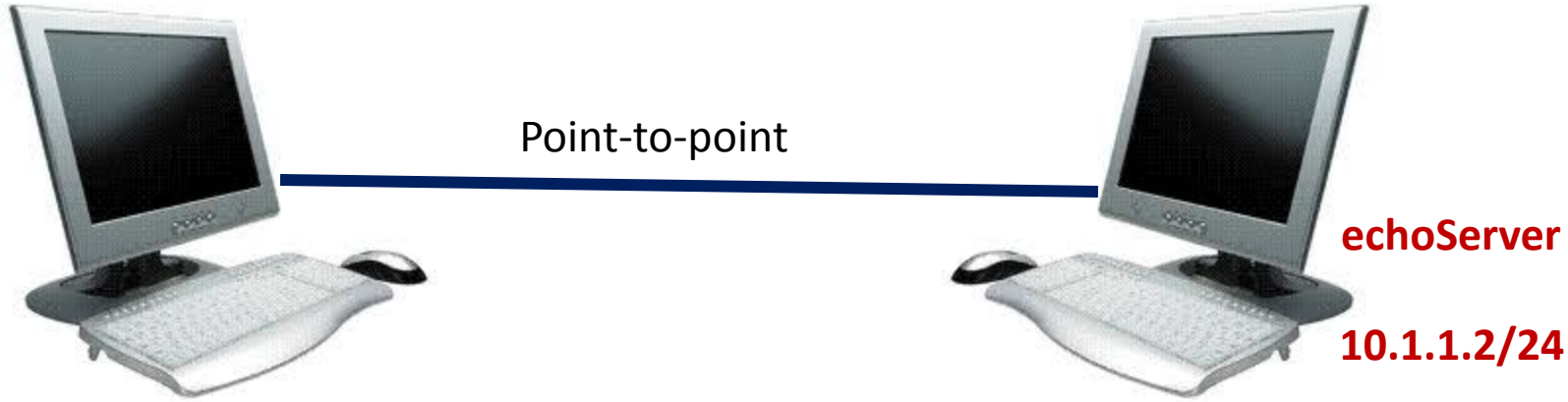
Building and Running Your Script

- Copy `examples/tutorial/first.cc` into the `scratch` directory
  - **`cp examples/tutorial/first.cc scratch/myfirst.cc`**
- Now build your script using **`waf`**
  - **`./waf`**
- Run it out of the scratch directory
  - **`./waf --run scratch/myfirst`**
- Output

```
Waf: Entering directory `/home/anan/tarballs/ns-allinone-3.10/ns-3.10/build'
[ 568/1204] cxx: scratch/myfirst.cc -> build/debug/scratch/myfirst_3.o
[1204/1204] cxx_link: build/debug/scratch/myfirst_3.o -> build/debug/scratch/myfirst
Waf: Leaving directory `/home/anan/tarballs/ns-allinone-3.10/ns-3.10/build'
'build' finished successfully (8.591s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

output

```
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

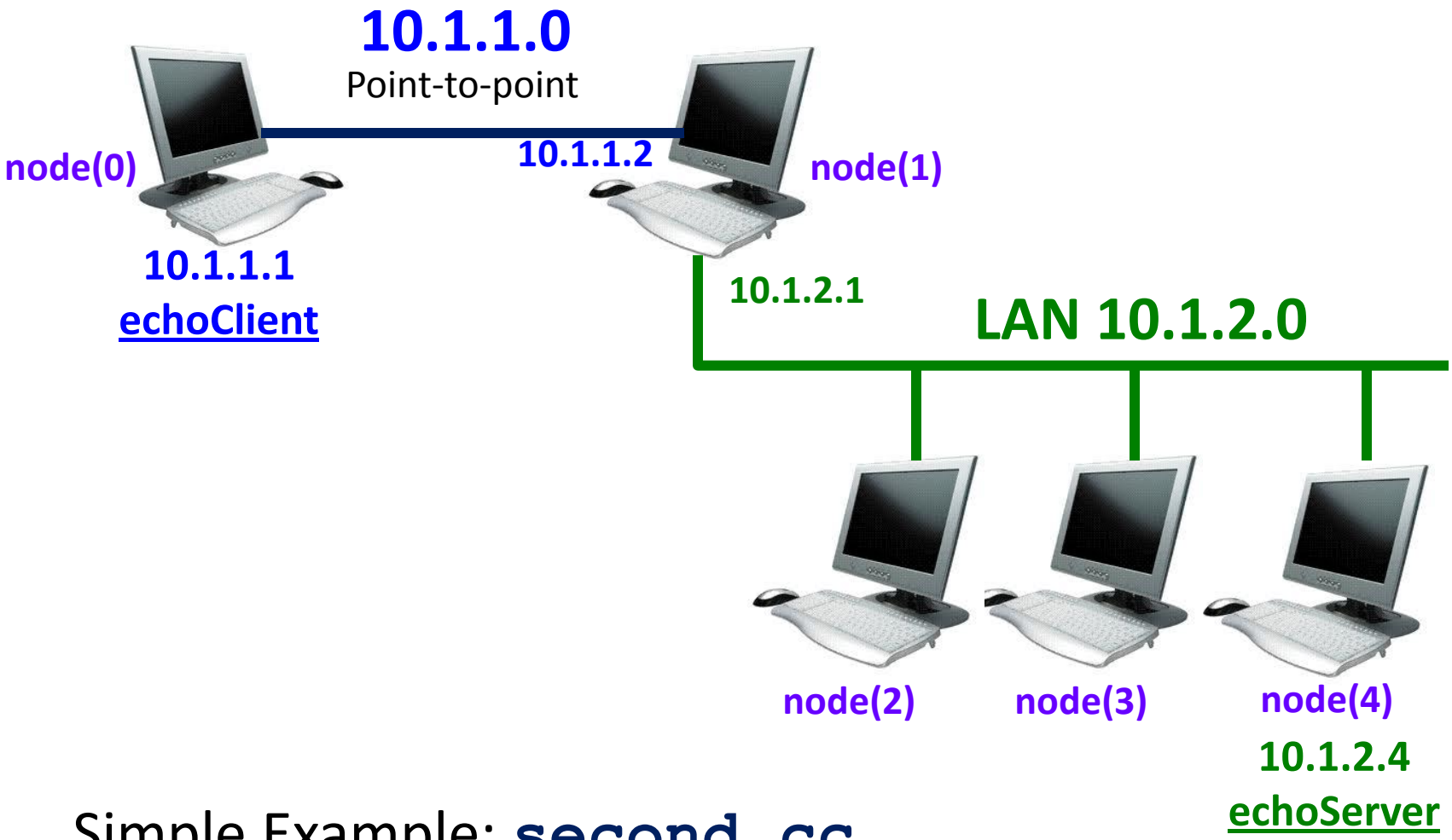


Logging

- Adding log in the code before creating nodes
  - `NS_LOG_INFO ("JCSSE 2011");`
- `./waf`
- `export NS_LOG=`
- `./waf --run scratch/myfirst`

- `export NS_LOG=FirstScriptExample=info`
- `./waf --run scratch/myfirst`

Move on …
Building a Bus Network Topology



Simple Example: `second.cc`

second.cc

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");

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

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

  cmd.Parse (argc,argv);

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

second.cc

```
 nCsma = nCsma == 0 ? 1 : nCsma;

NodeContainer p2pNodes;
p2pNodes.Create (2);

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

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

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

second.cc

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

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

InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);

Ipv4AddressHelper address;
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);
```

second.cc

```
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.)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

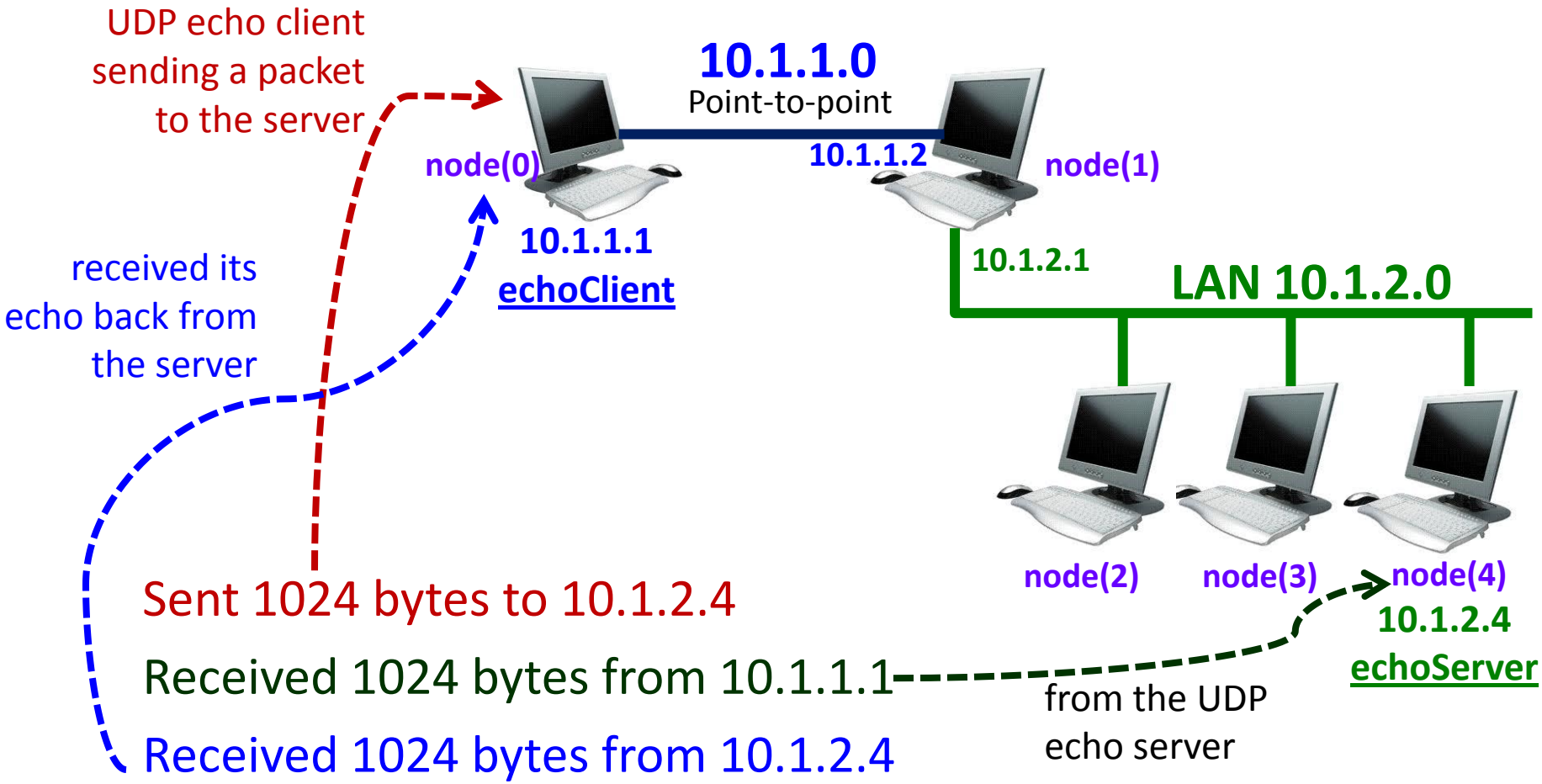
second.cc

```
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  pointToPoint.EnablePcapAll ("second");
  csma.EnablePcap ("second", csmaDevices.Get (1), true);

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

Building and Running Your Script

- Copy `examples/tutorial/second.cc` into the `scratch` directory
  - **`cp examples/tutorial/second.cc scratch/mysecond.cc`**
- Now build your script using **`waf`**
  - **`./waf`**
- Run it out of the scratch directory
  - **`./waf --run scratch/mysecond`**

Output... mysecond

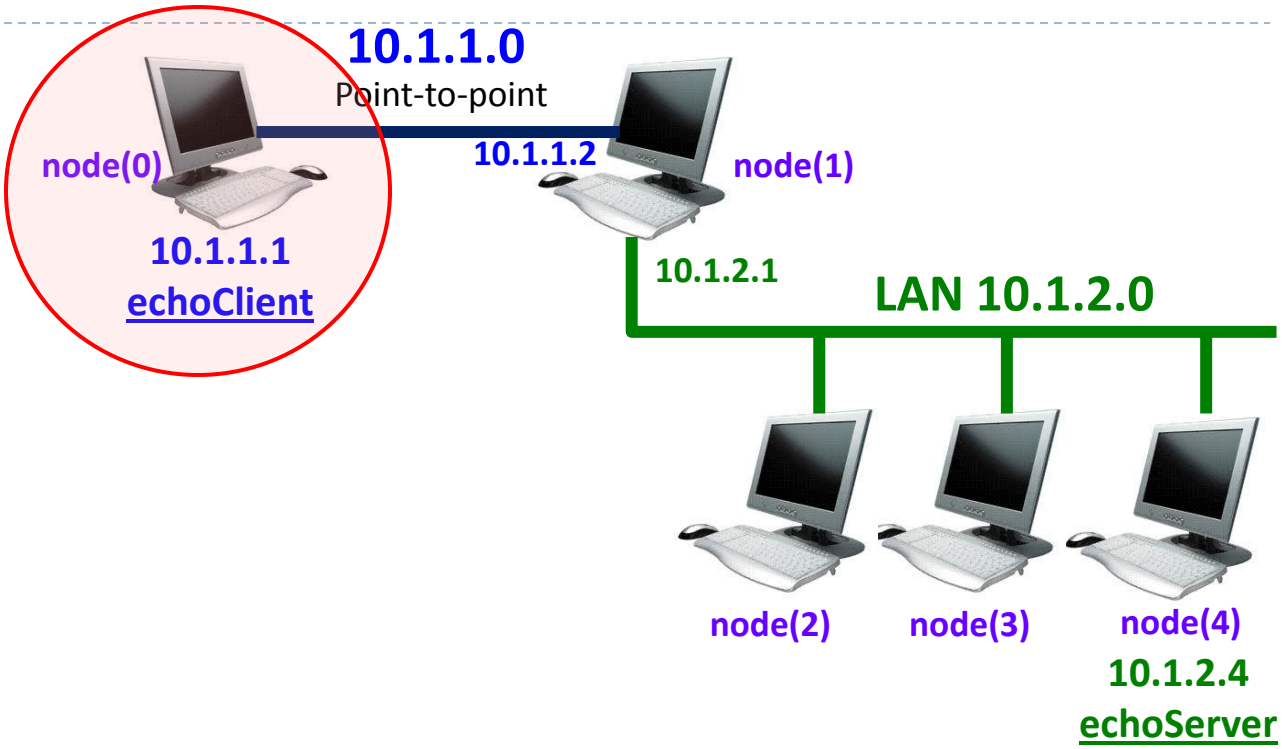


Pcap Output

```
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second",csmaDevices.Get(1),true);
```

- top level directory
  - **second-0-0.pcap**
  - **second-1-0.pcap**
  - **second-2-0.pcap**

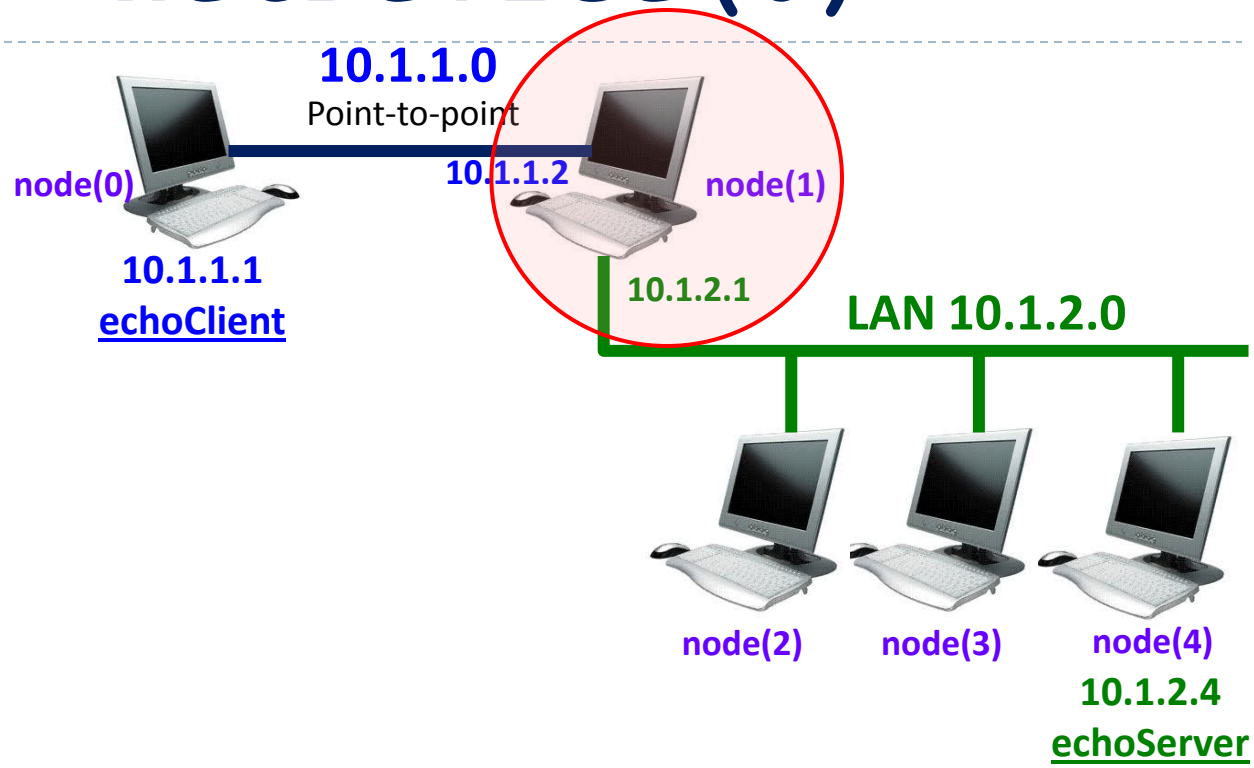
node(0)- netDevice(0)



- **tcpdump -nn -tt -r second-0-0.pcap**

```
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.007602 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

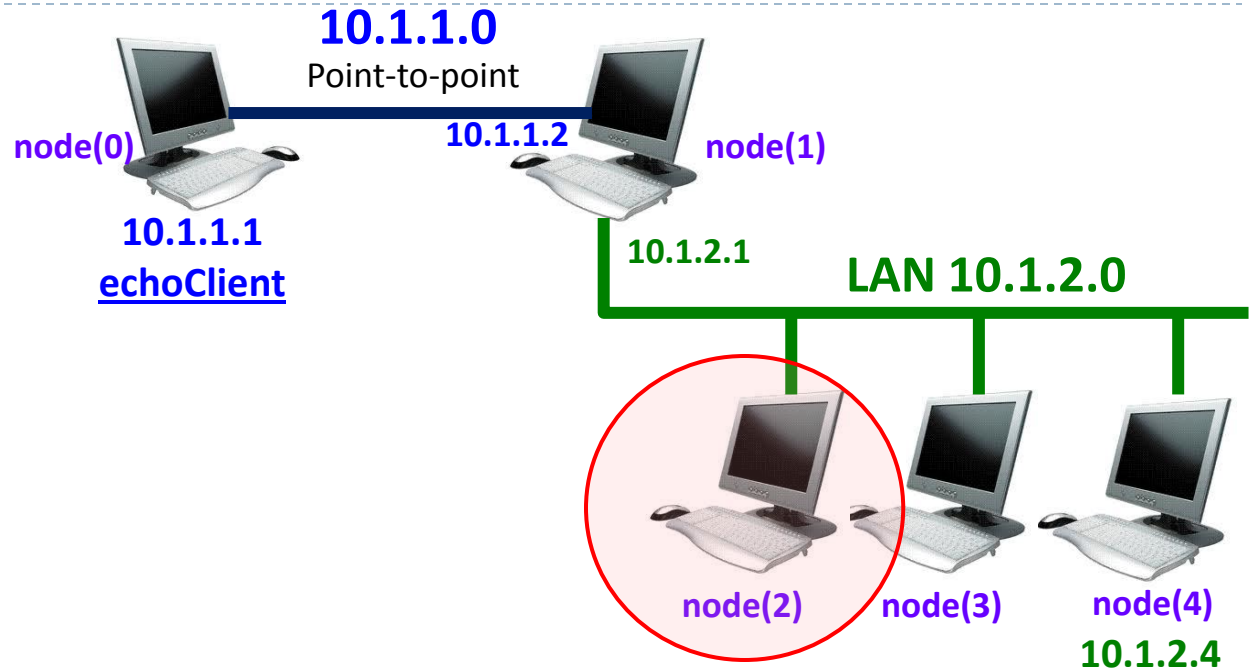
node(1)- netDevice(0)



- **tcpdump -nn -tt -r second-1-0.pcap**

```
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.003915 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

node(2)- netDevice(0)



- **tcpdump -nn -tt -r second-2-0.pcap**

```
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.003696 arp who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1
2.003707 arp reply 10.1.2.4 is-at 00:00:00:00:00:06
2.003801 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.003811 arp who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4
2.003822 arp reply 10.1.2.1 is-at 00:00:00:00:00:03
2.003915 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

More on second.cc

- `wireshark` **`second-2-0.pcap`**
- **`./waf --run "scratch/mysecond –nCsma=50"`**