Network Simulator (NS-3)

Anand Baswade

Motivation: (Network Analysis Techniques)

- **Analytical approaches**

  - mathematical analysis/modeling of systems

- **Simulations**

  - model the system at abstract level via software

  - various network simulators exist (e.g: NS3 , OPNET, QualNet)

- **Emulations**

  - HW component that behave like real system (e.g., mininet)

- **Measurements**

  - active (e.g.: ping, trace route) or passive (e.g: wireshark)

- **Experimentations**

  - experiment on a testbed for realism (eg: GENI, PlanetLab, OrbitLab)

Motivation for Network Simulations

Goals

- build software simulation model of networking systems

- to analyze/study/improve/develop network archs & protocols

Reasons

- real systems are expensive, complex, unavailable

Advantages

- relatively easy to setup, deploy and instrument

- reproducibility, scalability

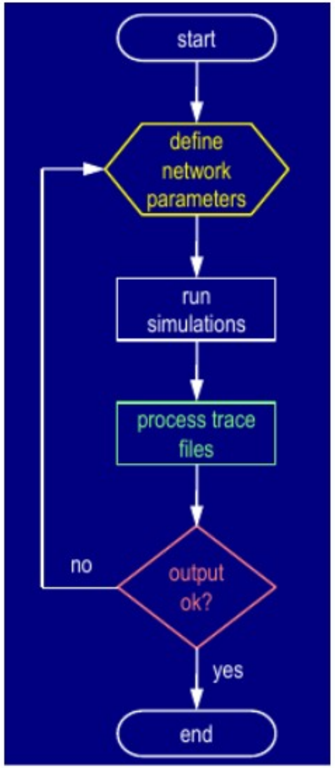
Disadvantages

- simplified view of complex interactions

- could be immensely misleading

Network Simulation Flowchart

- Define topology, protocols/algos, (channel/traffic) models, flows
- Specify initial values, start/end times
- Run simulations for many seeds
- Process output /trace files
- Measure avg. network performance: throughput, goodput, delay, jitter, loss, fairness, etc
- If not happy, repeat above steps again!



What is NS (or NS-2/3)?

- NS is a discrete-event network simulator for internet systems

- Protocol design, multiple levels of abstraction

- Written in C++, with bindings available for python

- NS has a companion network animator called nam
  - hence, has been called the nsnam project (www.nsnam.org)

NS-3: Introduction

- NS-3 is a free, open source packet-level discrete event network simulator available under GNU GPLv2 license for research, development, and use.

- Technical Goals
  - Build and maintain a simulation core aligned with the needs of modern networking research
  - Help to improve the technical rigor of network simulation practice

- NS-3 Timeline
  - NS-1   1990s
  - NS-2   1996
  - NS-3 Core development 2006-08 (Tom Henderson and Mathieu Lacage)
  - NS3.1 June 2008
  - Now Generally new release every Quarterly (fix bugs and new features, etc..)
  - NS3.39 July 5, 2023 (Recent Stable Release) You can install developer version to get access to latest modules like 5G/Wi-Fi 7 module, etc...

NS-3 uses 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/
- For those familiar with autotools:
  - configure -> ./waf -d [optimized|debug] configure
  - make -> ./waf

Hello World NS-3 Program

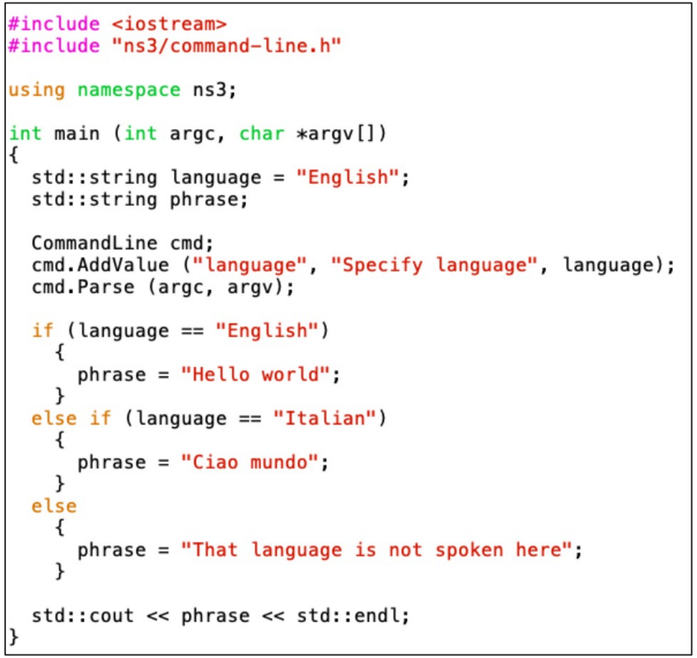
This is basic C++, except:

1) We are using methods/classes defined in ‘ns3’ namespace

2) The object ‘cmd’ is an instance of the CommandLine C++ class.

CommandLine exists in C++ namespace ‘ns3’.

CommandLine objects process command-line arguments.

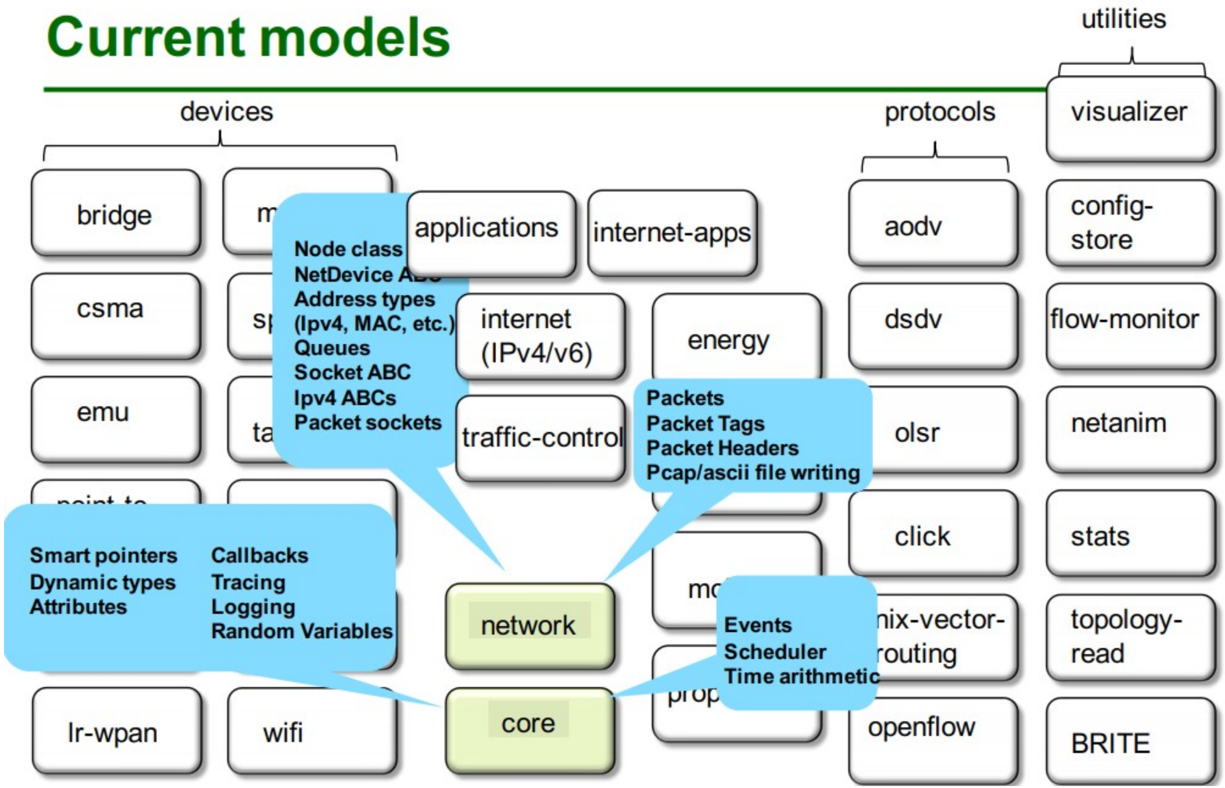


```
#include <iostream>
#include "ns3/command-line.h"

using namespace ns3;

int main (int argc, char *argv[])
{
  std::string language = "English";
  std::string phrase;

  CommandLine cmd;
  cmd.AddValue ("language", "Specify language", language);
  cmd.Parse (argc, argv);

  if (language == "English")
    {
      phrase = "Hello world";
    }
  else if (language == "Italian")
    {
      phrase = "Ciao mundo";
    }
  else
    {
      phrase = "That language is not spoken here";
    }

  std::cout << phrase << std::endl;
}
```

NS-3 Modules

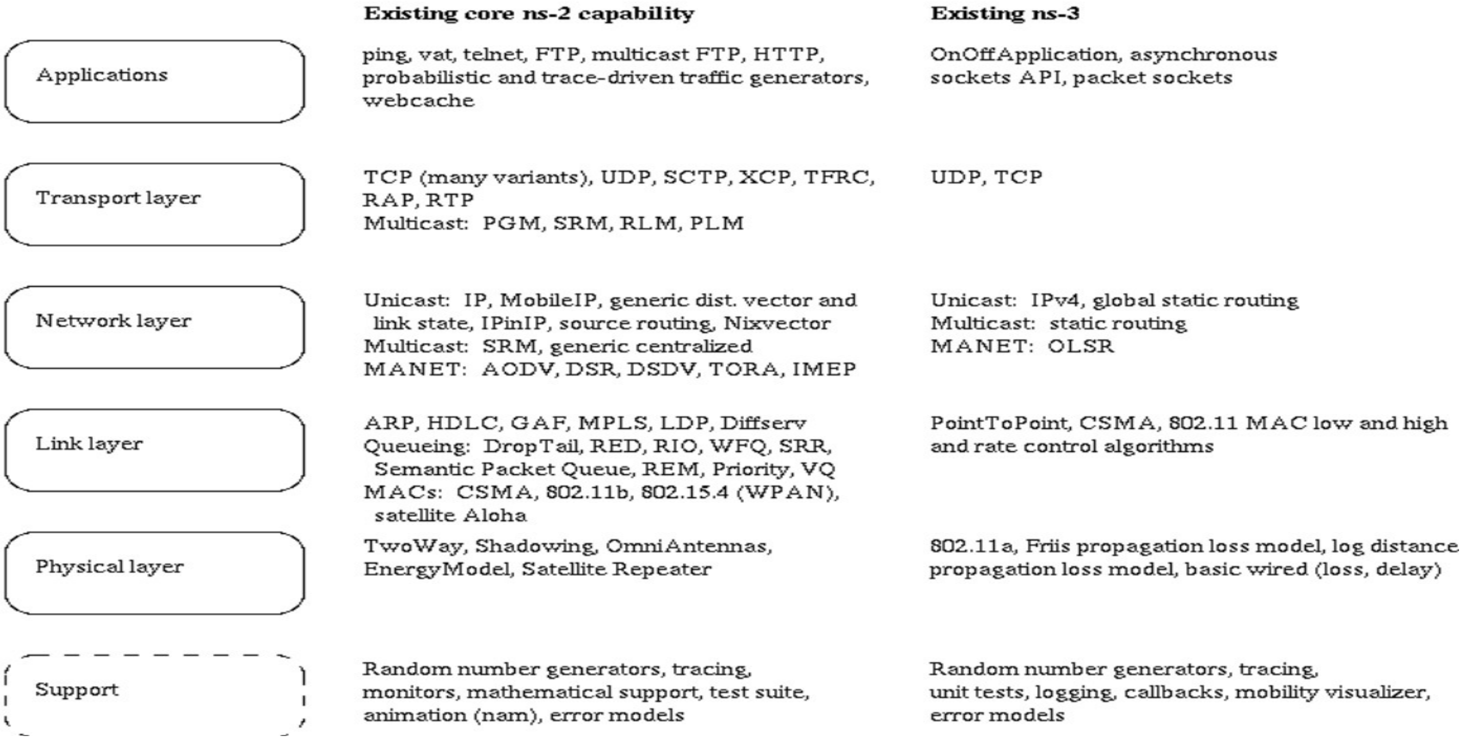
- A set of related classes, examples, and tests, that can be combined together into an ns-3 module so that they can be used with existing ns-3 modules and by other researchers
- All modules are under src directory
- Wired Network Modules
  - Point-to-Point, csma, Network, Internet, etc
- Wireless Network Modules
  - wifi, wimax, lte, olsr, aodv, dsdv, dsr, mesh, sixlowpan, spectrum, propogation, mobility, etc
- Generic modules
  - Core, energy, flow-monitor, stats, etc

NS-3 Modules



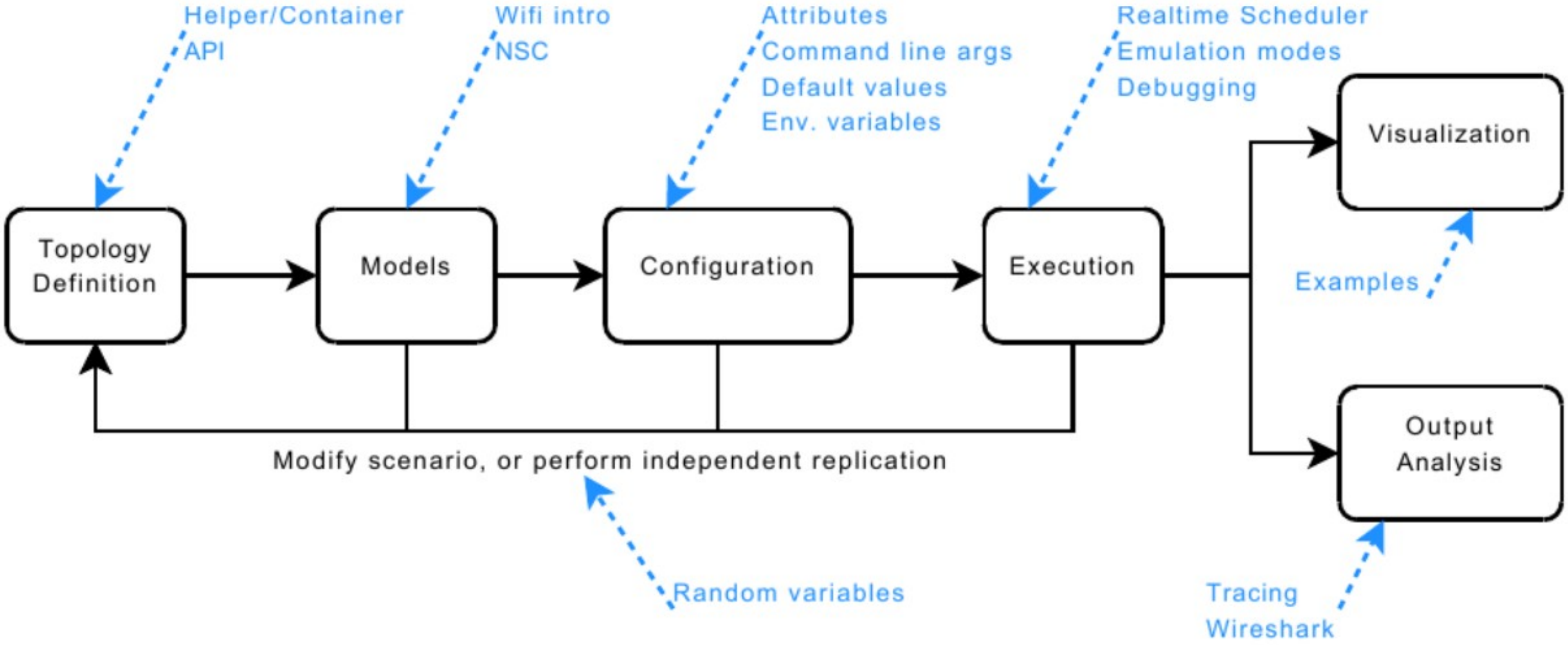
Current models
utilities
devices
protocols
visualizer
bridge
m
applications
internet-apps
aodv
config-store
csma
sp
internet (IPv4/v6)
energy
dsdv
flow-monitor
emu
ta
traffic-control
olsr
netanim
point-to
click
stats
network
mo
mix-vector-routing
topology-read
lr-wpan
wifi
core
prop
openflow
BRITE
Node class
NetDevice ABC
Address types (Ipv4, MAC, etc.)
Queues
Socket ABC
Ipv4 ABCs
Packet sockets
Packets
Packet Tags
Packet Headers
Pcap/ascii file writing
Smart pointers
Dynamic types
Attributes
Callbacks
Tracing
Logging
Random Variables
Events
Scheduler
Time arithmetic

NS-3 Models



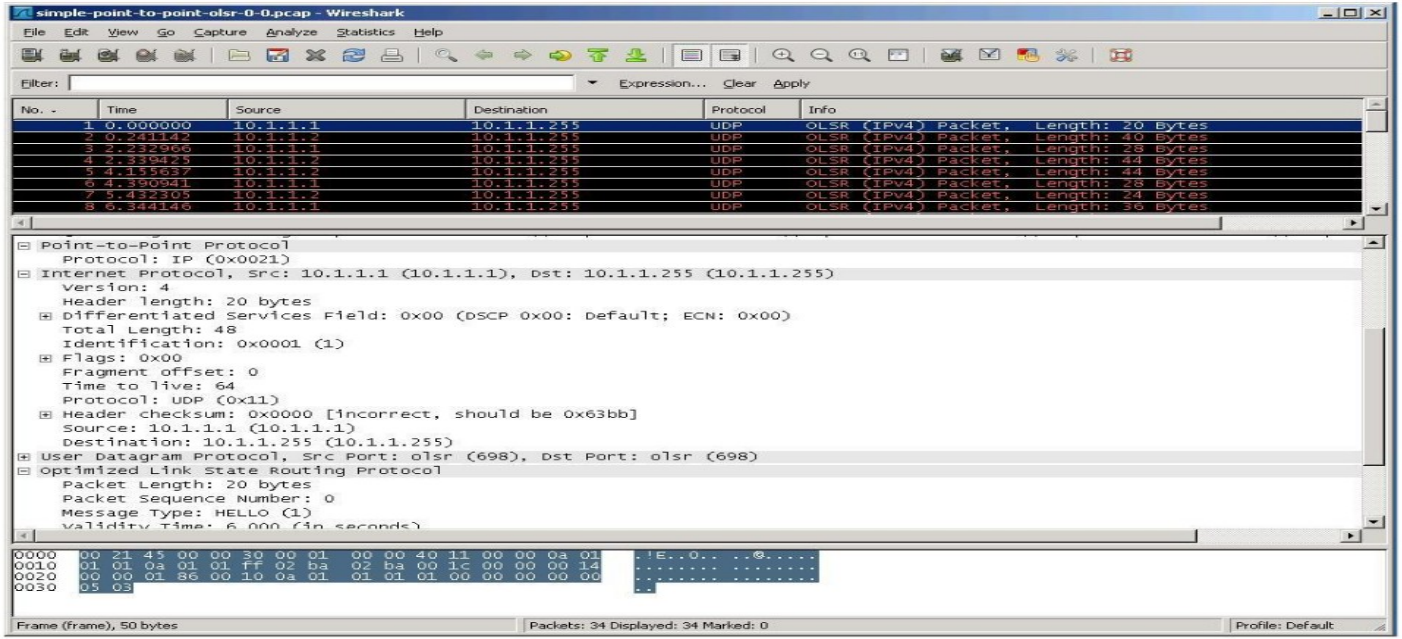
	Existing core ns-2 capability	Existing ns-3
Applications	ping, vat, telnet, FTP, multicast FTP, HTTP, probabilistic and trace-driven traffic generators, webcache	OnOffApplication, asynchronous sockets API, packet sockets
Transport layer	TCP (many variants), UDP, SCTP, XCP, TFRC, RAP, RTP Multicast: PGM, SRM, RLM, PLM	UDP, TCP
Network layer	Unicast: IP, MobileIP, generic dist. vector and link state, IPinIP, source routing, Nixvector Multicast: SRM, generic centralized MANET: AODV, DSR, DSDV, TORA, IMEP	Unicast: IPv4, global static routing Multicast: static routing MANET: OLSR
Link layer	ARP, HDLC, GAF, MPLS, LDP, Diffserv Queueing: DropTail, RED, RIO, WFQ, SRR, Semantic Packet Queue, REM, Priority, VQ MACs: CSMA, 802.11b, 802.15.4 (WPAN), satellite Aloha	PointToPoint, CSMA, 802.11 MAC low and high and rate control algorithms
Physical layer	TwoWay, Shadowing, OmniAntennas, EnergyModel, Satellite Repeater	802.11a, Friis propagation loss model, log distance propagation loss model, basic wired (loss, delay)
Support	Random number generators, tracing, monitors, mathematical support, test suite, animation (nam), error models	Random number generators, tracing, unit tests, logging, callbacks, mobility visualizer, error models

A typical NS-3 simulation structure



Software Integration

- NS-3 trace viewed with wireshark



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.1.1.1	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 20 Bytes
2	0.241142	10.1.1.2	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 40 Bytes
3	2.232966	10.1.1.1	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 28 Bytes
4	2.339425	10.1.1.2	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 44 Bytes
5	4.155637	10.1.1.2	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 44 Bytes
6	4.390941	10.1.1.1	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 28 Bytes
7	5.432305	10.1.1.2	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 24 Bytes
8	6.344146	10.1.1.1	10.1.1.255	UDP	OLSR (IPv4) Packet, Length: 36 Bytes

```
⊟ Point-to-Point Protocol
    Protocol: IP (0x0021)
⊟ Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.1.255 (10.1.1.255)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 48
    Identification: 0x0001 (1)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (0x11)
  ⊞ Header checksum: 0x0000 [incorrect, should be 0x63bb]
    Source: 10.1.1.1 (10.1.1.1)
    Destination: 10.1.1.255 (10.1.1.255)
⊞ User Datagram Protocol, Src Port: olsr (698), Dst Port: olsr (698)
⊟ Optimized Link State Routing Protocol
    Packet Length: 20 bytes
    Packet Sequence Number: 0
    Message Type: HELLO (1)
    Validity Time: 6.000 (in seconds)
```

```
0000  00 21 45 00 00 30 00 01  00 00 40 11 00 00 0a 01   .!E..0.. ..@.....
0010  01 01 0a 01 01 ff 02 ba  02 ba 00 1c 00 00 00 14   ........ ........
0020  00 00 01 86 00 10 0a 01  01 01 01 00 00 00 00 00   ........ ........
0030  05 03                                              ..
```

Frame (frame), 50 bytes | Packets: 34 Displayed: 34 Marked: 0 | Profile: Default

NS-3 Simulation Basis

- Simulation time moves in discrete jumps from event to event
- C++ functions schedule events to occur at specific simulation times
- A simulation scheduler orders the event execution
- Simulation::Run() gets it all started
- Simulation stops at a specific time or when all pending events end

NS-3 Event Scheduling Example

Location: ~/ns-allinone-3.24/ns-3.24/src/core/examples/sample-simulator.cc

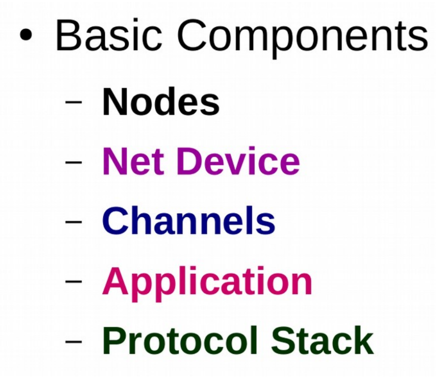
http://www.nsnam.org/doxygen/sample-simulator_8cc_source.html

Run

- ./waf --run scratch/sample-simulator
- ./waf --run ‘scratch/sample-simulator –RngRun=2’
- For latest ns3 release
- ./ns3 scratch/sample-simulator

Abstractions

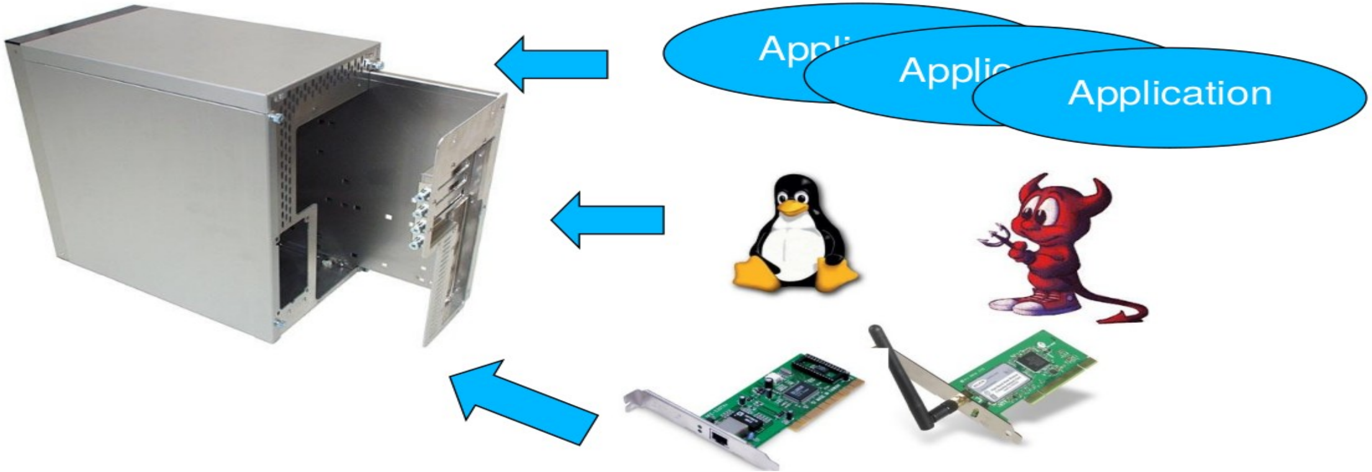
- Node
- Application
- Channel
- NetDevice
- Packet
- Topology Helpers – aggregate functionality of modules to make common operations easier than using the low-level API



- Basic Components
  - **Nodes**
  - **Net Device**
  - **Channels**
  - **Application**
  - **Protocol Stack**

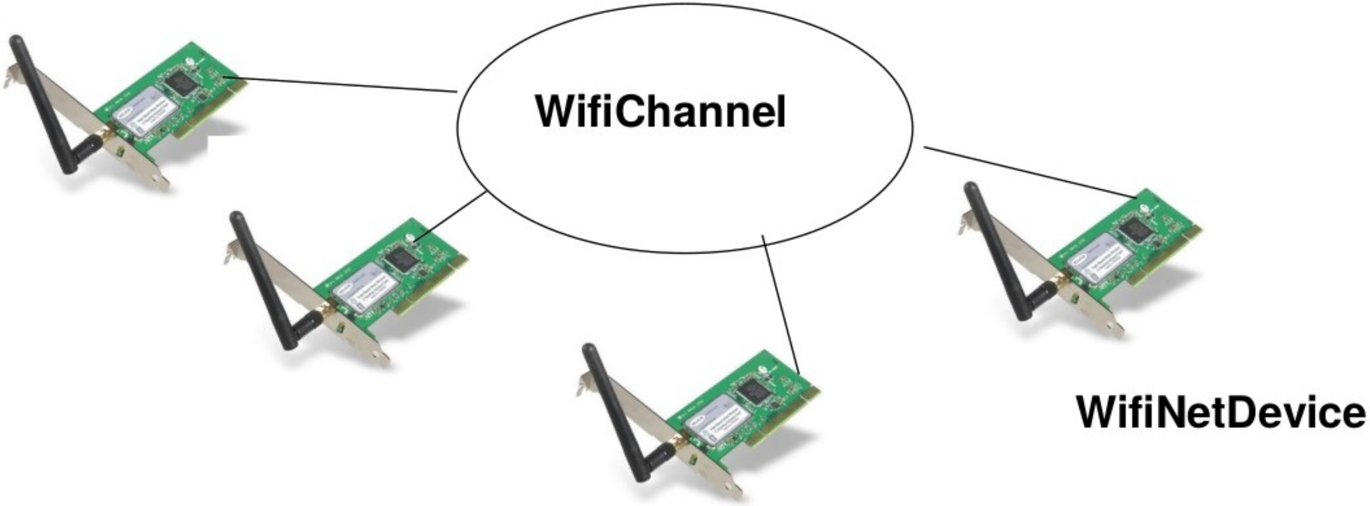
Fundamentals

- Key objects in the simulator are Nodes, Packets, and Channels
- Node is a husk of a computer to which applications, stacks, and NICs (NetDevs) are added
- Nodes are architected for multiple interfaces

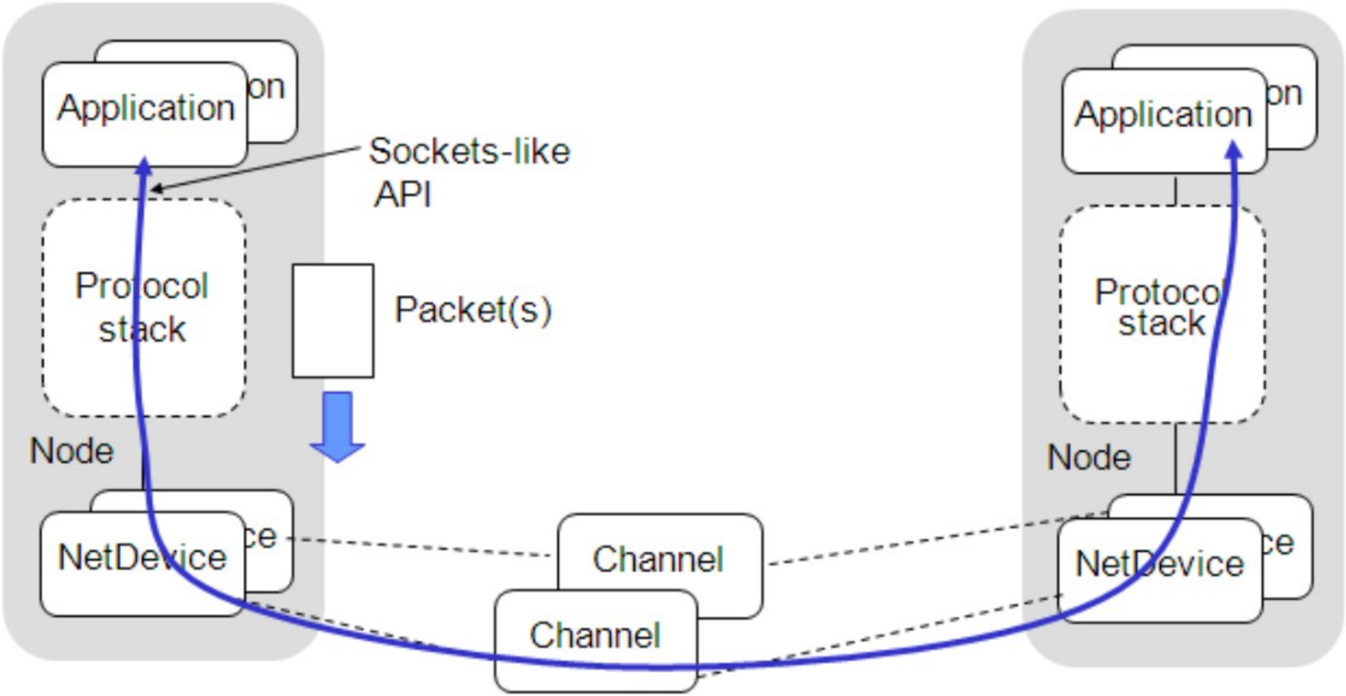


NetDevices and Channels

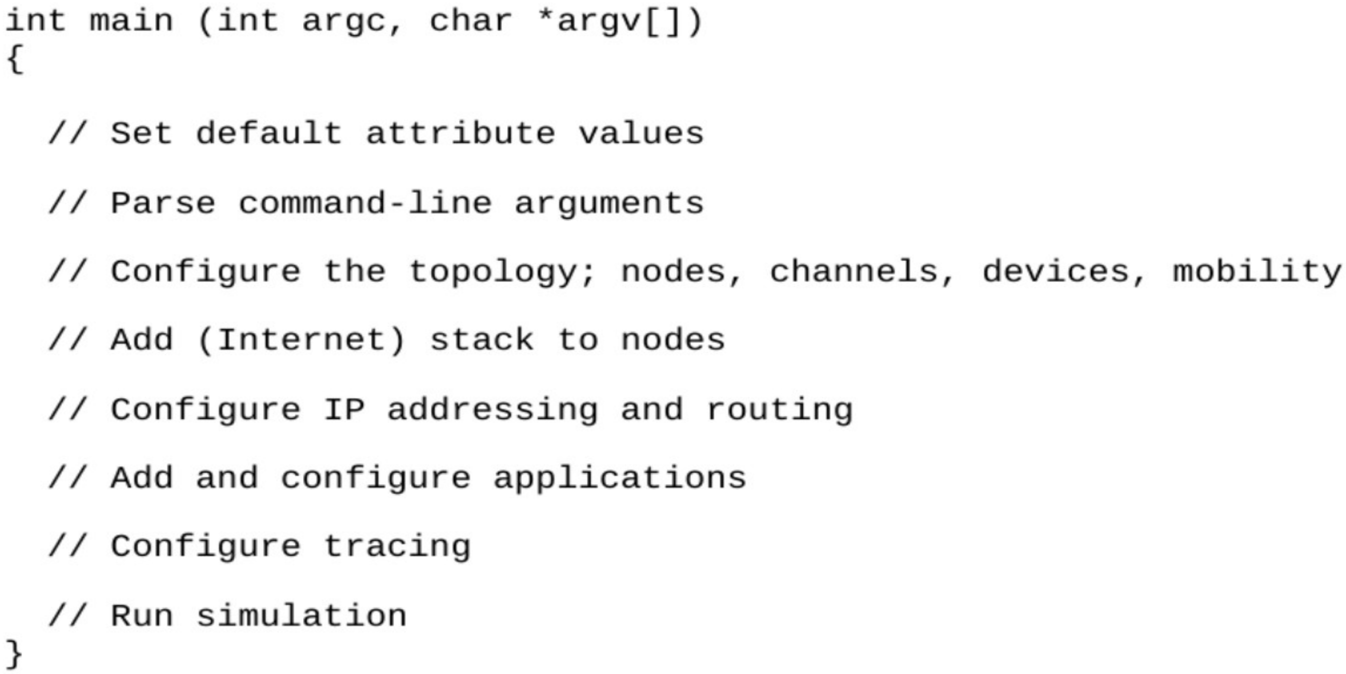
NetDevices (NICs) are strongly bound to channels of a matching type



Basic Model:

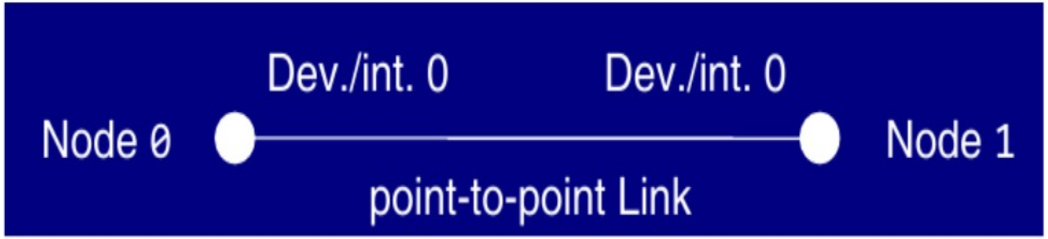

Application
on
Sockets-like API
Protocol stack
Packet(s)
Node
NetDevice
ce
Channel
Channel
Application
on
Protocol stack
Node
NetDevice
ce

Structure of NS-3 Program



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

  // Set default attribute values

  // Parse command-line arguments

  // Configure the topology; nodes, channels, devices, mobility

  // Add (Internet) stack to nodes

  // Configure IP addressing and routing

  // Add and configure applications

  // Configure tracing

  // Run simulation
}
```

Network Simulation Example:

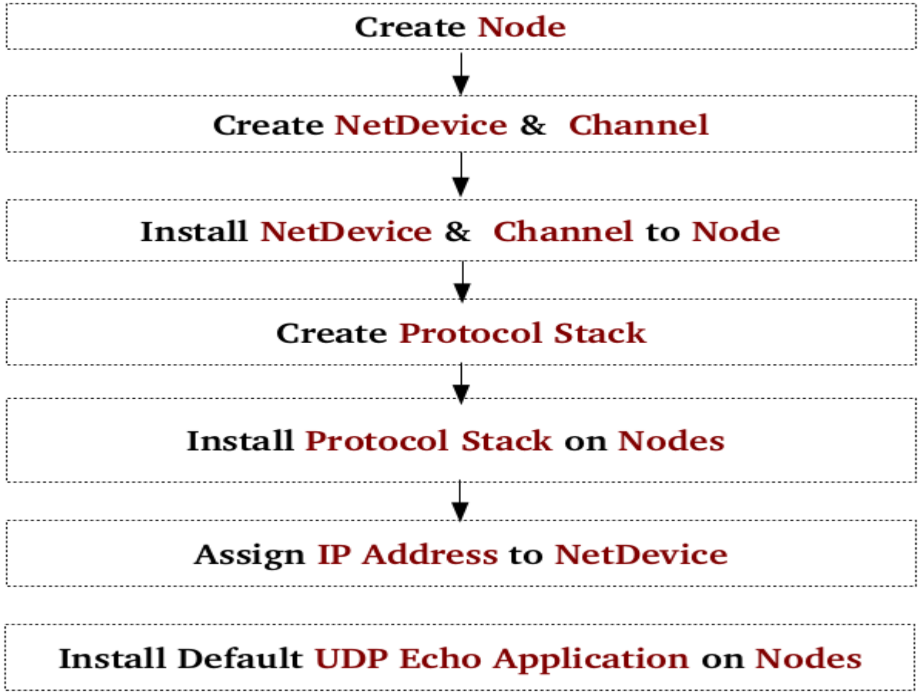


- Two nodes, one network interface device per node
- Point-to-point link

  - propagation delay: 2ms, data rate:5 Mbps
- Application

- Udp echo client on node 0, Udp echo server on node 1 on port 9
- Payload size of 1024-byte packet
- time interval between packet is 1 s

Flow Chart



Create Node

Create NetDevice & Channel

Install NetDevice & Channel to Node

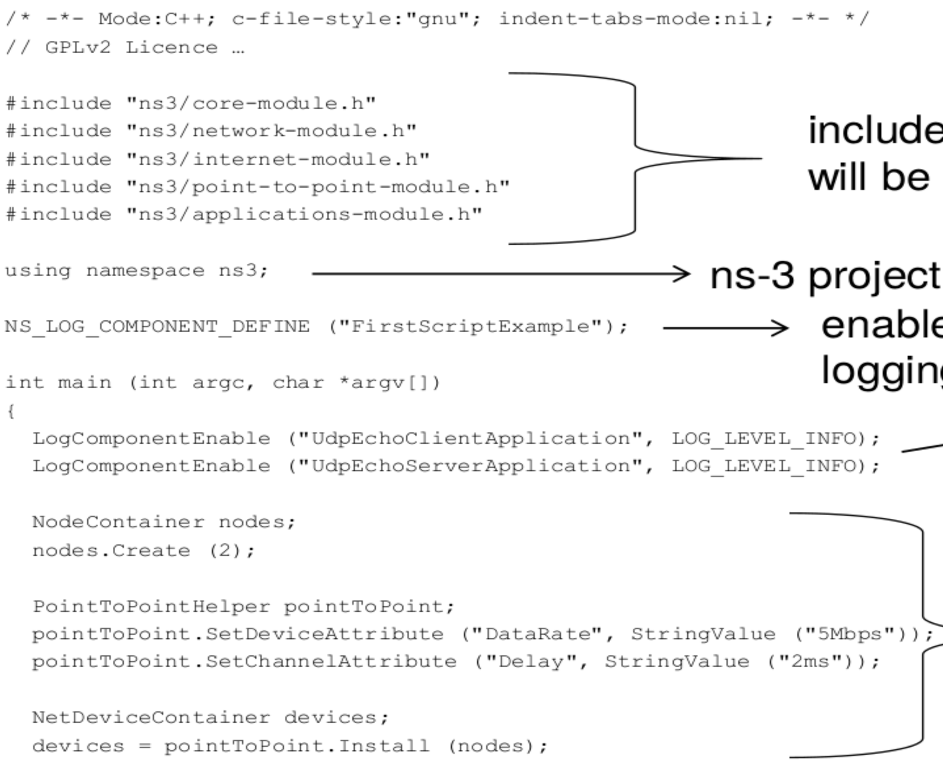
Create Protocol Stack

Install Protocol Stack on Nodes

Assign IP Address to NetDevice

Install Default UDP Echo Application on Nodes

Example Script - 1



```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
// GPLv2 Licence …

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

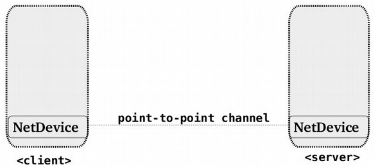
include modules that will be used

ns-3 project namespace

enable and disable console message logging by reference to the name

Create Node

Topology Configuration

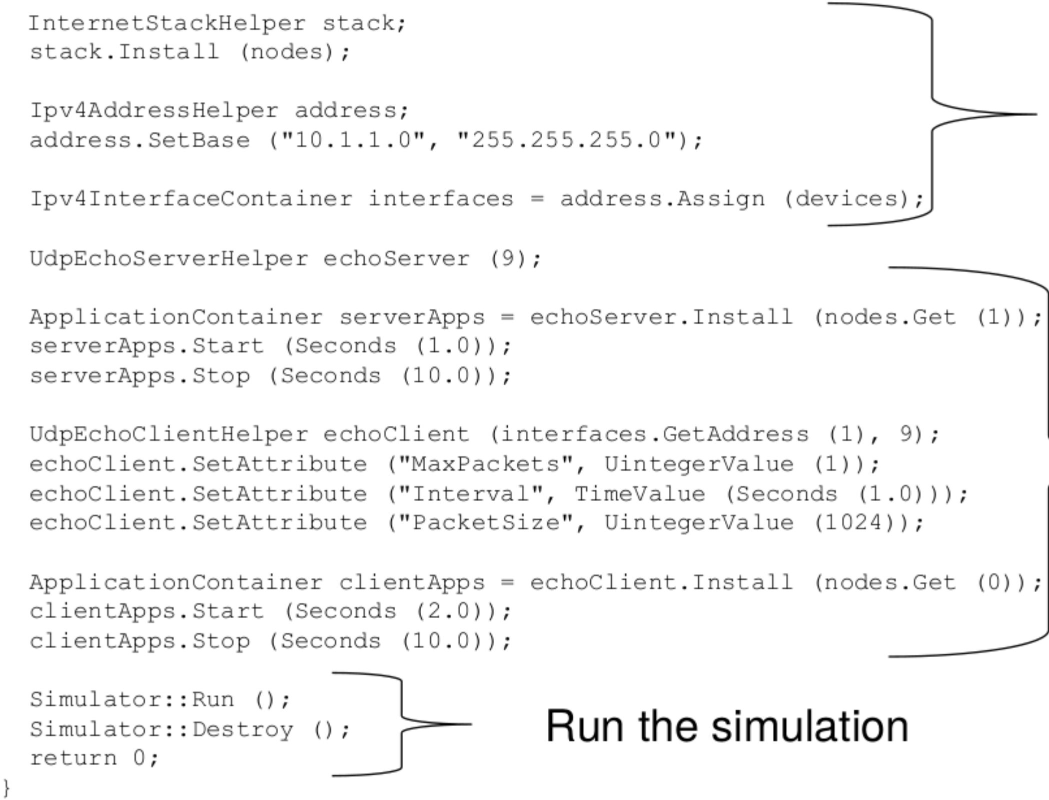


Example Script - 2

install Protocol Stack

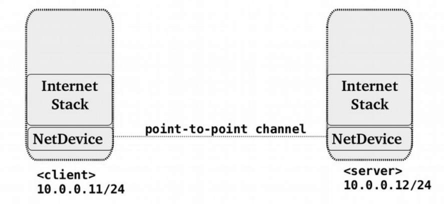
Set up Server

Set up Client

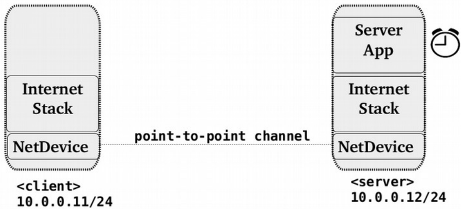


```
InternetStackHelper stack;
stack.Install (nodes);

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

Ipv4InterfaceContainer interfaces = address.Assign (devices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

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

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

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

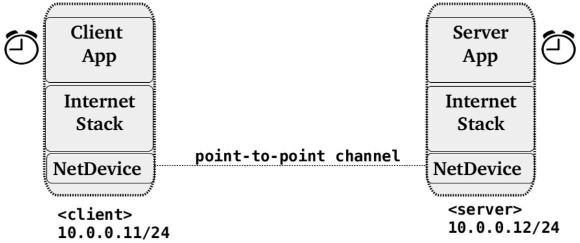
Set up internet stack



Set up applications

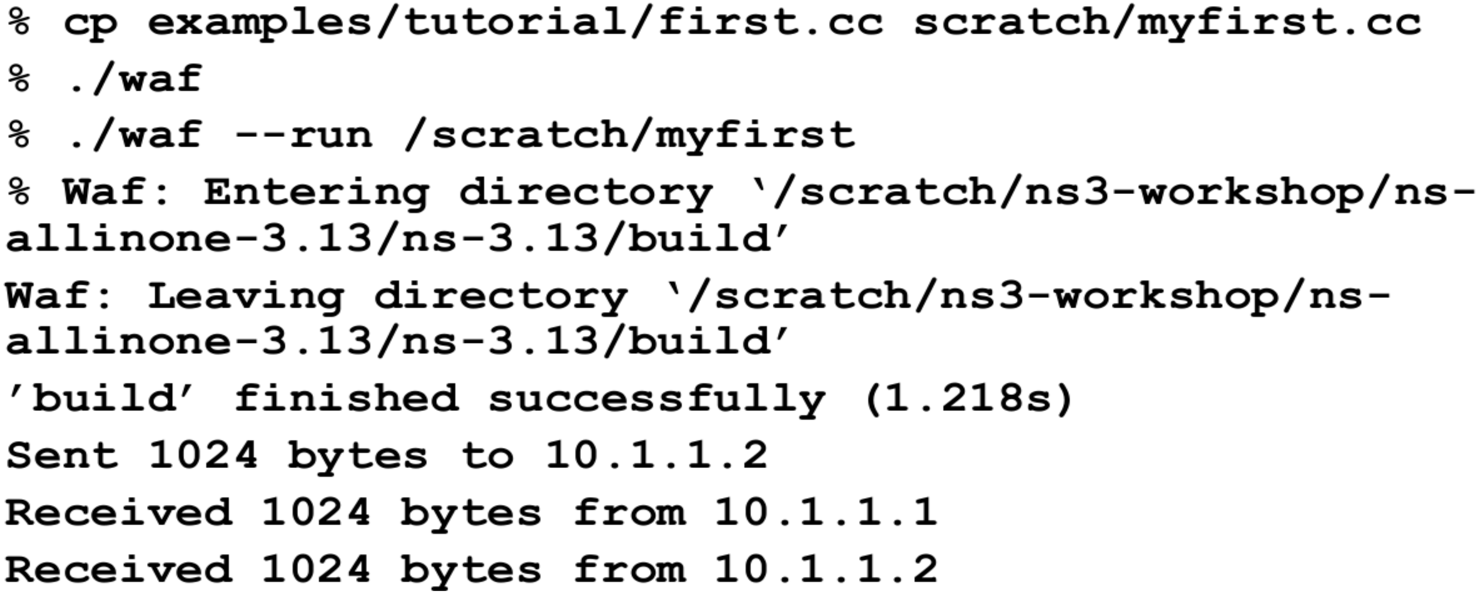


Run the simulation



Running Example

ALL SCENARIOS SHOULD BE RUN UNDER SCRATCH



```
% cp examples/tutorial/first.cc scratch/myfirst.cc
% ./waf
% ./waf --run /scratch/myfirst
% Waf: Entering directory `/scratch/ns3-workshop/ns-
allinone-3.13/ns-3.13/build'
Waf: Leaving directory `/scratch/ns3-workshop/ns-
allinone-3.13/ns-3.13/build'
'build' finished successfully (1.218s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Cmd Args

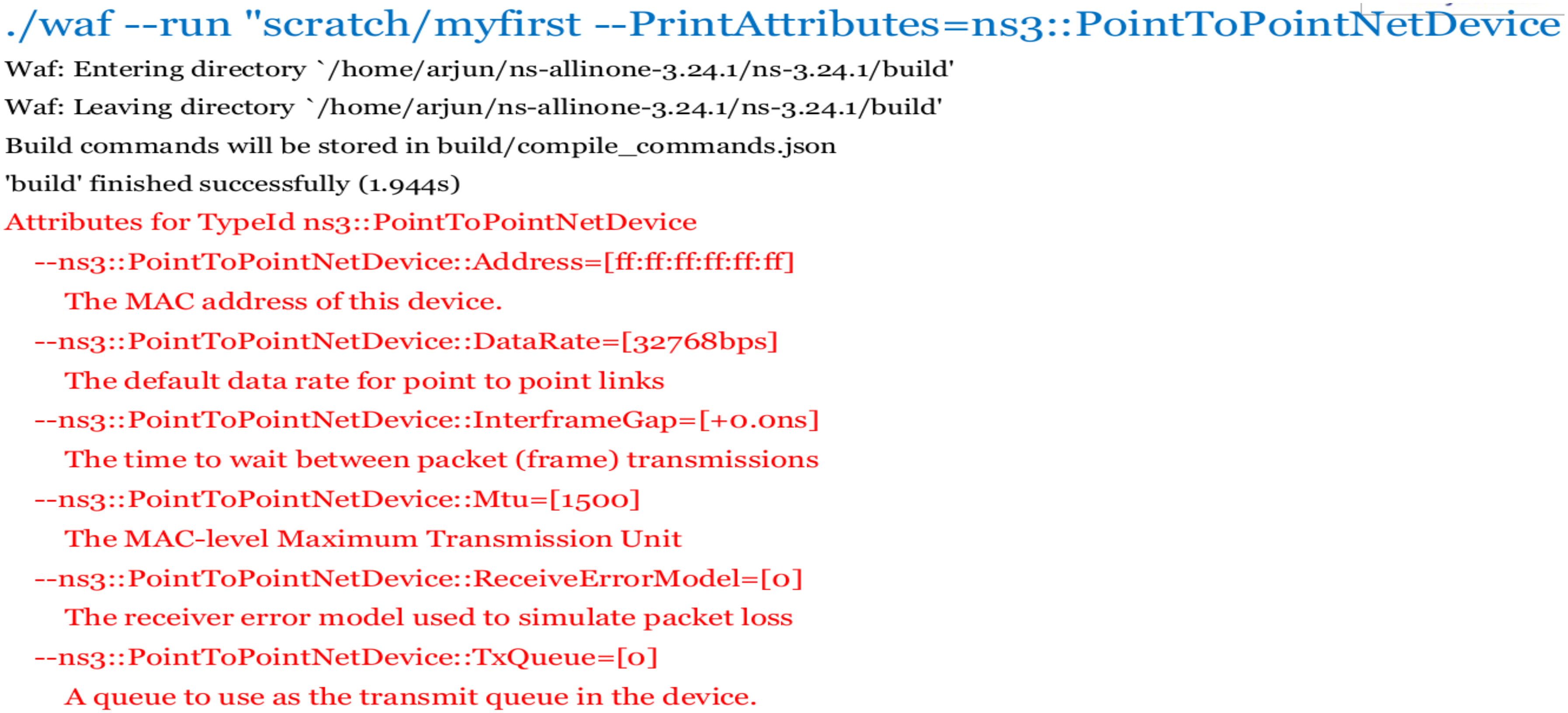
- ./waf --run "first --PrintHelp“
  - ./waf --run "scratch/myfirst --nPackets=2“

```
CommandLine cmd;
cmd.AddValue("nPackets", "Number of packets to send by echoClient App", nPackets);
cmd.Parse (argc, argv);
```

- ./waf --run scratch/myfirst --command-template="%s --help“
  - ./waf --run scratch/myfirst --command-template="%s --nPackets=2“

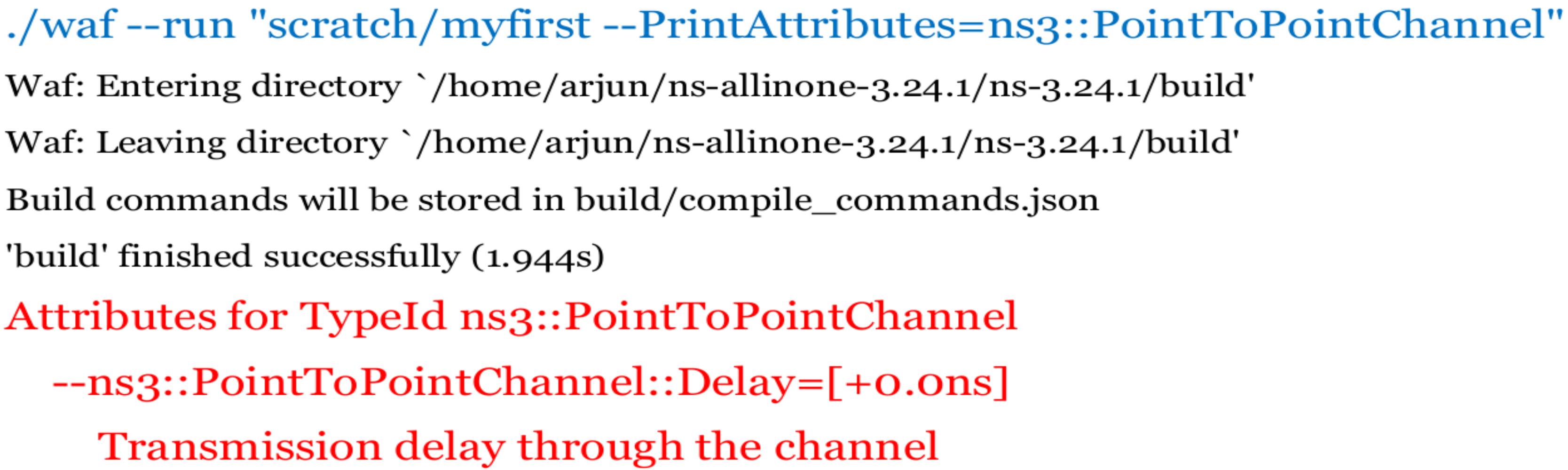
http://www.nsnam.org/docs/release/3.24/tutorial/singlehtml/index.html#using-command-line-arguments

myfirst.cc



```
./waf --run "scratch/myfirst --PrintAttributes=ns3::PointToPointNetDevice
Waf: Entering directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Waf: Leaving directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.944s)
Attributes for TypeId ns3::PointToPointNetDevice
    --ns3::PointToPointNetDevice::Address=[ff:ff:ff:ff:ff:ff]
        The MAC address of this device.
    --ns3::PointToPointNetDevice::DataRate=[32768bps]
        The default data rate for point to point links
    --ns3::PointToPointNetDevice::InterframeGap=[+0.0ns]
        The time to wait between packet (frame) transmissions
    --ns3::PointToPointNetDevice::Mtu=[1500]
        The MAC-level Maximum Transmission Unit
    --ns3::PointToPointNetDevice::ReceiveErrorModel=[0]
        The receiver error model used to simulate packet loss
    --ns3::PointToPointNetDevice::TxQueue=[0]
        A queue to use as the transmit queue in the device.
```

myfirst.cc

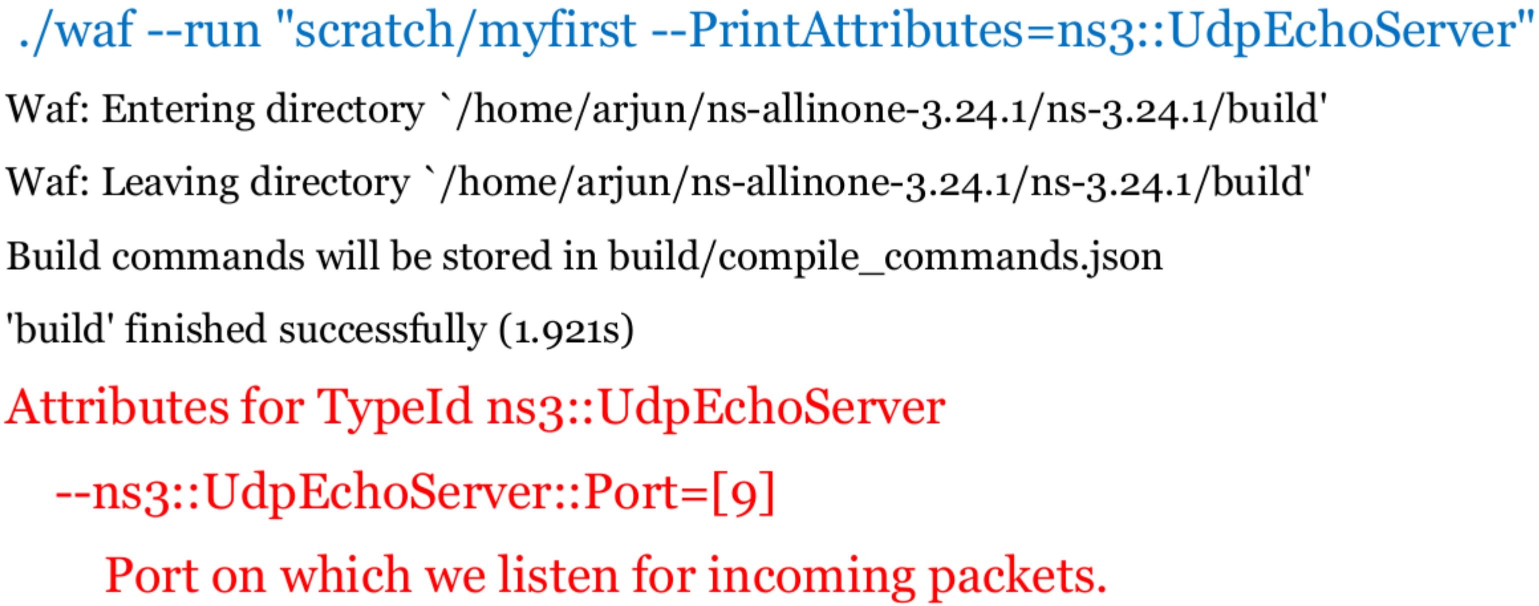


```
./waf --run "scratch/myfirst --PrintAttributes=ns3::PointToPointChannel"
Waf: Entering directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Waf: Leaving directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.944s)
Attributes for TypeId ns3::PointToPointChannel
    --ns3::PointToPointChannel::Delay=[+0.0ns]
        Transmission delay through the channel
```

myfirst.cc

```
./waf --run "scratch/myfirst --PrintAttributes=ns3::UdpEchoClient"
Waf: Entering directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Waf: Leaving directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.985s)
Attributes for TypeId ns3::UdpEchoClient
    --ns3::UdpEchoClient::Interval=[+1000000000.0ns]
        The time to wait between packets
    --ns3::UdpEchoClient::MaxPackets=[100]
        The maximum number of packets the application will send
    --ns3::UdpEchoClient::PacketSize=[100]
        Size of echo data in outbound packets
    --ns3::UdpEchoClient::RemoteAddress=[00-00-00]
        The destination Address of the outbound packets
    --ns3::UdpEchoClient::RemotePort=[0]
        The destination port of the outbound packets
```

myfirst.cc



```
./waf --run "scratch/myfirst --PrintAttributes=ns3::UdpEchoServer"
Waf: Entering directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Waf: Leaving directory `/home/arjun/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.921s)
Attributes for TypeId ns3::UdpEchoServer
    --ns3::UdpEchoServer::Port=[9]
        Port on which we listen for incoming packets.
```

myfirst.cc

Passing values to user defined variables and Attributes (by overriding their default values) from command line:

```
./waf --run "scratch/myfirst --nPackets=2 --ns3::UdpEchoClient::PacketSize=5000“
```

Note that if the programmer sets any new value to PacketSize in the program, that overrides the above command line arg value of 5000 Bytes. Same thing even with nPackets user-defined variable.

Logs

- Logs are generally used to get useful information.
- NS-3 provides different levels of logs which are as follows:-

– LOG_INFO

– LOG_FUNCTION

– LOG_LOGIC

– LOG_ALL

– etc.

Enable Log

- To enable log :
  - LogComponentEnable(module_name,log_level)

- Example
  - Module_name
    - ns3::UdpEchoClientApplication
    - ns3::UdpEchoServerApplication
    - And,more

  - Log_level
    - LOG_LEVEL_INFO
    - LOG_LEVEL_FUNCTION
    - LOG_LEVEL_ALL
    - and ,more

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

Acknowledgements: Mathieu Lacage, Tom Henderson, www.nsnam.org

Training: https://www.nsnam.org/consortium/activities/training/