

**LAB MANUAL OF
COMPUTER NETWORKS /
DATA COMMUNICATION AND NETWORKS LAB
ETCS 354/ ETEC 358**



Maharaja Agrasen Institute of Technology, PSP area,
Sector – 22, Rohini, New Delhi – 110085
(Affiliated to Guru Gobind Singh Indraprastha University,
Dwarka, New Delhi)

INDEX OF THE CONTENTS

- 1. Introduction to the lab manual**
- 2. Lab requirements (details of H/W & S/W to be used)**
- 3. List of experiments**
- 4. List of Advance programs**
- 5. Projects to be allotted**
- 6. Format of lab record to be prepared by the students.**
- 7. Marking scheme for the practical exam**
- 8. Details of the each section of the lab along with the examples, exercises & expected viva questions.**

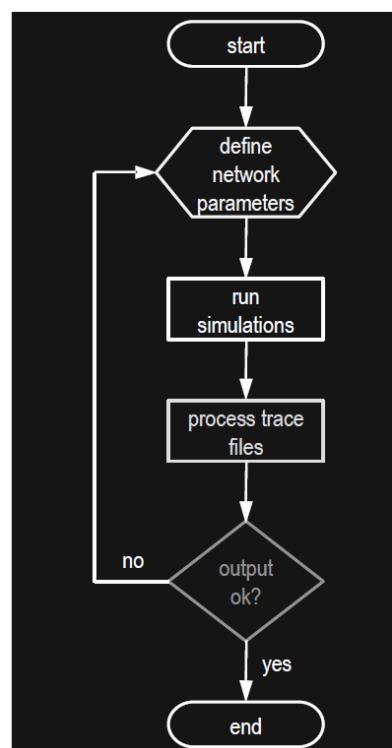
Introduction to computer networks / data communication and networks lab

This lab gives in depth view of how computer networks works in real time, simulation of various topologies are performed using ns3 tool.

ns-3 has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, *ns-3* provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use *ns-3* include to perform studies that are more difficult or not possible to perform with real systems, to study system behavior in a highly controlled, reproducible environment, and to learn about how networks work. Users will note that the available model set in *ns-3* focuses on modeling how Internet protocols and networks work, but *ns-3* is not limited to Internet systems; several users are using *ns-3* to model non-Internet-based systems.

Many simulation tools exist for network simulation studies. Below are a few distinguishing features of *ns-3* in contrast to other tools.

- *ns-3* is designed as a set of libraries that can be combined together and also with other external software libraries. While some simulation platforms provide users with a single, integrated graphical user interface environment in which all tasks are carried out, *ns-3* is more modular in this regard. Several external animators and data analysis and visualization tools can be used with *ns-3*. However, users should expect to work at the command line and with C++ and/or Python software development tools.
- *ns-3* is primarily used on Linux systems, although support exists for FreeBSD, Cygwin (for Windows), and native Windows Visual Studio support is in the process of being developed.



2. LAB REQUIREMENTS

H/W Detail	24 Nos.
Intel i3/C2D Processor/2 GB RAM/500GB HDD/MB/Lan Card/	
Key Board/ Mouse/CD Drive/15" Color Monitor/ UPS	
LaserJet Printer	1 No.
S/W Detail	Ubuntu, Fedora Linux, NS3

List of experiments

1. Introduction about discrete events simulation and its tools
2. Installation of NS3 in linux
3. Program in NS3 to connect two nodes
4. Program in NS3 for connecting three nodes considering one node as a central node.
5. Program in NS3 to implement star topology
6. Program in NS3 to implement a bus topology.
7. Program in NS3 for connecting multiple routers and nodes and building a hybrid topology.
8. Installation and configuration of NetAnim
9. Program in NS3 to implement FTP using TCP bulk transfer.
10. Program in NS3 for connecting multiple routers and nodes and building a hybrid topology and then calculating network performance
11. To analyse network traces using wireshark software.

4. FORMAT OF THE LAB RECORD TO BE PREPARED BY THE STUDENTS

The front page of the lab record prepared by the students should have a cover page as displayed below.

NAME OF THE LAB

Font should be (Size 20", italics bold, Times New Roman)

Faculty name

Student name

Roll No.:

Semester:

Group:

Font should be (12", Times Roman)



Maharaja Agrasen Institute of Technology, PSP Area,
Sector – 22, Rohini, New Delhi – 110085

Font should be (18", Times Roman)

The second page in the record should be the index as displayed below.

LAB NAME
PRACTICAL RECORD

PAPER CODE :

Name of the student :

University Roll No. :

Branch :

Section/ Group :

PRACTICAL DETAILS

Experiments according to the lab syllabus prescribed by GGSIPU

Exp. no	Experiment Name	Date of performance	Date of checking	Remarks	Marks

5. MARKING SCHEME FOR THE PRACTICAL EXAMS

There will be two practical exams in each semester.

- Internal Practical Exam
- External Practical Exam

INTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 40

Division of 40 marks is as follows

1. Regularity: 30

- Performing program in each turn of the lab
- Attendance of the lab
- File

2. Viva Voice: 10

NOTE: For the regularity, marks are awarded to the student out of 5 for each experiment performed in the lab and at the end the average marks are giving out of 30.

EXTERNAL PRACTICAL EXAM

It is taken by the concerned faculty of the batch and by an external examiner. In this examination student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 60

Division of 60 marks is as follows

- | | |
|---------------------------------|----|
| 1. Sheet filled by the student: | 20 |
| 2. Viva Voice: | 15 |
| 3. Experiment performance: | 15 |
| 4. File submitted: | 10 |

NOTE:

- Internal marks + External marks = Total marks given to the students
(40 marks) (60 marks) (100 marks)
- Experiments given to perform can be from any section of the lab.

6. DETAILS OF EACH SECTION

ALONG WITH

EXAMPLES, EXERCISES

&

EXPECTED VIVA QUESTIONS

Chapter 1: Introduction to Discrete-Event Simulation

System:

A collection of entities that act and interact together toward the accomplishment of some logical end.

Discrete system:

State variables change instantaneously at separated point in time, e.g., a bank, since state variables - number of customers, change only when a customer arrives or when a customer finishes being served and departs

Continuous system:

State variable change continuously with respect to time, e.g., airplane moving through the air, since state variables - position and velocity change continuously with respect to time

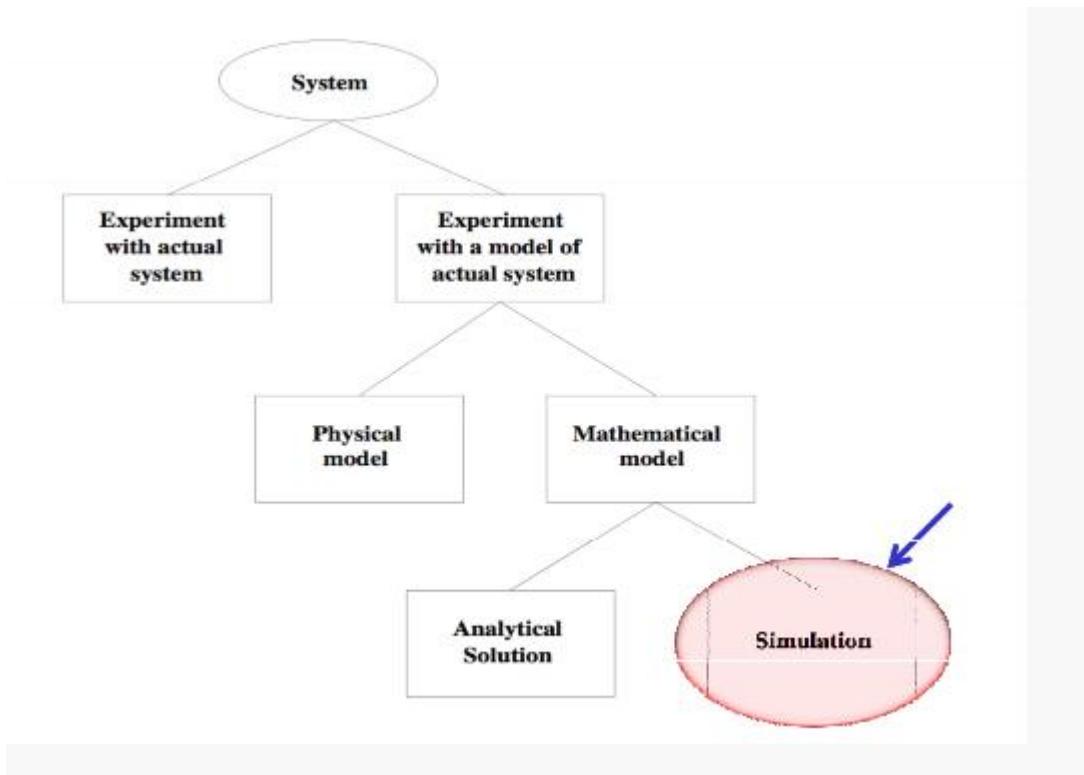


Fig 1: Ways to study the system

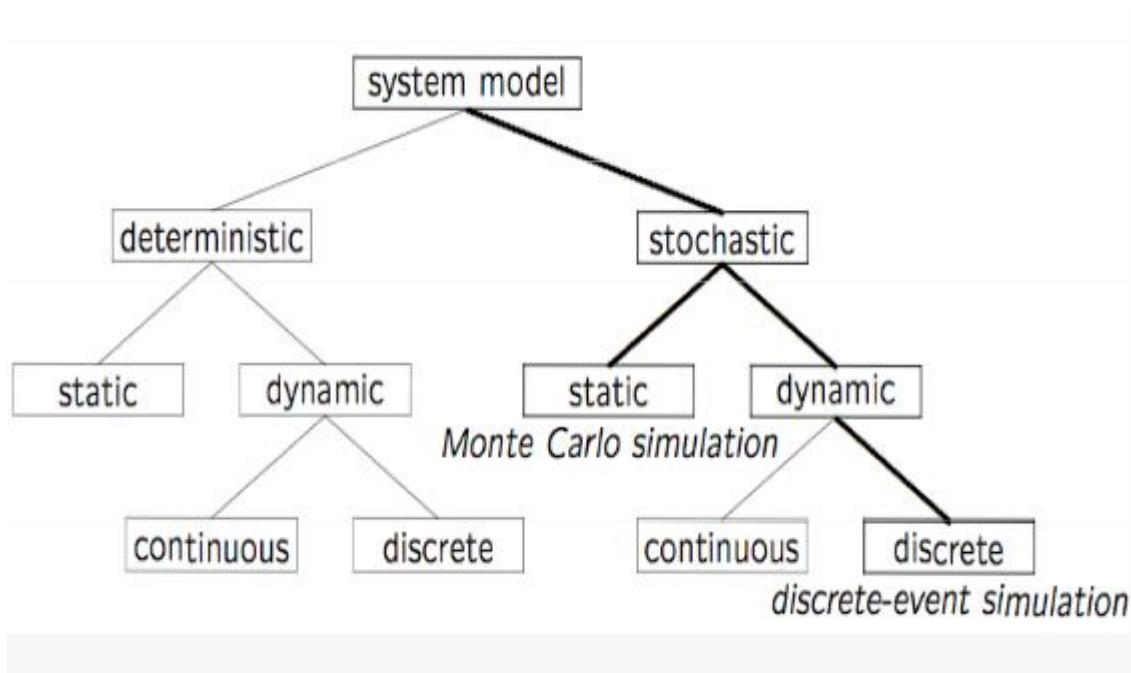


Fig 2: Model Taxonomy

Why Simulation?

- Many systems are highly complex, precluding the possibility of analytical solution
- The analytical solutions are extraordinarily complex, requiring vast computing resources
- Thus, such systems should be studied by means of simulation

numerically exercising the model for inputs in question to see how they affect the output measures of performance

“Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of a system.”

What is Discrete-Event Simulation (DES)

A discrete-event simulation

models a system whose state may change only at discrete points in time

System:

is composed of objects called entities that have certain properties called attributes.

State:

a collection of attributes or state variables that represent the entities of the system.

Event:

an instantaneous occurrence in time that may alter the state of the system

- **Discrete-event simulation is stochastic, dynamic, and discrete**
- **Stochastic = Probabilistic**
 - Inter-arrival times and service times are random variables
 - Have cumulative distribution functions
- **Discrete = Instantaneous events are separated by intervals of time**
 - The state variables change instantaneously at separate points in time
 - The system can change at only a countable number of points in time.
 - These points in time are the ones at which an event occurs.
- **Dynamic = Changes over time**
 - Simulation clock
 - Keep track of the current value of simulated time as the simulation proceeds
 - A mechanism to advance simulated time from one value to another
 - Next-event time advance

Fig 3: Introduction to Discrete Event Simulation

Chapter 2:Introduction to NS3

Introduction

In this lab, we will be using the Network Simulator, NS3, available from www.nsnam.org. NS3 is a powerful program, however we will only be looking at some basic features. NS3 simulations are built in C++.

NS2versus NS3

	NS2	NS3
Programminglanguages	<ul style="list-style-type: none">• NS2 is implemented using a combination of TCL (for scripts describing the network topology) and C++ (The core of the simulator).• This system was chosen in the early 1990s to avoid the recompilation of C++ as it was very time consuming using the hardware available at that time, oTCL recompilation takes less time than C++.• <u>TCL disadvantage:</u> there is overhead introduced with large simulations.• oTCL is the only available scripting language.	<ul style="list-style-type: none">• NS3 is implemented using C++• With modern hardware capabilities, compilation time was not an issue like for NS2, NS3 can be developed with C++ entirely.• A simulation script can be written as a C++ program, which is not possible in NS2.• There is limited support for Python in scripting and visualization.
MemoryManagement	<ul style="list-style-type: none">• NS2 requires basic manual C++ memory management functions.	<ul style="list-style-type: none">• Because NS3 is implemented in C++, all normal C++ memory management functions such as new, delete, malloc, and free are still available.• Automatic de-allocation of objects is supported using reference counting (track number of pointers to an object); this is useful when dealing with packet objects.

Packets	<ul style="list-style-type: none"> A packet consists of 2 distinct regions; one for headers, and the second stores payload data. NS2 never frees memory used to store packets until the simulation terminates, it just reuses the allocated packets repeatedly, as a result, the header region of any packet includes all headers defined as part of the used protocol even if that particular packet won't use that particular header, but just to be available when this packet allocation is reused. 	<ul style="list-style-type: none"> A packet consists of a single buffer of bytes, and optionally a collection of small tags containing meta-data. The buffer corresponds exactly to the stream of bits that would be sent over a real network. Information is added to the packet by using subclasses; Header, which adds information to the beginning of the buffer, Trailer, which adds to the end. Unlike NS2, there is generally an easy way to determine if a specific header is attached.
Performance	<ul style="list-style-type: none"> The total computation time required to run a simulation scales better in NS3 than NS2. This is due to the removal of the overhead associated with interfacing to Tcl with C++, and the overhead associated with the oTcl interpreter. 	<ul style="list-style-type: none"> NS3 performs better than NS2 in terms of memory management. The aggregation system prevents unneeded parameters from being stored, and packets don't contain unused reserved header space.
Simulation output	<ul style="list-style-type: none"> NS2 comes with a package called NAM (Network Animator), it's a Tcl based animation system that produces a visual representation of the network described. 	<ul style="list-style-type: none"> NS3 employs a package known as PyViz, which is a Python-based real-time visualization package.

Some Important Points about NS3:

1. NS3 is not backward compatible with NS2; it's built from scratch to replace NS2.
2. NS3 is written in C++, Python Programming Language can be optionally used as an interface.
3. NS3 is trying to solve problems present in NS2.
4. There is very limited number of contributed codes made with NS3 compared to NS2.
5. In NS2, bi-languagesystem made debugging complex (C++/Tcl), but for NS3 only knowledge of C++ is enough (single-language architecture is more robust in the long term).
6. NS3 has an emulation mode, which allows for the integration with real networks.

	Existing core ns-2 capability	ns-2 contributed code
Applications	ping, vat, telnet, FTP, multicast FTP, HTTP, probabilistic and trace-driven traffic generators, webcache	NSWEB, Video traffic generator, MPEG generator, BonnTraffic, ProtoLib, AgentJ, SIP, NSIS, ns2voip, AgentPlant
Transport layer	TCP (many variants), UDP, SCTP, XCP, TFRC, RAP, RTP Multicast: PGM, SRM, RLM, PLM	TCP PEP, SCPS-TP SNACK, TCP Pacing, DCCP, Simulation Cradle, TCP Westwood, SIMD, TCP-RH, MFTP, OTERS, TCP Eifel
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	AODV+, AODV-UU, AOMDV, ns-click, ZRP, IS-IS, CDS, Dynamic Linkstate, DYMO, OLSR, ATM, AntNet, Mobile IPv6, IP micro-mobility, MobileIP, GPSR, RSVP, PGM, PLM, SSM, PUMA, ActiveNetworks
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	802.16, 802.11e HCCA, 802.11e EDCA, 802.11a multirate, UWB DCC-MAC, TDMA DAMA, EURANE, UMTS, GPRS, BlueTooth, 3GPP 3GPP2 PCE, 802.11 PSM, MPLS, WFQ schedulers, Bandwidth Broker, CSFQ, BLUE
Physical layer	TwoWay, Shadowing, OmniAntennas, EnergyModel, Satellite Repeater	ET/SNRT/BER-based Phy, IR-UWB
Support	Random number generators, tracing, monitors, mathematical support, test suite, animation (nam), error models	Emulation, CANU mobility, BonnMotion mobility, SGB Topology Generators, NSG2, sind, ns2measure, ns-2/akaroa-2, yavista, tracegraph, huginn, multistate error model, RPI graphing package, jTrana, GEA,

From http://www-npa.lip6.fr/~rehmani/ns3_v1.pdf

Figure4: NS2contributedcode

	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

From http://www-npa.lip6.fr/~rehmani/ns3_v1.pdf

Figure3:NS2andNS3existingcorecapabilities(inJuly2010)

Chapter 3: Installation of NS3

Following are the basic steps which must be followed for installing NS3

1. Install prerequisite packages
2. Download ns3 codes
3. Build ns3
4. Validate ns3

Prerequisite packages for Linux are as follows:

1. Minimal requirements for Python: gcc g++ python
2. Debugging and GNU Scientific Library (GSL) support: gdbpython-dev
3. valgrind gsl-bin libgsl0-dev libgsl0ldbl Network Simulation Cradle (nsc): flex bison
Reading pcap packet traces: tcpdump
4. Database support for statistics framework: sqlite sqlite3
5. Xml-based version of the config store: libxml2
6. A GTK-based configuration system: libgtk2.0-0
7. Experimental with virtual machines and ns-3: vtun lxc

Detail steps are as follows:

1. sudo apt-get update / dnf update
2. sudo apt-get upgrade / dnf upgrade
- 3 Once ubuntu/fedora is installed run following command opening the terminal(ctrl+alt+T) window.
4. To install prerequisites dependancy packages- Type the following command in terminal window.

```
sudo apt-get/ dnf install gcc g++ python python-dev mercurial bzip2 gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex bison tcpdump sqlite sqlite3 libsqlite3-dev libxml2 libxml2-dev libgtk2.0-0 libgtk2.0-dev uncrustify doxygen graphviz imagemagick texlive texlive-latex-extra texlive-generic-extra texlive-generic-recommended texinfo dia texlive texlive-latex-extra texlive-extra-utils texlive-generic-recommended texi2html python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev python-pygccxml
```

5.After downloading NS3 on the drive, extract all the files in the NS3 folder, which you have created.

6.Then you can find build.py along with other files in NS3 folder.

Then to build the examples in ns-3 run :

```
./build.py --enable-examples --enable-tests
```

If the build is successful then it will give output

"Build finished successfully".

7. Now run the following command on the terminal window,to configure with waf(build tool)

```
./waf -d debug --enable-examples --enable-tests configure
```

To build with waf(optional)

```
./waf
```

8.To test everything allright run the following command on the terminal window,

```
./test.py
```

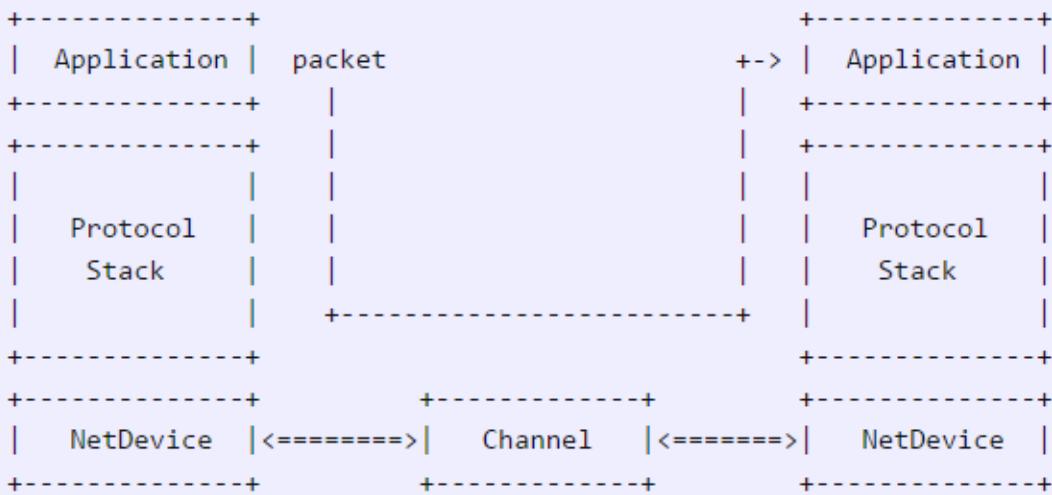
If the tests are ok the installation is done

9.Now after installing ns3 and testing it run some programs first to be ns3 user:

make sure you are in directory where waf script is available then run

EXPERIMENT No.3

Program in NS3 to connect two nodes



Node

Because in any network simulation, we will need nodes. So ns-3 comes with [NodeContainer](#) that you can use to manage all the nodes (Add, Create, Iterate, etc.).

```
// Create two nodes to hold.  
NodeContainer nodes;  
nodes.Create (2);
```

Channel and NetDevice

In the real world, they correspond to network cables (or wireless media) and peripheral cards (NIC). Typically these two things are intimately tied together. In the first example, we are using [PointToPointHelper](#) that wraps the Channel and NetDevice.

```
// Channel: PointToPoint, a direct link with `DataRate` and `Delay` specified.  
PointToPointHelper pointToPoint;  
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));  
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

Then we need to install the devices. The internal of `Install` is actually more complicated, but for now, let's just skip the magic behind the scene.

```
// NetDevice: installed onto the channel  
  
NetDeviceContainer devices;  
  
devices = pointToPoint.Install (nodes);
```

Protocols

Internet and IPv4. Since Internet is the current largest network to study, ns-3 has a particular focus on it. The `InternetStackHelper` will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.

```
// Protocol Stack: Internet Stack  
  
InternetStackHelper stack;  
  
stack.Install (nodes);
```

To assign IP addresses, use a helper and set the base. The low level ns-3 system actually remembers all of the IP addresses allocated and will generate a fatal error if you accidentally cause the same address to be generated twice.

```
// Since IP Address assignment is so common, the helper does the dirty work!  
  
// You only need to set the base.  
  
Ipv4AddressHelper address;  
  
address.SetBase ("10.1.1.0", "255.255.255.0");  
  
  
// Assign the address to devices we created above  
  
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

Applications

Every application needs to have `Start` and `Stop` function so that the simulator knows how to schedule it. Other functions are application-specific. We will use `UdpEchoServer` and `UdpEchoClient` for now.

```

// Application layer: UDP Echo Server and Client

// 1, Server:

UdpEchoServerHelper echoServer (9);

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

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));


// 2, Client:

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));

```

Simulation

```

// Start Simulation

Simulator::Run ();

Simulator::Destroy ();

return 0;

```

CODE:-

```
#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[])
{
    Time::SetResolution (Time::NS);
    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);

    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;
}
```

EXPERIMENT No.4

Programin NS3 for connecting three nodes considering one node as a central node.

```
#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[])
{
    Time::SetResolution (Time::NS);
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

    NodeContainer nodes;
    nodes.Create (3);

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

    NetDeviceContainer devices, devices1;
    devices = pointToPoint.Install (nodes.get (0), nodes.get (1));
    devices1 = pointToPoint.Install (nodes.get (2), nodes.get (1));

    InternetStackHelper stack;
    stack.Install (nodes);

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

    Ipv4InterfaceContainer interfaces = address.Assign (devices);
    Ipv4InterfaceContainer interfaces1 = address.Assign (devices1);

    UdpEchoServerHelper echoServer (90);

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

    UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 90);
    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));

UdpEchoClientHelper echoClient (interfaces1.GetAddress (1), 90);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps1 = echoClient.Install (nodes.Get (2));
clientApps1.Start (Seconds (2.0));
clientApps1.Stop (Seconds (10.0));

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

EXPERIMENT No.5

Programin NS3 to implement star topology

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

// Network topology (default)
//
// n2 n3 n4 .
// \ / .
// \|/
// n1--- n0---n5 .
// /|\ .
// / | \ .
// n8 n7 n6 .
//

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Star");

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

    //
    // Set up some default values for the simulation.
    //
Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (137));

    // ??? try and stick 15kb/s into the data rate
Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue ("14kb/s"));

    //
    // Default number of nodes in the star. Overridable by command line argument.
    //
    uint32_t nSpokes = 8;

CommandLine cmd;
cmd.AddValue ("nSpokes", "Number of nodes to place in the star", nSpokes);
cmd.Parse (argc, argv);

NS_LOG_INFO ("Build star topology.");
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
PointToPointStarHelper star (nSpokes, pointToPoint);

NS_LOG_INFO ("Install internet stack on all nodes.");
```

```

InternetStackHelper internet;
star.InstallStack (internet);

NS_LOG_INFO ("Assign IP Addresses.");
star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));

NS_LOG_INFO ("Create applications.");
// Create a packet sink on the star "hub" to receive packets.
// uint16_t port = 50000;
Address hubLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress);
ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());
hubApp.Start (Seconds (1.0));
hubApp.Stop (Seconds (10.0));

// Create OnOff applications to send TCP to the hub, one on each spoke node.
//
OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
onOffHelper.SetAttribute ("OnTime", StringValue
    ("ns3::ConstantRandomVariable[Constant=1]"));
onOffHelper.SetAttribute ("OffTime", StringValue
    ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer spokeApps;

for (uint32_t i = 0; i < star.SpokeCount (); ++i)
{
AddressValue remoteAddress (InetSocketAddress (star.GetHubIpv4Address (i), port));
onOffHelper.SetAttribute ("Remote", remoteAddress);
spokeApps.Add (onOffHelper.Install (star.GetSpokeNode (i)));
}
spokeApps.Start (Seconds (1.0));
spokeApps.Stop (Seconds (10.0));
NS_LOG_INFO ("Enable static global routing.");
//
// Turn on global static routing so we can actually be routed across the star.
//
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

NS_LOG_INFO ("Enable pcap tracing.");
//
// Do pcap tracing on all point-to-point devices on all nodes.
//
pointToPoint.EnablePcapAll ("star");

NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");

return 0;

```

EXPERIMENT No.6

Program in NS3 to implement a bus topology.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"

// Default Network Topology

//      10.1.1.0

// n0 ----- n1  n2  n3  n4
//   point-to-point |  |  |  |
//                   =====
//                   LAN 10.1.2.0

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);
}

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);

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);
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 (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
Simulator::Run ();
Simulator::Destroy ();
return 0;
}

```

```

[ root@mta-192-180-1-110 ns-3.24.1]# ./waf --run udpro8
Waf: Entering directory `/home/mait/Downloads/ns-allinone-3.24.1/ns-3.24.1/build'
Waf: Leaving directory `/home/mait/Downloads/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.566s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.0078s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.0078s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.01761s client received 1024 bytes from 10.1.2.4 port 9
[ root@mta-192-180-1-110 ns-3.24.1]#

```

EXPERIMENT No.7

Installation and configuration of NetAnim

Installing NetAnim

The website:

<http://www.nsnam.org/wiki/index.php/NetAnim>

1. Install Mercurial:

```
apt-get/dnf install mercurial
```

2. Install QT4 development package:

```
apt-get/dnf install qt4-dev-tools
```

3. You can use Synaptic too, to install both the above packages.

4. Download NetAnim: hg clone http://code.nsnam.org/netanim

5. Build NetAnim:

```
cd netanim
```

```
make clean qmake
```

```
NetAnim.pro make
```

Compiling code with NetAnim

So you will have to make the following changes to the code, in order to view the animation on NetAnim.

```
#include "..."  
  
#include "ns3/netanim-module.h"           //1 Include...  
  
int main ( int argc , char *argv [ ] )  
  
{ std :: string animFile = "somename.xml"; //2 Name of file for animation  
  
    ...  
    AnimationInterface anim ( animFile );           //3 Animation interface  
    Simulator :: Run ();  
    Simulator :: Destroy ();  
    return 0;  
}
```

- 4.1 To run the code:

1. Move the waf , waf.bat , wsript and wutils.py files in to the scratch folder (~ns-allinone-3.24/ns-3.24/scratch/).
2. Move the example code to the scratch folder and make the changes required for NetAnim, as shown above.
3. Now cd to the scratch folder (cd ~ns-allinone-3.24/ns-3.24/scratch/).
4. Run the code using the command:

```
./waf --run <filename>
```

Note: <filename> should not contain the extension .cc

4.2 To visualize on NetAnim:

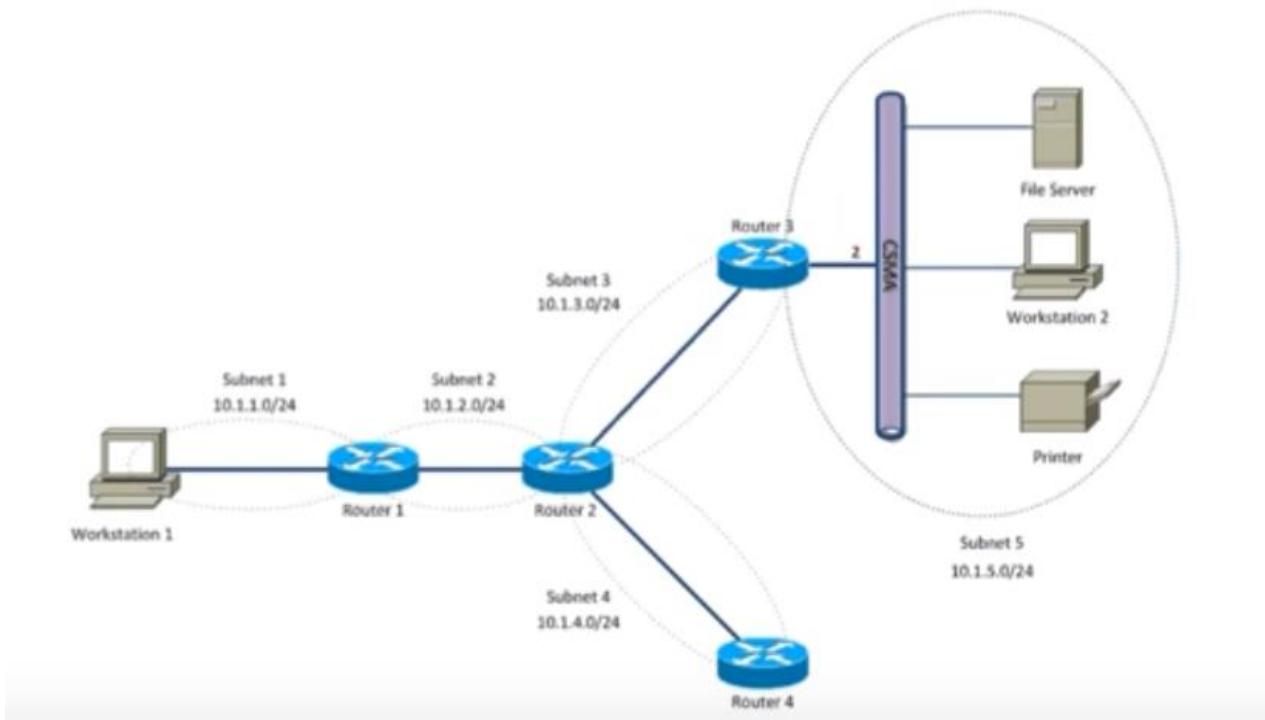
1. cd to the netanim folder (cd ~/netanim/).
2. Run Netanim:

```
./NetAnim
```
3. Include the .xml file generated in the ns-3.24 folder (~ns-allinone-3.17/ns3.24/).

EXPERIMENT-8

AIM- Write a program showing the connection of 2 nodes and 4 router such that the extremes nodes act as client and server.

BUILDING NETWORK TOPOLOGY



CODE:

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

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

```

#include "ns3/ipv4-global-routing-helper.h"

// Default Network Topology      10.1.5.0
//          r2-----n1
//          / 10.1.3.0
// no-----r0-----r1
// 10.1.1.0   10.1.2.0 \ 10.1.4.0
//                      r3

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");

int
main (int argc, char *argv[])
{
bool verbose = true;
if (verbose)
{
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}

NodeContainer host, router, host1;
host.Create (2);
router.Create (4);

NodeContainer subnet1;
subnet1.Add (host.Get (0));
subnet1.Add (router.Get (0));

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

NetDeviceContainer subnet1Devices;
subnet1Devices = pointToPoint.Install (subnet1);

InternetStackHelper stack;
stack.Install (router);
stack.Install (host);

Ipv4AddressHelper address1, address2, address3, address4, address5, address6,;
Address1.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer subnet1Interfaces;
subnet1Interfaces = address1.Assign (subnet1Devices);

NodeContainer subnet2;

```

```

subnet2.Add (router.Get (0));
subnet2.Add (router.Get (1));

NetDeviceContainer subnet2Devices;
subnet2Devices = pointToPoint.Install (subnet2);

address2.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer subnet2Interfaces;
subnet2Interfaces = address2.Assign (subnet2Devices);

NodeContainer subnet3;
subnet3.Add (router.Get (1));
subnet3.Add (router.Get (2));

NetDeviceContainer subnet3Devices;
subnet3Devices = pointToPoint.Install (subnet3);

address3.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer subnet3Interfaces;
subnet3Interfaces = address3.Assign (subnet3Devices);

NodeContainer subnet4;
subnet4.Add (router.Get (1));
subnet4.Add (router.Get (3));

NetDeviceContainer subnet4Devices;
subnet4Devices = pointToPoint.Install (subnet4);

address4.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer subnet4Interfaces;
subnet4Interfaces = address4.Assign (subnet4Devices);

NodeContainer subnet5;
subnet5.Add (router.Get (2));
subnet5.Add (host.Get (1));

NetDeviceContainer subnet5Devices;
subnet5Devices = pointToPoint.Install (subnet5);

address5.SetBase ("10.1.5.0", "255.255.255.0");
Ipv4InterfaceContainer subnet5Interfaces;
subnet5Interfaces = address5.Assign (subnet5Devices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (subnet5.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

```

```
UdpEchoClientHelper echoClient (subnet5Interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (3));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
ApplicationContainer clientApps = echoClient.Install (subnet1.Get (0));
clientApps.Start (Seconds (1.0));
clientApps.Stop (Seconds (10.0));
```

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

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

OUTPUT

```
[root@mta-192-180-1-231 ns-3.24.1]# ./waf --run scratch/pragati2
Waf: Entering directory `/home/mait/Downloads/ns-allinone-3.24.1/ns-3.24.1/buil
Waf: Leaving directory `/home/mait/Downloads/ns-allinone-3.24.1/ns-3.24.1/buil
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.631s)
At time 1s client sent 1024 bytes to 10.1.5.2 port 9
At time 1.01475s server received 1024 bytes from 10.1.1.1 port 49153
At time 1.01475s server sent 1024 bytes to 10.1.1.1 port 49153
At time 1.02949s client received 1024 bytes from 10.1.5.2 port 9
At time 2s client sent 1024 bytes to 10.1.5.2 port 9
At time 2.01475s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.01475s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.02949s client received 1024 bytes from 10.1.5.2 port 9
At time 3s client sent 1024 bytes to 10.1.5.2 port 9
At time 3.01475s server received 1024 bytes from 10.1.1.1 port 49153
At time 3.01475s server sent 1024 bytes to 10.1.1.1 port 49153
At time 3.02949s client received 1024 bytes from 10.1.5.2 port 9
[root@mta-192-180-1-231 ns-3.24.1]# 
```