**NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY**

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA

****

**COMPUTER NETWORKS LAB (14CSL68)**

Network simulator 3 Demo

*Submitted in partial fulfillment of the requirement for the course on CN-II (14CSL68)*

*Computer Science and Engineering*

*Submitted by:*

|  |  |
| --- | --- |
| B.SHRAVANI | 1NT16CS147 |
|  |  |
|  |  |

Department of Computer Science and Engineering

2018-19



**NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY**

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

, APPROVED BY AICTE & GOVT.OF KARNATAKA)

Department of Computer Science and Engineering

****

**CERTIFICATE**

This is to certify that the this is an authentic work carried out by **B.SHRAVANI(1NT16CS147)** bonafide students of Nitte Meenakshi Institute of Technology, Bangalore in partial fulfilment for the course in CN LAB (14CSL68),Department of COMPUTER SCIENCE AND ENGINEERING during the academic year ***2018-2019.***

**Course In-charge** SHARMILA SEQUEIRA,

PRATHEEKSHA HEGDE

Professors,

Dept. of CSE,

NMIT , Bangalore.

**Acknowledgment**

I would like to express my deepest appreciation to our Computer networks teachers for giving us this opportunity to present this demo.

I would also thank our head of computer science department **Dr. Thippeswamy M.N.** for guiding us through this project.

**Table of Contents**

1. Introduction
2. NS2 VS NS3
3. Setting up ns3
4. Code(4-Node TCP&UDP Data Transfer)
5. Working of code
6. Output
7. Conclusion
8. Bibliography

**1.INTRODUCTION**

**1.1NS-3**

The *NS-3* simulator is a discrete-event network simulator targeted primarily for research and educational use. The NS-3 project, started in 2006, is an open-source project developing *NS-3*.A few key points are worth noting at the onset,*NS-3* is open-source, and the project strives to maintain an open environment for researchers to contribute and share their software.

*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 or macOS systems, although support exists for BSD systems and also for Windows frameworks that can build Linux code, such as Windows Subsystem for Linux, or Cygwin. Native Windows Visual Studio is not presently supported although a developer is working on future support. Windows users may also use a Linux virtual machine.

**2.NS2 VS NS3**

For those familiar with *NS-2* (a popular tool that preceded *NS-3*), the most visible outward change when moving to *NS-3* is the choice of scripting language. Programs in *NS-2* are scripted in OTcl and results of simulations can be visualized using the Network Animator nam. It is not possible to run a simulation in *NS-2* purely from C++ (i.e., as a main() program without any OTcl). Moreover, some components of *ns-2* are written in C++ and others in OTcl. In *NS-3*, the simulator is written entirely in C++, with optional Python bindings. Simulation scripts can therefore be written in C++ or in Python. New animators and visualizers are available and under current development.

But there are similarities as well (both, for example, are based on C++ objects, and some code from NS-2 has already been ported to ns-3). We will try to highlight differences between NS-2 and NS-3 as we proceed in this tutorial.

A question that we often hear is “Should I still use NS-2 or move to NS-3?” , a user will be more productive with *NS-3* for the following reasons:

* *NS-3* is actively maintained with an active, responsive users mailing list, while *NS-2* is only lightly maintained and has not seen significant development in its main code tree for over a decade.
* *NS-3* provides features not available in *NS-2*, such as a implementation code execution environment (allowing users to run real implementation code in the simulator)
* *NS-3* provides a lower base level of abstraction compared with *ns-2*, allowing it to align better with how real systems are put together. Some limitations found in *NS-2* (such as supporting multiple types of interfaces on nodes correctly) have been remedied in *NS-3*.

**3.SETTING UP NS -3**

*NS-3* is built as a system of software libraries that work together. User programs can be written that links with (or imports from) these libraries. User programs are written in either the C++ or Python programming languages.

*NS-3* is distributed as source code, meaning that the target system needs to have a software development environment to build the libraries first, then build the user program. *NS-3* could in principle be distributed as pre-built libraries for selected systems, and in the future it may be distributed that way, but at present, many users actually do their work by editing *NS-3* itself, so having the source code around to rebuild the libraries is useful. If someone would like to undertake the job of making pre-built libraries and packages for operating systems, please contact the ns-developers mailing list.

**3.1.Prerequisites**

| **Prerequisite** | **Package/version** |
| --- | --- |
| C++ compiler | clang++ or g++ (g++ version 4.9 or greater) |
| Python | python2 version >= 2.7.10, or python3 version >=3.4 |
| Git | any recent version . |
| Tar | any recent version (to unpack an **ns-3 release**) |
| bunzip2 | any recent version (to uncompress an *ns-3* release) |

**3.2.Downloading**

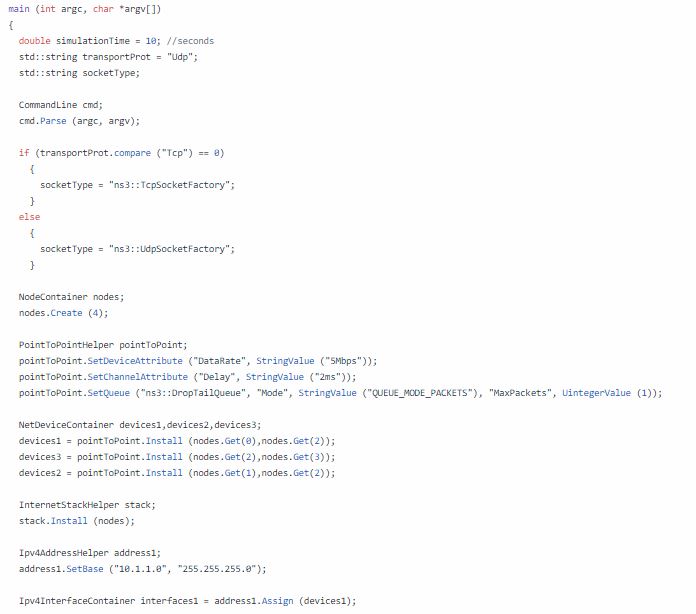
The simplest way to get started using Git repositories is to fork or clone the NS-3-allinone environment. This is a set of scripts that manages the downloading and building of the most commonly used subsystems of NS-3.

**3.3.Building NS-3**

Building with waf, **Waf** is a build automation tool designed to assist in the automatic compilation and installation of computer software.An installation of Waf is bundled with the NS-3 source code. Most users quickly transition to using Waf directly to configure and build NS-3.  Running the script To run a program, simply use the --run option in Waf. Let’s run the NS-3 equivalent of the ubiquitous hello world program by typing **“$ ./waf --run hello-simulator”**

**4.CODE**

****

****

****

****

****

**5.WORKING OF THE CODE**

The first line in the file is an emacs mode line. This tells emacs about the formatting conventions (coding style) we use in our source code.

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

The first line ensures that developers who use the emacs editor will be able to indent your code correctly. The following lines ensure that your code is licensed under the GPL, that the copyright holders are properly identified (typically, you or your employer), and that the actual author of the code is identified. The latter is purely informational and we use it to try to track the most appropriate person to review a patch or fix a bug.

**Module includes and logging**

**#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"***

The set of headers listed in our script are used to accomplish different types of services ,say.,to set up simulator,for local Mac address,IPv4,to create ftp/cbr packets,to handle flow,to check delay,throughput….,etc.

The next line **using namespace ns3;** makes every class, function and variable declared inside the ns-3 namespace available for direct usage without using the qualifier ns3:: The ns-3 project is implemented in a C++ namespace called ns3. This groups all ns-3-related declarations in a scope outside the global namespace, This is a fancy way of saying that after this declaration, you will not have to type ns3:: scope resolution operator before all of the ns-3 code in order to use it.

Remember, we havent used the standard namespace std of C++. So we need the qualifier std:: to get standard C++ constructs, e.g., std::cout<< Hello ns-3; will work but cout<<Hello ns-3;will result in an error.

**NS\_LOG\_COMPONENT\_DEFINE**(SimulationOne).,which is our next line,declares logging component named SimulationOne. By referring to this logging object console, message logging can be enabled and disabled.

The next line our script **int main (int argc, char \*argv[]){**This is just the declaration of the main function of your program (script). Just as in any C++ program, you need to define a main function that will be the first function run. There is nothing at all special here. Your ns-3 script is just a C++ program.

**CommandLine cmd;**

**cmd.Parse (argc, argv);**

The above lines of our script is the another way you can change how ns-3 scripts behave without editing and building is via command line arguments. We provide a mechanism to parse command line arguments and automatically set local and global variables based on those arguments.The first step in using the command line argument system is to declare the command line parser.

**Node Container**

In Internet jargon, a computing device that connects to a network is called host or sometimes an end-system.Because ns-3 is network simulator,not specifically an Internet simulator, we intentionally do not use the term host since it is closely associated with the Internet and its protocols. Instead, we use a more generic term also used by other simulators that originates in Graph Theory — the node.You should think of a Node as a computer to which you will add functionality. One adds things like applications, protocol stacks and peripheral cards with their associated drivers to enable the computer to do useful work.

The next two lines of code in our script will actually create the ns-3 Node objects that will represent the computers in the simulation.

**NodeContainer nodes;**

**nodes.Create (2);**

This represents a computer to which we are going to add things like protocol stacks, applications and peripheral cards. The NodeContainer topology helper provides a convenient way to create, manage and access any Node objects that we create in order to run a simulation. The first line above just declares a NodeContainer which we call nodes. The second line calls the Create method on the nodes object and asks the container to create two nodes. Hence it create two Node objects and stores pointers to those objects internally.

**PointToPointHelper**

Two of our key abstractions are the NetDevice and the Channel. In the real world, these terms correspond roughly to peripheral cards and network cables. We use a single PointToPointHelper to configure and connect ns- 3 device and channel.

The next three lines in the script are,

**PointToPointHelper pointToPoint;**

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

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

The first line,instantiates a PointToPointHelper object on the stack from a high-level perspective.

The next line tells the PointToPointHelper object to use the value “5Mbps” (five megabits per second) as the “DataRate” when it creates a PointToPointNetDevice object.

The final line, tells the PointToPointHelper to use the value “2ms” (two milliseconds) as the value of the propagation delay of every point to point channel it subsequently creates.

**NetDevice Container**

At this point in the script, we have a NodeContainer that contains two nodes. We have a PointToPointHelper that is primed and ready to make PointToPointNetDevices and wire PointToPointChannel objects between them. Just as we used the NodeContainertopology helper object to create the Nodes for our simulation, we will ask the  PointToPointHelper  to do the work involved in creating, configuring and installing our devices for us. We will need to have a list of all of the NetDevice objects that are created, so we use a NetDeviceContainer to hold them just as we used a NodeContainer to hold the nodes we created. The following two lines of code,

**NetDeviceContainer devices;**

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

will finish configuring the devices and channel. The first line declares the device container mentioned above ,the Install method of the PointToPointHelper takes a NodeContainer as a parameter. Internally, a NetDeviceContainer is created.

**InternetStackHelper**

We now have nodes and devices configured, but we don’t have any protocol stacks installed on our nodes. The next two lines of code will take care of that.

**InternetStackHelper stack;**

**stack.Install (nodes);**

The InternetStackHelper is a topology helper that is to internet stacks what the PointToPointHelper is to point-to-point net devices. The Install method takes a NodeContainer as a parameter. When it is executed, it will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.

**Ipv4AddressHelper**

Next we need to associate the devices on our nodes with IP addresses. We provide a topology helper to manage the allocation of IP addresses. The only user-visible API is to set the base IP address and network mask to use when performing the actual address allocation (which is done at a lower level inside the helper).

The next two lines of code in our script,

**Ipv4AddressHelper address;**

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

declares an address helper object and tell it that it should begin allocating IP addresses from the network 10.1.1.0 using the mask 255.255.255.0 to define the allocatable bits. By default the addresses allocated will start at one and increase monotonically, so the first address allocated from this base will be 10.1.1.1, followed by 10.1.1.2, etc.

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 (which is a very hard to debug error, by the way).

The next line of code,

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

performs the actual address assignment. In *ns-3* we make the association between an IP address and a device using an Ipv4Interface object.

**Applications set up**

System Software organizes various computer resources such as memory, processor cycles, disk, network, etc., according to some computing model. System software usually does not use those resources to complete tasks that directly benefit a user. A user would typically run an application that acquires and uses the resources controlled by the system software to accomplish some goal.

Applications are set up that use *tcp* and *udp* , first a port number is set, and an address class holds multiple values is set up, an application instance is created using a **packetsinkhelper** and the application instance is stored in **applicationcontainer**.

An **onoffhelper** is used to help instantiate an *application* , here it sets the attributes such as *ontime , offtime , packetsize, datarate* .

Finally the applications are started.

**Flow monitors**

A **flowmonitor** object is used to monitor the packet flows , a flowmonitor helper method installall() returns an object of flowmonitor that monitor all nodes.

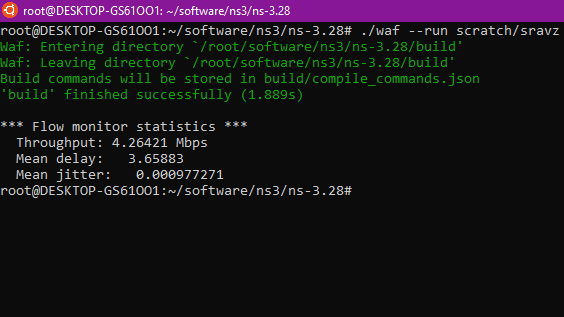
**Simulator**

What we need to do at this point is to actually run the simulation. This is done using the global function **Simulator::Run**. And similarly stop and destroy .When the events scheduled are executed, there are no further events to process and Simulator::Run returns. The simulation is then complete.

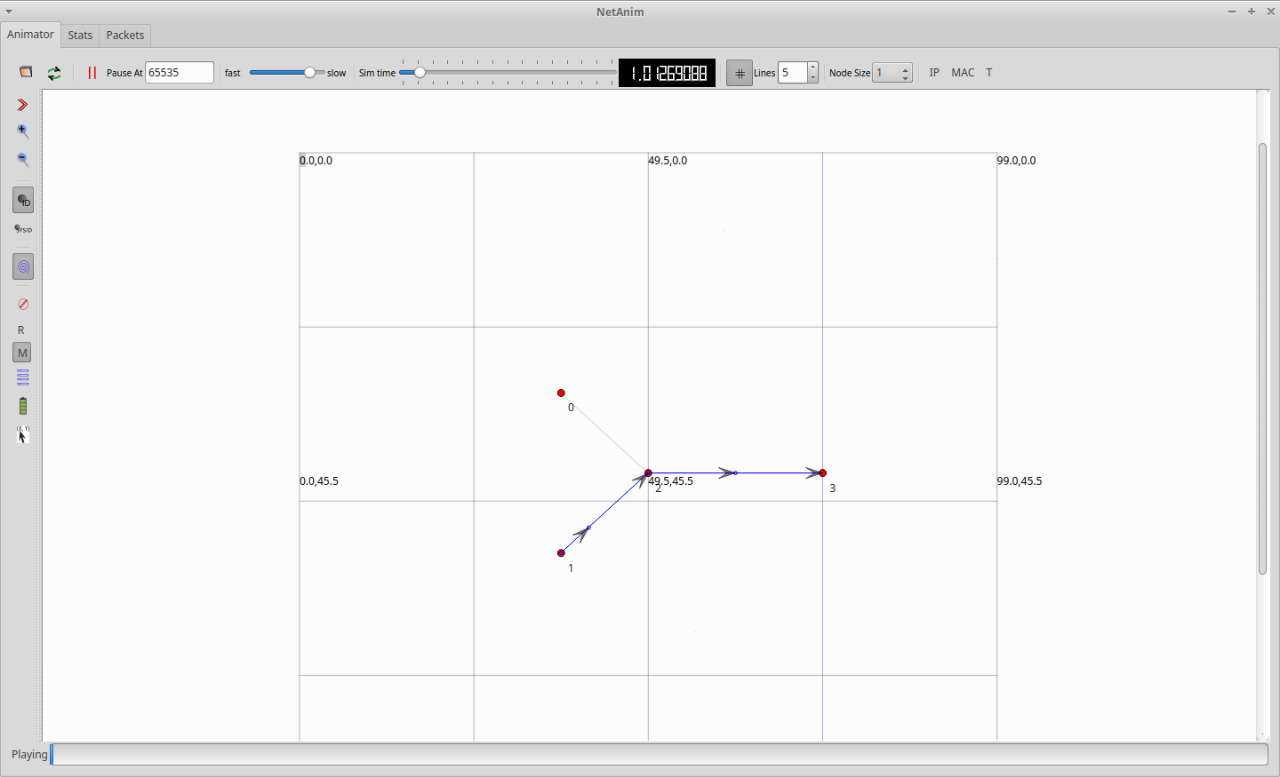
All that remains is to clean up. This is done by calling the global function **Simulator::Destroy**. As the helper functions (or low level ns-3 code) executed, they arranged it so that hooks were inserted in the simulator to destroy all of the objects that were created. You did not have to keep track of any of these objects yourself — all you had to do was to call Simulator::Destroy and exit. The ns-3 system took care of the hard part for you.

**6.OUTPUT**

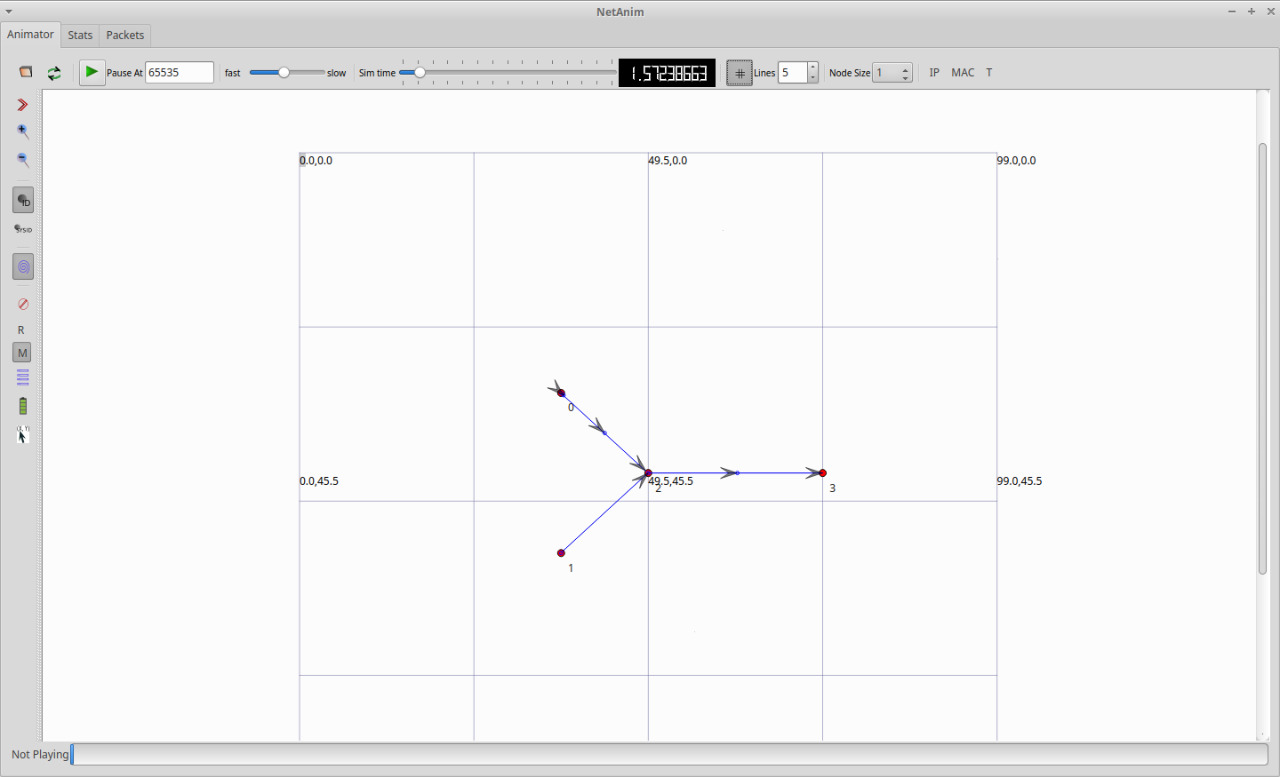
*Fig1:Running ns-3 script using waf*

****

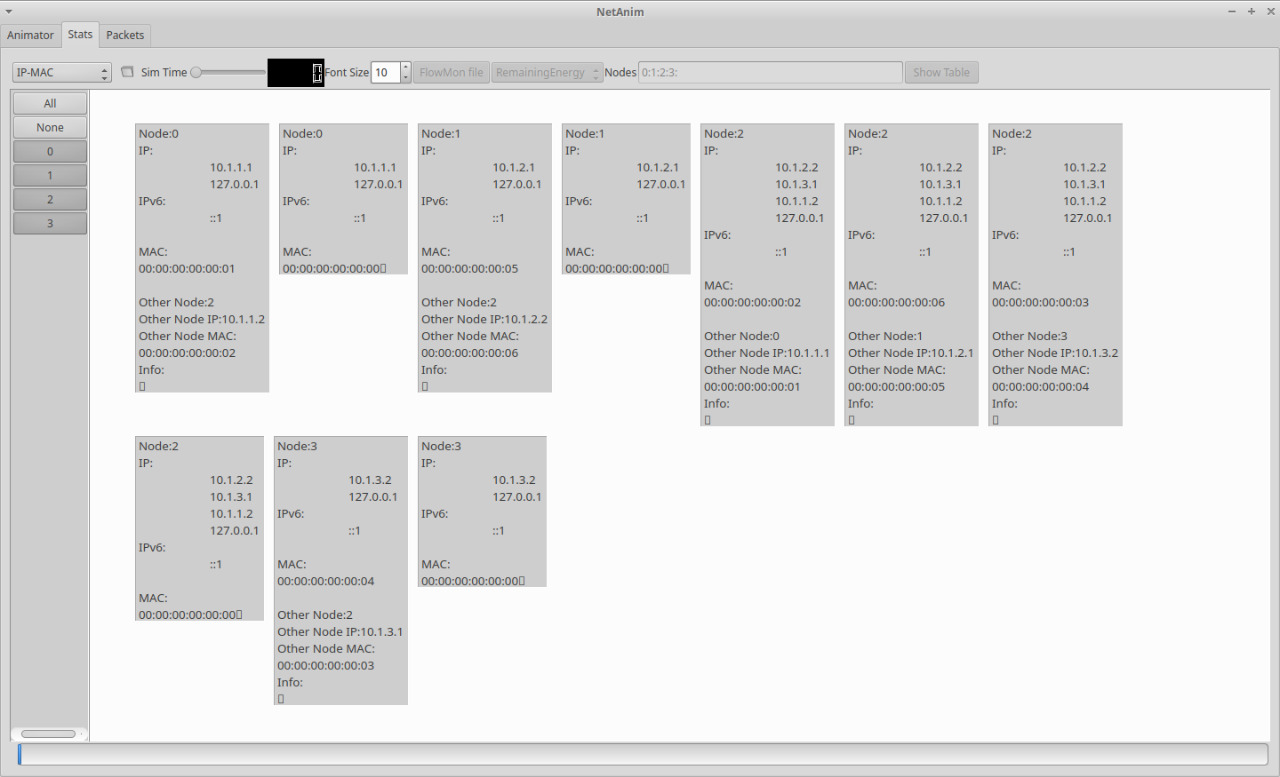
*Fig2: Animated Topology at 1.01269088*

**

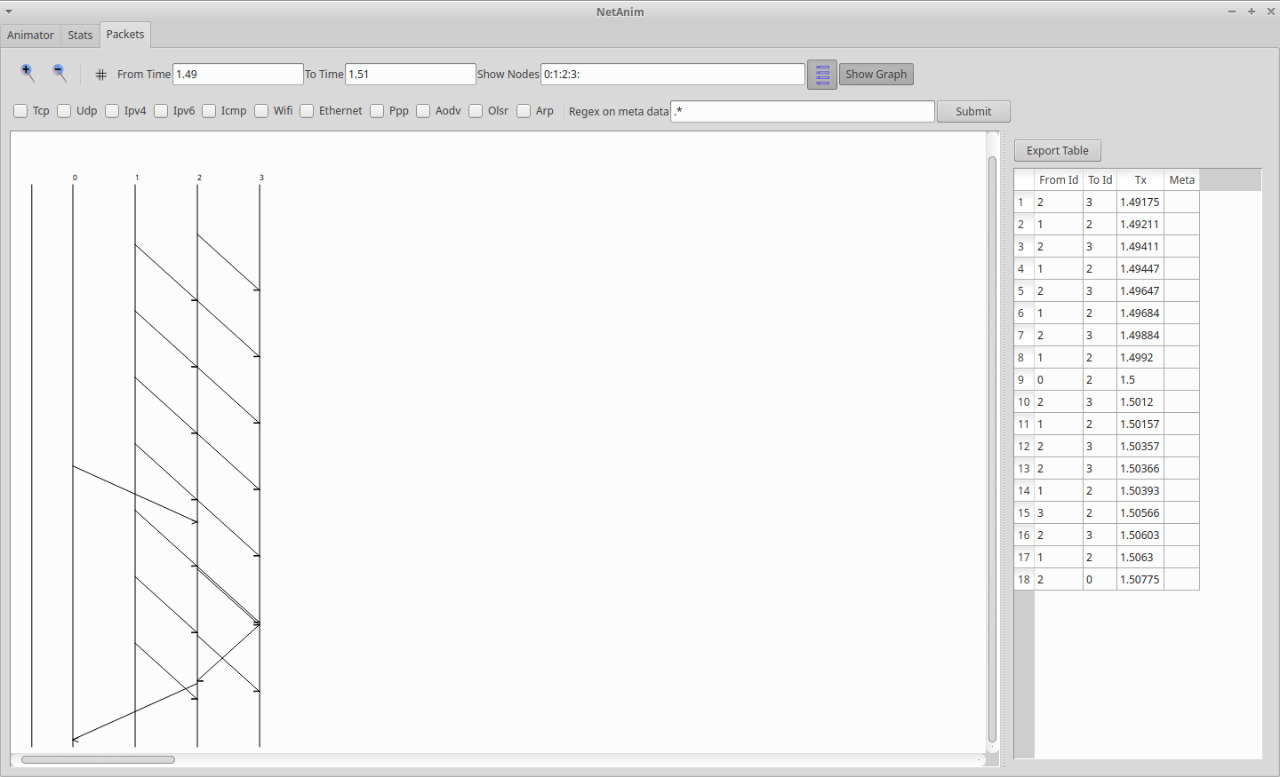
*Fig3: Animated Topology at 1.57238663*



*Fig4:Nodes information(stats)*

**

*Fig5:Representation of flow of packets form Node to Node*

**

**7.CONCLUSION**

A network simulation was set up, simulated and analyzed using NS-3. A simple *4-node point-to-point TCP/UDP data* transfer is designed and established successfully using NS-3 simulator

**8.BIBLIOGRAPHY**

1. <https://www.nsnam.org/>
2. [https://en.wikipedia.org](https://en.wikipedia.org/)