Corso: Networked Embedded & IoT Systems

Docente: Prof. David Guaglia

Studente: Monruethai Sueksakan (VR466571)

Data di consegna: 02/08/2024

Simulation of networks with ns-3 and OMNeT++

Simulating networks with ns-3 and OMNeT++ is a common process for evaluating the performance and fidelity of different network protocols to compare them and determine which is best suited for certain scenarios or applications. Both tools are powerful network simulators, but they have slightly different approaches and features. Here is a brief overview of how to perform simulations with both:

- **ns-3 (Network Simulator 3):**

1. Installation: First, you need to download and install ns-3 on your system following the official instructions provided on the ns-3 website.

2. Choice of Network Model: ns-3 offers a wide range of network models, protocols and simulation scenarios. You must choose the appropriate network model for your scenario of interest.

3. Code Writing: Using the C++ or Python programming language, you can write simulation scripts to configure the network topology, nodes, links and protocol behaviors.

4. Simulation Execution: Once the simulation script is written, you can execute the simulation using the ns-3 environment. During execution, ns-3 records network performance and behavior metrics that can be analyzed later.

- **OMNeT++:**

1. Installation: Similar to ns-3, you need to download and install OMNeT++ following the instructions provided on the official website.

https://doc.omnetpp.org/omnetpp/InstallGuide.pdf

2. Network Templates: OMNeT++ uses the concept of "templates" to represent network nodes, channels and protocols. You need to select or develop the appropriate models for the desired simulation.

3. Configuration and Design: Using OMNeT++'s graphical design environment, you can configure the network topology, simulation parameters and node behaviors.

4. Simulation Execution: After configuring the simulation model, you can execute the simulation through the interface of OMNeT++. During execution, OMNeT++ collects simulation data that can be analyzed later.

Comparison:

To determine which tool is faster and more faithful, it is necessary to run identical or similar simulations using both tools and compare the results obtained. Several factors can be considered, including:

- Execution Time: Measure how long it takes to complete a simulation on both platforms.
- Results Accuracy: Compare the performance metrics and simulation data produced by both tools to assess the fidelity of the results.
- Ease of Use: Consider the ease of use, flexibility and documentation available for both tools.

Ultimately, the choice between ns-3 and OMNeT++ depends on the specific needs of the project, personal preferences, and simulator features that best fit the simulation context.

Conclusion

Simulating networks with ns-3 and OMNeT++ is a common practice for evaluating network protocols and determining their suitability for specific scenarios or applications.

Both ns-3 and OMNeT++ are powerful network simulators, although they differ in their approaches and features. To simulate networks using ns-3, you need to install the tool, choose a network model, write simulation scripts using C++ or Python, and execute the simulation within the ns-3 environment. On the other hand, to simulate networks using OMNeT++, you need to install the software, select appropriate network templates, configure network topology and parameters using the graphical design environment, and execute the simulation through the OMNeT++ interface.

To compare the two tools, factors such as execution time, accuracy of results, and ease of use should be considered. The choice between ns-3 and OMNeT++ ultimately depends on project needs, personal preferences, and the specific features that align with the simulation context.

So i choose to focus on OMNet++ simulation

OMNeT++ is a discrete event simulation environment primarily used for simulating communication networks, multiprocessors, and other distributed systems. It provides an extensible, modular architecture and comes with a rich library of components for building complex simulations. Here’s a guide on how OMNeT++ simulation works and how to create and run a simple simulation.

Installation

Before running simulations in OMNeT++, you need to install it. Here’s a brief guide for installation:

1. Download OMNeT++: Go to the [OMNeT++ website](https://omnetpp.org/) and download the latest version.

2. Install Required Software:

   - On Linux: Install build-essential, gcc, g++, make, and other required libraries.

   - On Windows: Install MinGW and MSYS.

   - On macOS: Install Xcode and its command-line tools.

3. Extract and Install OMNeT++: Extract the downloaded archive and follow the installation instructions in the `README` or `INSTALL` file.

4. Set Up Environment: Add OMNeT++ binary directories to your system path.

Basic Concepts

1. Modules: The basic building blocks of OMNeT++ models. Modules can be simple (atomic) or compound (nested).

2. Channels: Define communication paths between modules.

3. Messages: Used for communication between modules.

4. Network Description: Written in NED (Network Description) language which defines the structure and connections of the network.

5. Simulation Kernel: Manages the simulation events and scheduling.

Creating a Simple Simulation

Step 1: Project Setup

1. Create a New Project: Open OMNeT++ IDE and create a new project.

   - File -> New -> OMNeT++ Project

2. Project Structure:

   - src/: Source files.

   - simulations/: Configuration and NED files.

Step 2: Define Network Topology (NED Files)

Create a NED file that describes the network structure.

Example: `simple_network.ned`

```ned
network SimpleNetwork
{
```

```
  submodules:
    node1: StandardHost;
    node2: StandardHost;
  connections:
    node1.ethg++ <--> Ethernet100m <--> node2.ethg++;
}
```

Step 3: Define Simple Module

Create a simple module that performs basic operations, like sending and receiving messages.

Example: `SimpleModule.ned`

```ned
simple SimpleModule
{
  parameters:
    @display("i=block/routing");
  gates:
    input in;
    output out;
}
```

Example: `SimpleModule.cc`

```cpp
```

```
include <omnetpp.h>

using namespace omnetpp;

class SimpleModule : public cSimpleModule
{
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(SimpleModule);

void SimpleModule::initialize()
{
    if (par("sendInitialMessage").boolValue()) {
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
    }
}

void SimpleModule::handleMessage(cMessage *msg)
{
    send(msg, "out");
}
```

Step 4: Simulation Configuration (omnetpp.ini)

Configure the simulation parameters, network to use, and other settings in `omnetpp.ini`.

Example: `omnetpp.ini`

```ini
[General]
network = SimpleNetwork
sim-time-limit = 10s
.node1.numApps = 1
.node1.app[0].typename = "UdpBasicApp"
.node1.app[0].destAddresses = "node2"
.node1.app[0].destPort = 5000
.node2.numApps = 1
.node2.app[0].typename = "UdpSink"
.node2.app[0].localPort = 5000
```

Step 5: Build and Run the Simulation

1. Build the Project:
   - Right-click on the project -> Build Project
2. Run the Simulation:
   - Right-click on `omnetpp.ini` -> Run As -> OMNeT++ Simulation

Visualization and Analysis

OMNeT++ provides graphical tools for running simulations, visualizing the network, and analyzing results:

- Tkenv: GUI environment for interactive simulation.

- Qtenv: Modern GUI environment (alternative to Tkenv).

- Plove: Tool for plotting simulation results.

Advanced Features

OMNeT++ offers various advanced features:

- Mobility: Simulate mobile nodes and changing network topologies.

- Custom Protocols: Implement custom network protocols and applications.

- Integration with Other Tools: Integrate with other simulation frameworks like INET, Veins, etc., for more complex simulations.

Example Workflow

1. Design the Network: Define the network structure using NED files.

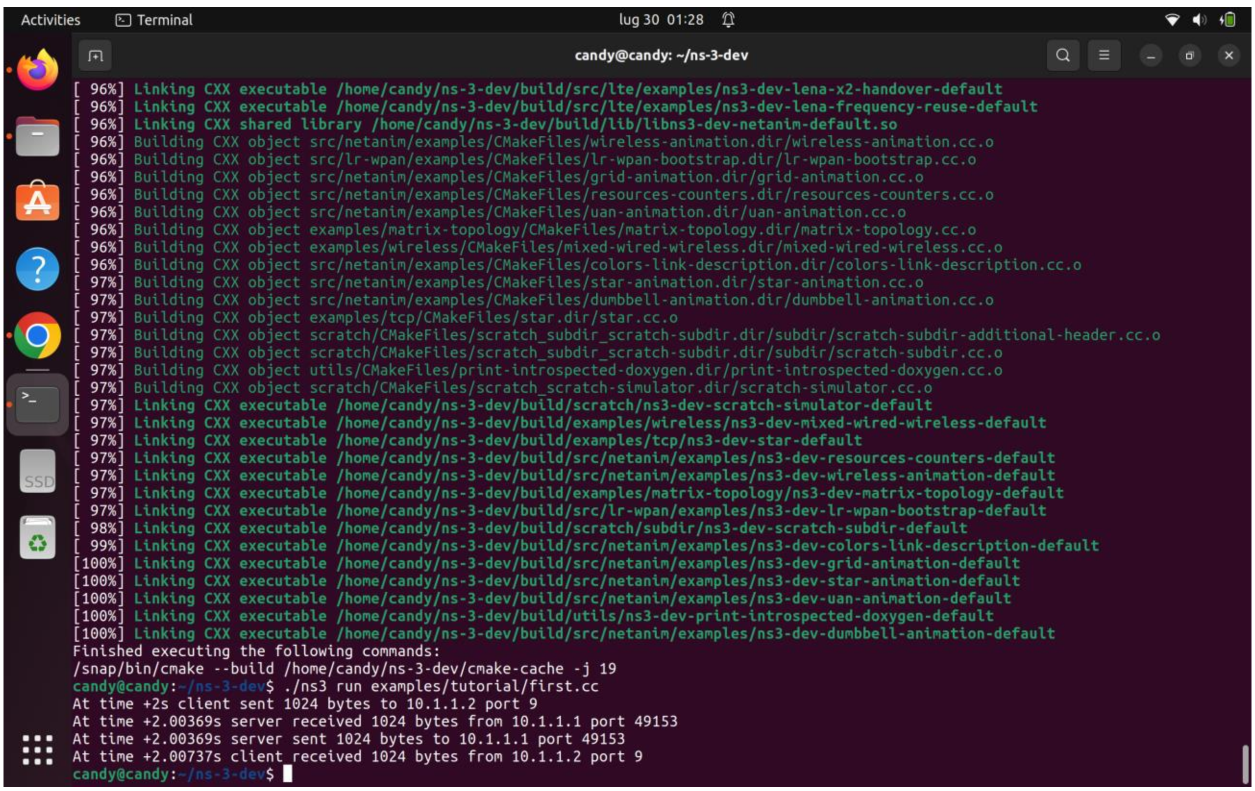
2. Implement Modules: Write the logic for simple and compound modules in C++.

3. Configure Simulation: Set simulation parameters in `omnetpp.ini`.

4. Build and Execute: Compile the project and run the simulation using the provided GUI.

5. Analyze Results: Use OMNeT++ tools to visualize and analyze simulation results.

Starting with ns3



The output you provided indicates that the execution of the first.cc example was successful

The output you provided indicates that the execution of the first.cc example was successful. This example is a simple demonstration in NS-3, showcasing a client sending packets to a server, which then replies to the client.

The output lines provide details about the simulation events:

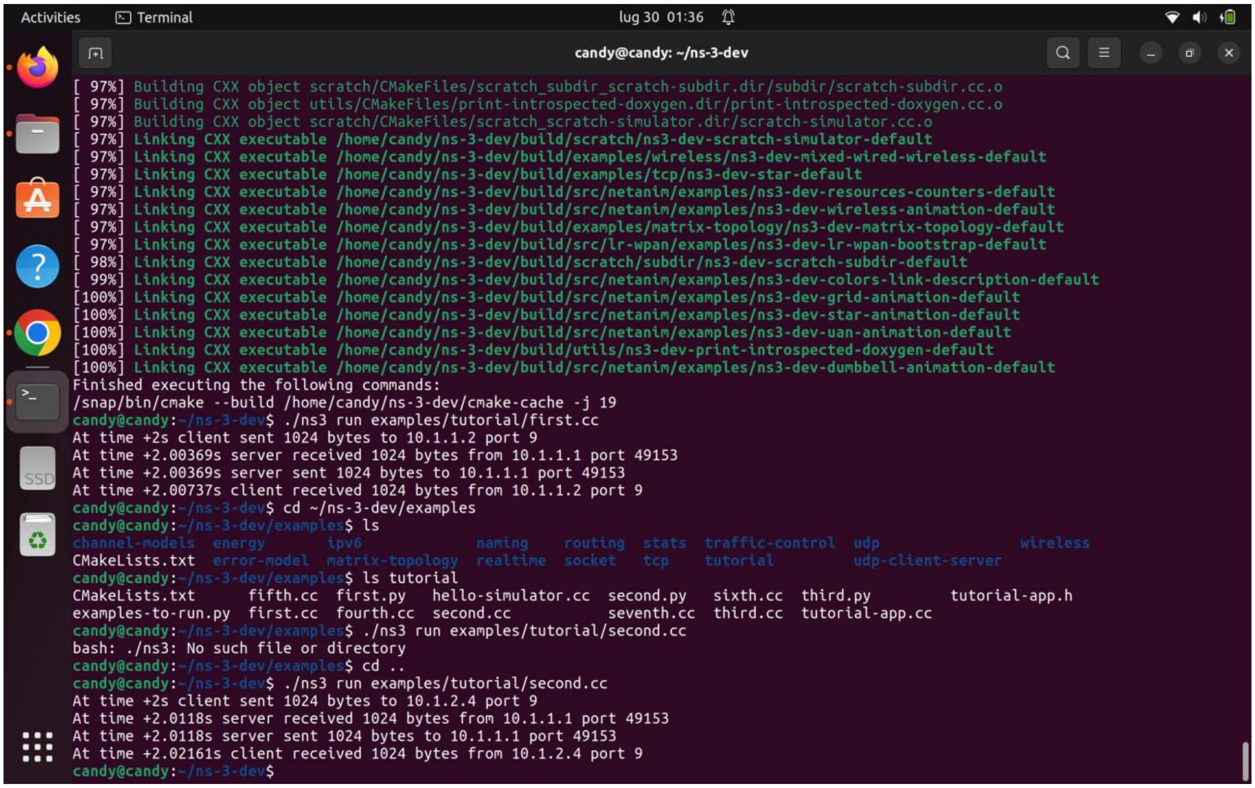
1.**Packet Sent**: The client sends a 1024-byte packet to the server.
2.**Packet Received by Server**: The server receives the packet from the client and responds.
3.**Response Received by Client**: The client receives the server's response.

These results confirm that your NS-3 setup is correctly configured and working as expected.

Next Steps

Now that your setup is complete and everything is functioning, you can proceed with the following:

1. **Explore Other Examples**: Try running other examples included in NS-3 to understand its capabilities better. You can find these examples in the examples directory.
2. **Create Custom Simulations**: Start developing your simulation scripts to model specific network scenarios that interest you.
3. **Visualize Results**: If you're using tools like NetAnim, you can visualize the simulations graphically. Ensure that the netanim module is enabled and properly configured.



The output indicates that the second.cc example from the NS-3 tutorial has successfully run

This example likely demonstrates a more complex scenario than first.cc, possibly involving multiple network nodes or different types of network configurations.

Understanding the Output

1. **Client Sent Data**: The client node sent a 1024-byte packet to the server located at IP address 10.1.2.4 on port 9 at simulation time +2s.
2. **Server Received Data**: The server received the packet from the client at IP address 10.1.1.1 with a source port of 49153 at simulation time +2.0118s.
3. **Server Responded**: The server then sent a 1024-byte response back to the client.
4. **Client Received Response**: The client received the response from the server at simulation time +2.02161s.

These events confirm that the communication between the client and server nodes was successfully simulated, showing the data exchange process.

Next Steps

1. **Explore Different Scenarios**: Continue exploring other examples in the examples directory. Each example showcases different aspects of network simulations, such as routing protocols, mobility models, and wireless communication.
2. **Modify Examples**: Try modifying the examples to change parameters like packet size, transmission times, network topology, etc. This will help you understand how different factors affect network performance.
3. **Create Custom Simulations**: Based on what you learn, start creating your own custom simulation scripts. Place them in the scratch directory or any other directory and use the same ./ns3 run command to execute them.
4. **Use Visualization Tools**: If you want to visualize the network topology and packet flows, consider using visualization tools like NetAnim. Make sure to configure and enable NetAnim in your NS-3 setup.
5. **Deep Dive into Documentation**: Refer to the NS-3 documentation for detailed explanations of specific modules, APIs, and configuration options.

To simulate a network scenario with TCP for transferring a file between two nodes in NS-3, you can use a combination of NS-3 modules to set up the network, establish TCP connections, and simulate file transfers.

Steps to Create the TCP File Transfer Simulation

1. **Setup the Network Topology**
2. **Configure TCP Protocol**
3. **Set Up Applications for File Transfer**
4. **Run the Simulation and Collect Results**

Here's a step-by-step guide with a simple example:

1. Setup the Network Topology

First, create a basic network topology. For this example, we can set up two nodes connected via a point-to-point link.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
```

```cpp
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;

int main(int argc, char *argv[])
{
// Enable logging
LogComponentEnable("BulkSendApplication", LOG_LEVEL_INFO);
LogComponentEnable("PacketSink", LOG_LEVEL_INFO);



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

// Configure the point-to-point link
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));

// Install the point-to-point devices on the nodes
NetDeviceContainer devices;
devices = pointToPoint.Install(nodes);

// Install the internet stack on the nodes
InternetStackHelper stack;
stack.Install(nodes);

// Assign IP addresses
Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(devices);

// Set up the TCP server on node 1
uint16_t port = 9; // Port number
Address serverAddress = InetSocketAddress(interfaces.GetAddress(1), port);
PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory", serverAddress);
ApplicationContainer serverApp = packetSinkHelper.Install(nodes.Get(1));
serverApp.Start(Seconds(1.0));
serverApp.Stop(Seconds(10.0));
```

```
// Set up the TCP client on node 0
OnOffHelper clientHelper("ns3::TcpSocketFactory", serverAddress);
clientHelper.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
clientHelper.SetAttribute("DataRate", DataRateValue(DataRate("5Mbps")));
clientHelper.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientApp = clientHelper.Install(nodes.Get(0));
clientApp.Start(Seconds(2.0));
clientApp.Stop(Seconds(10.0));

// Set up NetAnim
AnimationInterface anim("animation.xml");
// Run the simulation
Simulator::Run();
Simulator::Destroy();

return 0;
}
```

2. Configure TCP Protocol

In the code above, the TCP protocol is automatically used by the ns3::TcpSocketFactory. You can further configure TCP attributes like congestion control algorithms, segment size, etc., by using NS-3's TCP-related classes and methods.

3. Set Up Applications for File Transfer

- **Server**: The server is set up using PacketSinkHelper, which will listen for incoming TCP connections.
- **Client**: The client is configured using OnOffHelper to send data over TCP. The OnTime and OffTime attributes control the data transmission pattern, and DataRate specifies the speed.

4. Run the Simulation and Collect Results

- **Run the Simulation**: The simulation is run for 10 seconds, during which the client will attempt to send data to the server.
- **Results and Analysis**: Use the NS-3 logging or tracing capabilities to gather data on throughput, packet delivery, and other metrics.

Additional Considerations

- **File Size and Transfer Time**: To simulate a specific file size, adjust the data rate and packet size accordingly.
- **Congestion Control**: You can experiment with different TCP congestion control algorithms by setting ns3::TcpL4Protocol::SocketType to different values like TcpNewReno, TcpWestwood, etc.
- **Advanced Scenarios**: For more complex simulations, you can include multiple clients, different network topologies, or simulate different network conditions (e.g., packet loss, delay).

Running the Simulation

To compile and run this script, save it to a file (home/ns-3-dev/scratch/

tcp-file-transfer.cc) in the scratch directory of your NS-3 installation, then build and run it:

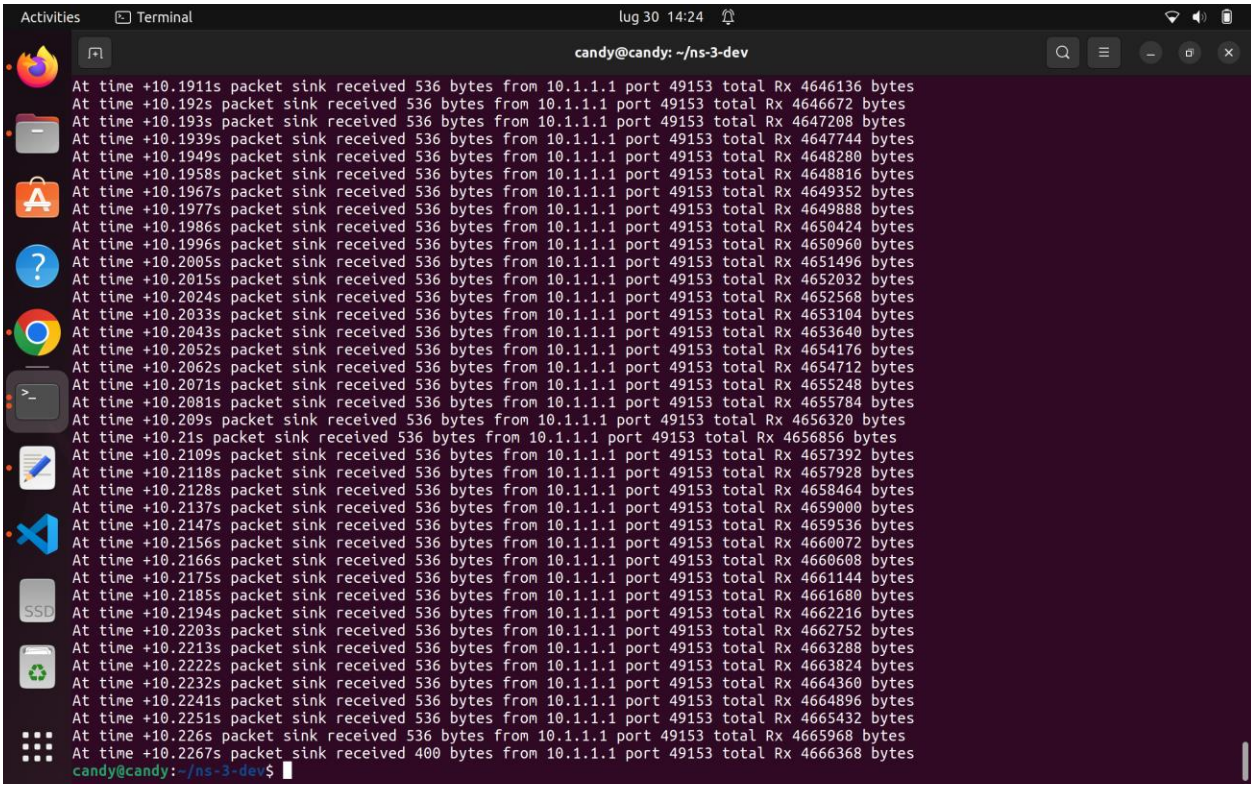
```
cd ~/ns-3-dev

./ns3 build

./ns3 run scratch/tcp-file-transfer
```

This basic setup provides a foundation for simulating TCP file transfers in NS-3.

Result



TCP File Transfer:

1. **Ritardo di propagazione nel canale**:
   - Nella simulazione, il ritardo di propagazione è configurato tramite l'attributo "Delay" del PointToPointHelper. Per esempio, se è stato impostato un valore di "2ms", significa che ogni pacchetto impiega 2 millisecondi per attraversare il canale di comunicazione tra il client e il server.
2. **Capacità del canale**:
   - La capacità del canale è definita dall'attributo "DataRate" del PointToPointHelper. Se impostato su "5Mbps", il canale può gestire fino a 5 Megabit di dati al secondo. Questo determina la larghezza di banda massima disponibile per il trasferimento dati tra il client e il server.
3. **Flusso di pacchetti tra client e server**:
   - Dalla schermata allegata, vediamo che il PacketSink (server) ha ricevuto pacchetti da 536 byte ciascuno dall'indirizzo IP 10.1.1.1 con un flusso costante fino a un totale di 4666368 byte ricevuti. Questo indica un trasferimento dati continuo tra il client (nodo 0) e il server (nodo 1) su una porta TCP specifica (49153). Il flusso di pacchetti è rappresentato dall'invio continuo di pacchetti dal client al server, con ogni ricezione registrata indicando il numero di byte ricevuti e il tempo di ricezione.

The output you've provided details the data received by the PacketSink application in an NS-3 simulation, indicating successful data transfer over a TCP connection. Let's break down what each part of the output means:

Breakdown of the Output

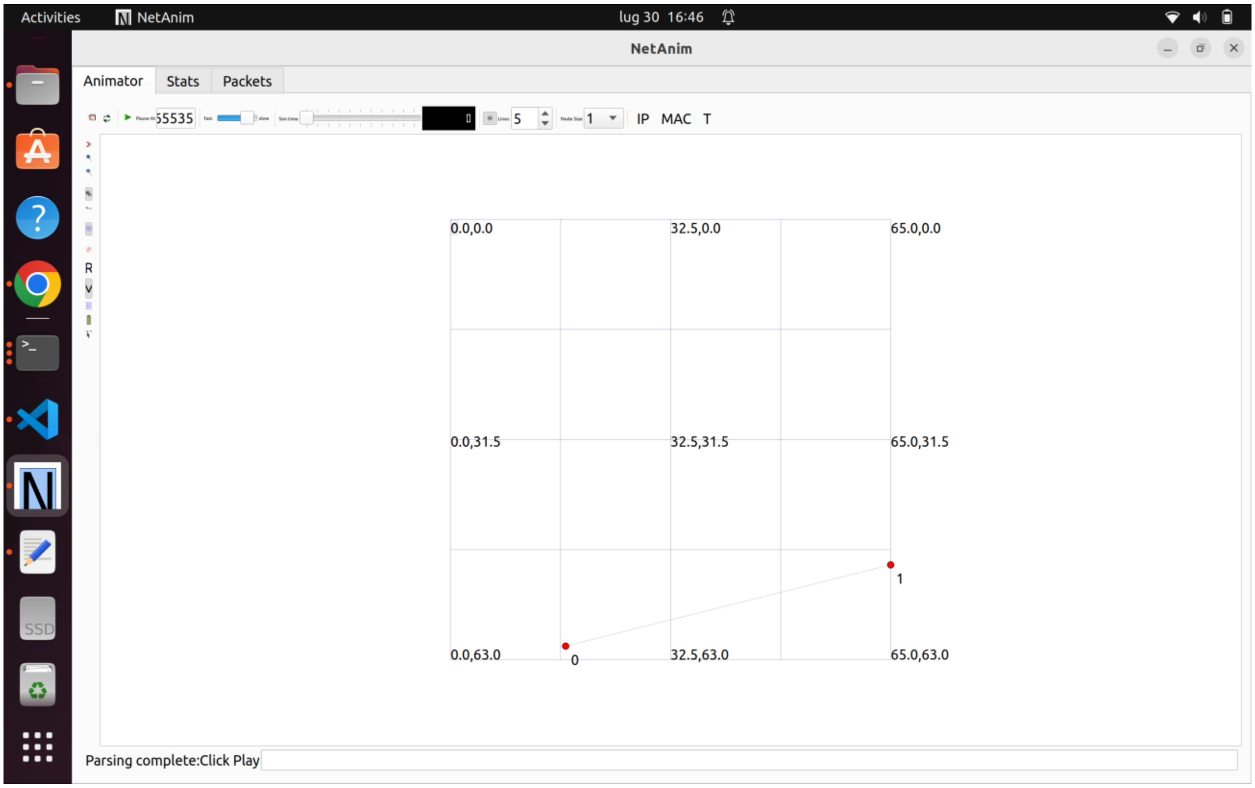
1. **Timestamp (At time +x.xxxs)**:
   - This indicates the simulation time at which the event occurred. The format +x.xxxs shows the time in seconds since the start of the simulation.
2. **Packet Sink Information (packet sink received 536 bytes from 10.1.1.1 port 49153)**:
   - packet sink received 536 bytes: This shows that the PacketSink application received a packet of 536 bytes.
   - from 10.1.1.1: The source IP address of the packet, indicating the sender's IP.
   - port 49153: The source port number from which the packet was sent.
3. **Total Received (total Rx xxxx bytes)**:
   - This indicates the cumulative total of bytes received by the PacketSink application up to that point in the simulation. The value increases as more data packets are received.

Explanation of Key Concepts

- **Packet Size**: The packets received are mostly 536 bytes, which is a typical payload size for TCP segments in a simulated environment like NS-3.
- **Cumulative Data**: The total Rx value shows the cumulative amount of data received. This is a running total that increases with each packet received, allowing you to track the amount of data transferred over the duration of the simulation.
- **Final Packet**: The last entry shows a packet of 400 bytes, which might be the last segment of the transfer, often smaller if the data doesn't perfectly fit into the previous full-sized packets.

This output confirms that the TCP file transfer simulation worked correctly. The client sent data packets to the server (PacketSink), which received them, incrementing the total received data count. You can visualize this data flow using the generated animation.xml file in NetAnim, where you'll be able to see the packet movements and interactions between nodes in the network.

Then i was tried to use NetAmin visualization



In the NetAnim interface, you can visualize the simulation of your ns-3 network scenario. Here's a breakdown of what you're seeing:

1.Grid and Nodes:

- The grid represents the simulation area.
- The red dots labeled "0" and "1" are the two nodes in your network. Node 0 is the client, and Node 1 is the server.

2.Line Between Nodes:

- The line between the nodes represents the point-to-point link established between them. It indicates that there is a direct connection, which in your simulation represents the TCP connection over which data is transferred.

3.Controls and Display Options:

- At the top of the interface, you have controls to play, pause, and adjust the speed of the animation. You can use these controls to visualize the flow of data packets over time.
- The "IP", "MAC", and "T" options allow you to display the IP addresses, MAC addresses, and packet types, respectively, on the nodes or links.

5.Animation:

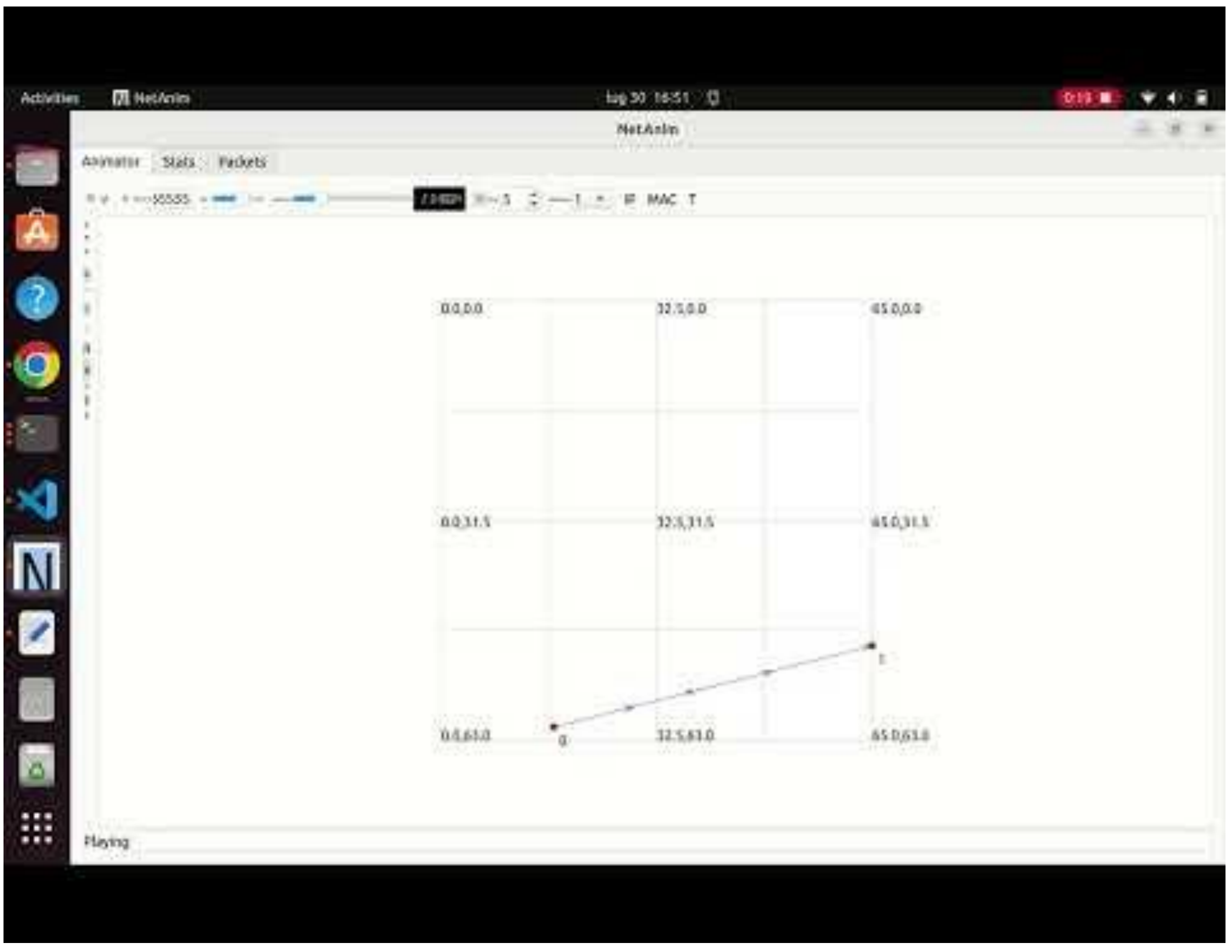
- When you click "Play," the simulation will animate the transfer of packets from the client to the server, showing the data flow over the established connection.

6. Simulation Time:

- At the bottom or top of the interface, you may see a simulation time indicator. This shows the current time in the simulation.

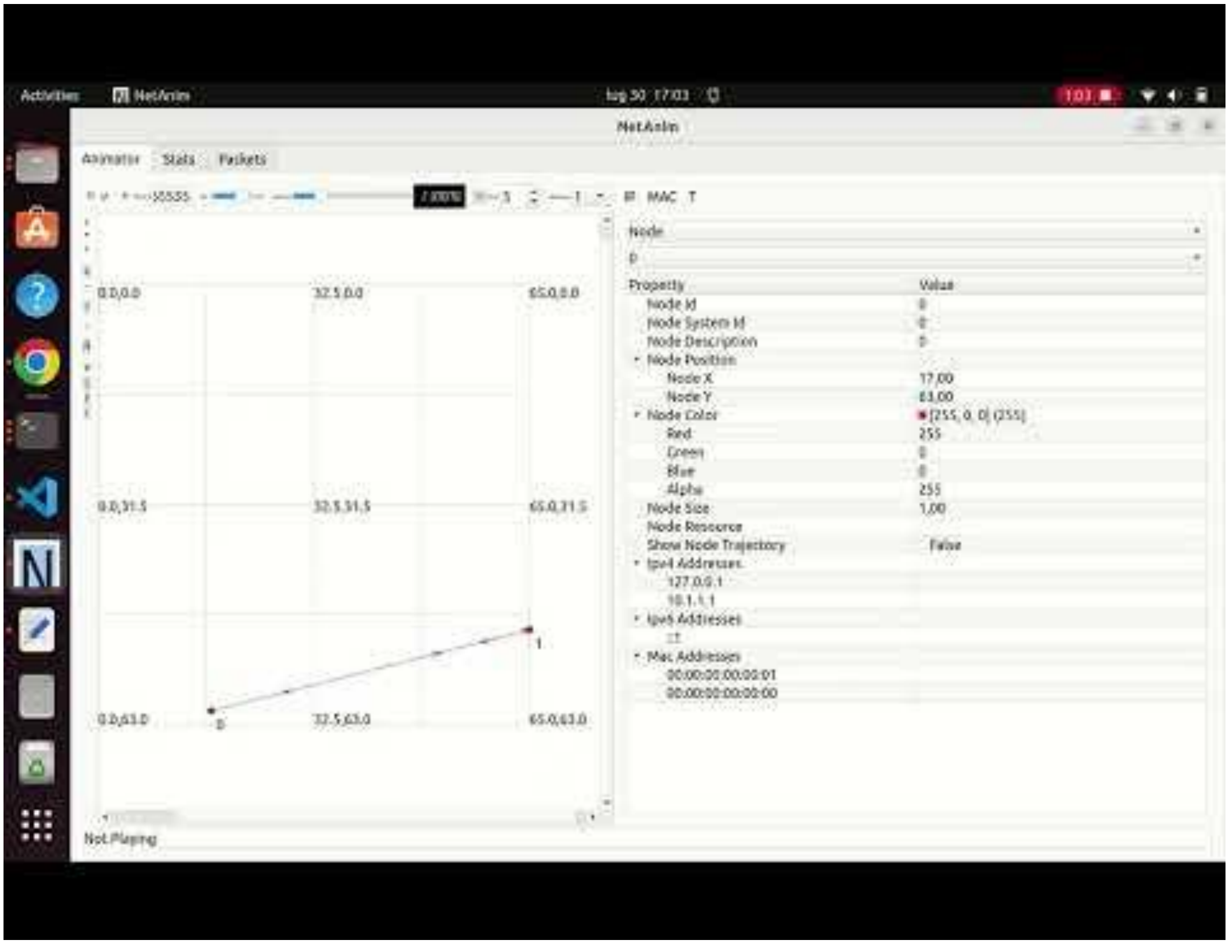
To better understand the simulation, you can:

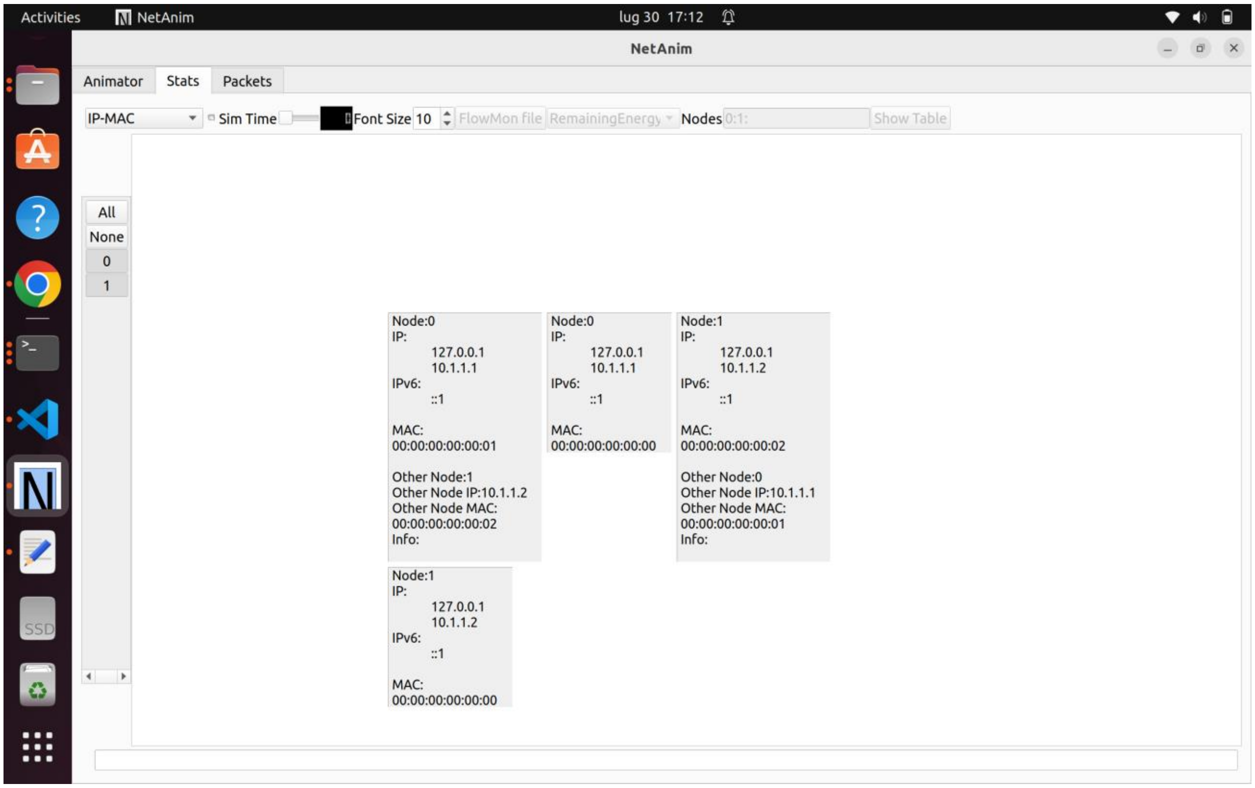
- ❖ **Play the animation** to see how data flows between the client and server over time.



- ❖ **Zoom in/out** and **pan around** the grid to get a better view of the nodes and links.
- ❖ **Check packet details** by enabling different display options (like IP, MAC, etc.).

This visualization helps you analyze the network performance, such as the speed and volume of data transfer, as well as any potential issues in the network setup





The screenshot you've provided shows the IP and MAC address details of the nodes in the network simulation within NetAnim. Here's a breakdown of what each section indicates:

1. **Node Information:**
   - **Node 0** and **Node 1** are the two nodes in your simulation. The details for each node are shown in separate boxes.
2. **IP Addresses:**
   - **127.0.0.1**: This is the loopback IP address, commonly used to refer to the local host. It's typically present by default.
   - **10.1.1.1** (for Node 0) and **10.1.1.2** (for Node 1): These are the assigned IP addresses for the respective nodes on the network. They represent the actual network addresses used for communication between the nodes.
3. **IPv6 Address:**
   - **::1**: This is the IPv6 loopback address, equivalent to 127.0.0.1 in IPv4. It is also present by default on all devices.
4. **MAC Addresses:**
   - Each node has a unique MAC (Media Access Control) address, which serves as a hardware identifier. In this simulation, the MAC addresses are shown as 00:00:00:00:00:01 for Node 0 and 00:00:00:00:00:02 for Node 1.
5. **Other Node Information:**

- This section displays details of the other node that the current node is connected to. For instance, Node 0 lists information about Node 1 and vice versa.

Additional Details:

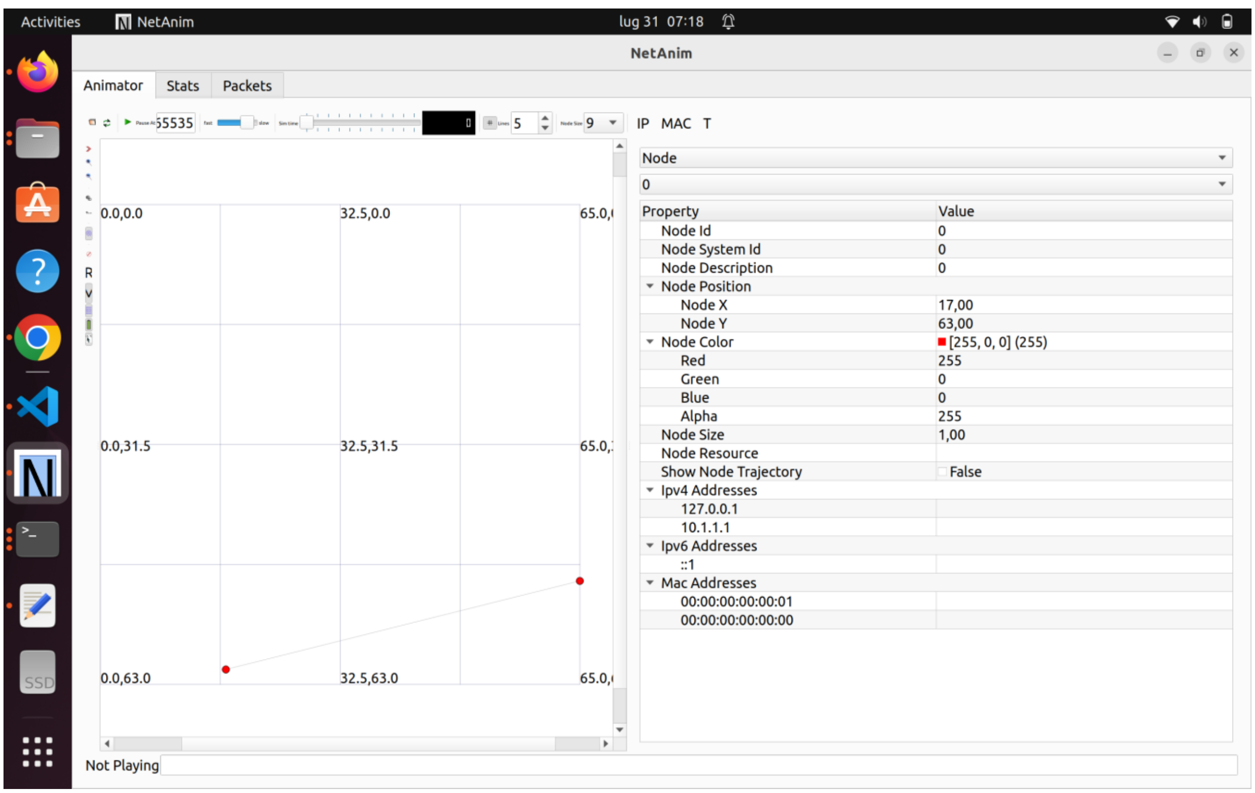
- The boxes also display other metadata such as the connected node's IP and MAC addresses.

How to Use this Information

This information is useful for identifying and confirming the setup of your network simulation. It shows:

- The assignment of IP addresses and MAC addresses for each node.
- The relationships and connections between the nodes.

In this simulation, you can verify that Node 0 is set as the client with IP 10.1.1.1 and Node 1 as the server with IP 10.1.1.2. The correct IP and MAC addresses indicate a proper network setup, allowing for the intended data transfer simulation.



This figure shows additional details about Node 0 in the network simulation within NetAnim. Let's break down the information displayed:

Node Details Panel

- **Node:** The dropdown shows the node selected, which is "Node 0" in this case.

Property and Value Table

4. **Node Id:**
    - The unique identifier for this node in the simulation, which is 0.
5. **Node System Id:**
    - Another identifier, usually set to 0 if not specifically assigned.
6. **Node Description:**
    - No description is provided, hence set to 0.
7. **Node Position:**
    - **Node X:** The X-coordinate of the node's position on the simulation grid, which is 17.00.
    - **Node Y:** The Y-coordinate of the node's position on the simulation grid, which is 63.00.
8. **Node Color:**
    - The color of the node as represented in the simulation. The color is specified in RGBA format:
        - **Red:** 255 (Maximum value, so the red component is fully saturated)
        - **Green:** 0 (No green component)
        - **Blue:** 0 (No blue component)
        - **Alpha:** 255 (Fully opaque)
    - The color represented is red, as indicated by [255, 0, 0].
9. **Node Size:**
    - The size of the node as displayed, set to 1.00.
10. **Node Resource:**
    - No specific resource is associated, so it's not detailed here.
11. **Show Node Trajectory:**
    - Indicates whether the movement path of the node will be displayed. It is set to False, so no trajectory will be shown.

IP and MAC Addresses

- **IPv4 Addresses:**
    - **127.0.0.1:** The loopback IP address.
    - **10.1.1.1:** The network IP address of the node.
- **IPv6 Addresses:**

    - **::1:** The loopback IPv6 address.
- **MAC Addresses:**
    - **00:00:00:00:00:01:** The MAC address of the node's interface connected to the network.
    - **00:00:00:00:00:00:** Another interface, possibly an uninitialized or default MAC address.

Visualization

- The map on the left side shows the physical position of the nodes. The red dot represents Node 0 at coordinates (17, 63). The connections and other details can be visually inspected, such as the link to other nodes.

Interpretation

This information is useful to understand the configuration and positioning of the nodes in the simulation. The detailed node properties help in debugging, analyzing, and modifying the network simulation setup. For instance, the node's color and size make it visually distinguishable, while the IP and MAC addresses are crucial for network layer configurations and traffic analysis.

Secondo scenario for NS3 simulation

To simulate the behavior of the TCP protocol between two nodes over a communication channel with specific delay and capacity using ns-3, you can follow these steps:

1. Create Nodes and Channel

Start by creating two nodes and a communication channel between them. Set the channel characteristics, such as delay and capacity (bandwidth).

2. Configure the TCP Protocol

Configure the TCP parameters, such as the version of the TCP protocol (e.g., NewReno, Tahoe, etc.).

3. Configure Source and Destination Applications

Set up an application that generates TCP traffic on one node and an application that receives it on another node.

4. Set Simulation Parameters

Define the simulation parameters, such as duration and start time of the applications.

5. Run the Simulation and Analyze Results

Run the simulation and use analysis tools like NetAnim or gnuplot to visualize the results.

Here's a basic example script for ns-3 by adding NetAnim Module:

```
#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"
#include "ns3/netanim-module.h"

using namespace ns3;

int main (int argc, char *argv[])
{
// Create two nodes
NodeContainer nodes;
nodes.Create (2);

// Configure a point-to-point channel with specific delay and capacity
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

// Install devices on the nodes
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);

// Install the Internet stack (IPv4)
InternetStackHelper stack;
stack.Install (nodes);
```

```cpp
// Assign IP addresses
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign (devices);

// Configure a TCP server application on node B
uint16_t port = 9; // Port for the server
Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", localAddress);
ApplicationContainer serverApp = packetSinkHelper.Install (nodes.Get (1));
serverApp.Start (Seconds (1.0));
serverApp.Stop (Seconds (10.0));

// Configure a TCP client application on node A
Address remoteAddress (InetSocketAddress (interfaces.GetAddress (1), port));
OnOffHelper onOffHelper ("ns3::TcpSocketFactory", remoteAddress);
onOffHelper.SetAttribute ("DataRate", StringValue ("1Mbps"));
onOffHelper.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApp = onOffHelper.Install (nodes.Get (0));
clientApp.Start (Seconds (2.0));
clientApp.Stop (Seconds (10.0));
// Set up NetAnim
AnimationInterface anim("animation.xml");
// Run the simulation
Simulator::Run ();
Simulator::Destroy ();

return 0;
}
```

1. Include the NetAnim Module:

   ```cpp
   #include "ns3/netanim-module.h"
   ```

   This line includes the necessary headers for using NetAnim.

2. AnimationInterface Instance:

```cpp
AnimationInterface anim("animation.xml");
```

This line creates an `AnimationInterface` object, which sets up the animation output to an XML file named `animation.xml`. You can open this file later with NetAnim to visualize the simulation.

Code Explanation

1.Creating Nodes and Channel:

- nodes.Create (2); creates two nodes.
- pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps")); and pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms")); set the channel's capacity and delay.

2.Installing Stacks and Interfaces:

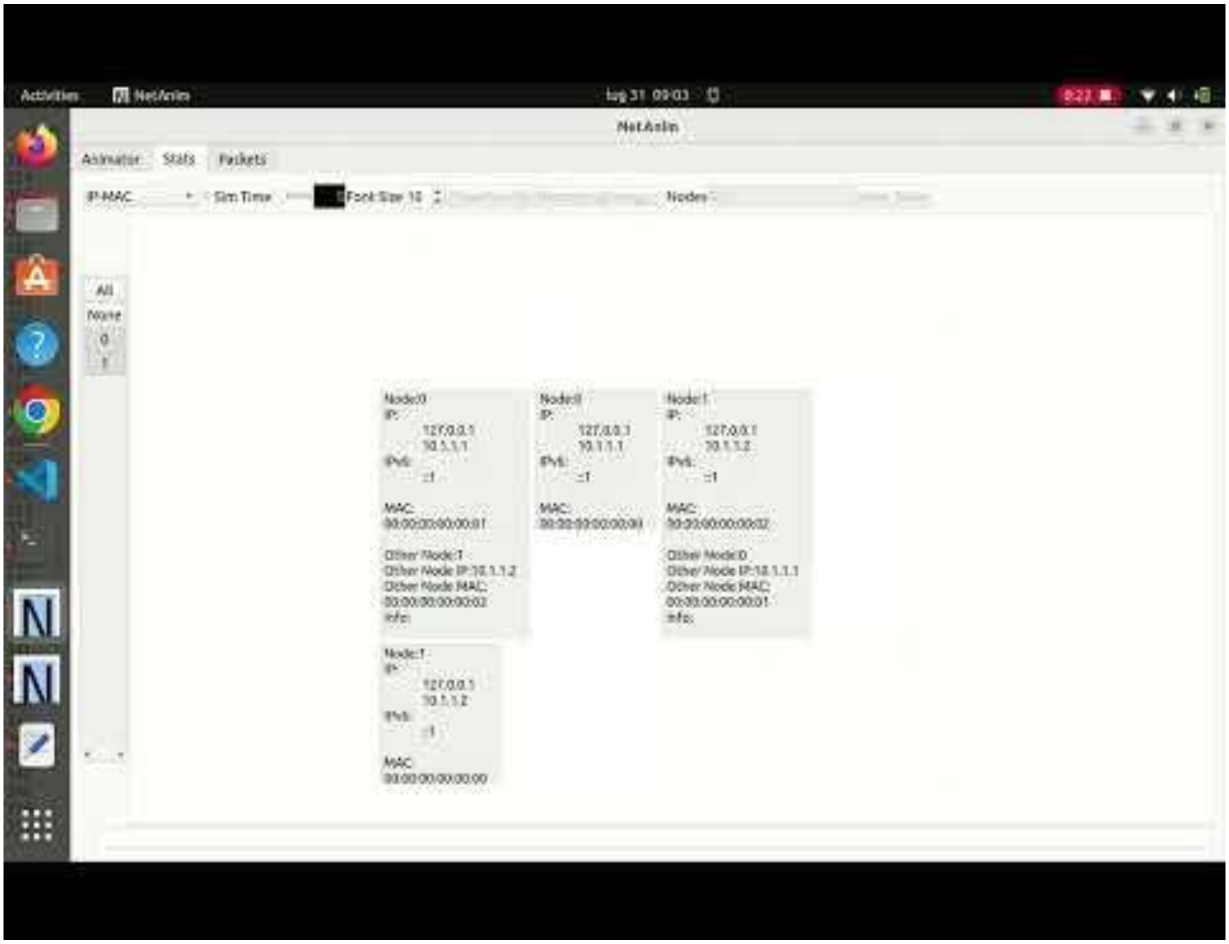
- stack.Install (nodes); installs the Internet stack on the nodes.
- address.Assign (devices); assigns the IP addresses.

3.Configuring Applications:

- The TCP server is configured on node B (nodes.Get (1)) using PacketSinkHelper.
- The TCP client is configured on node A (nodes.Get (0)) using OnOffHelper.

4.Running the Simulation:

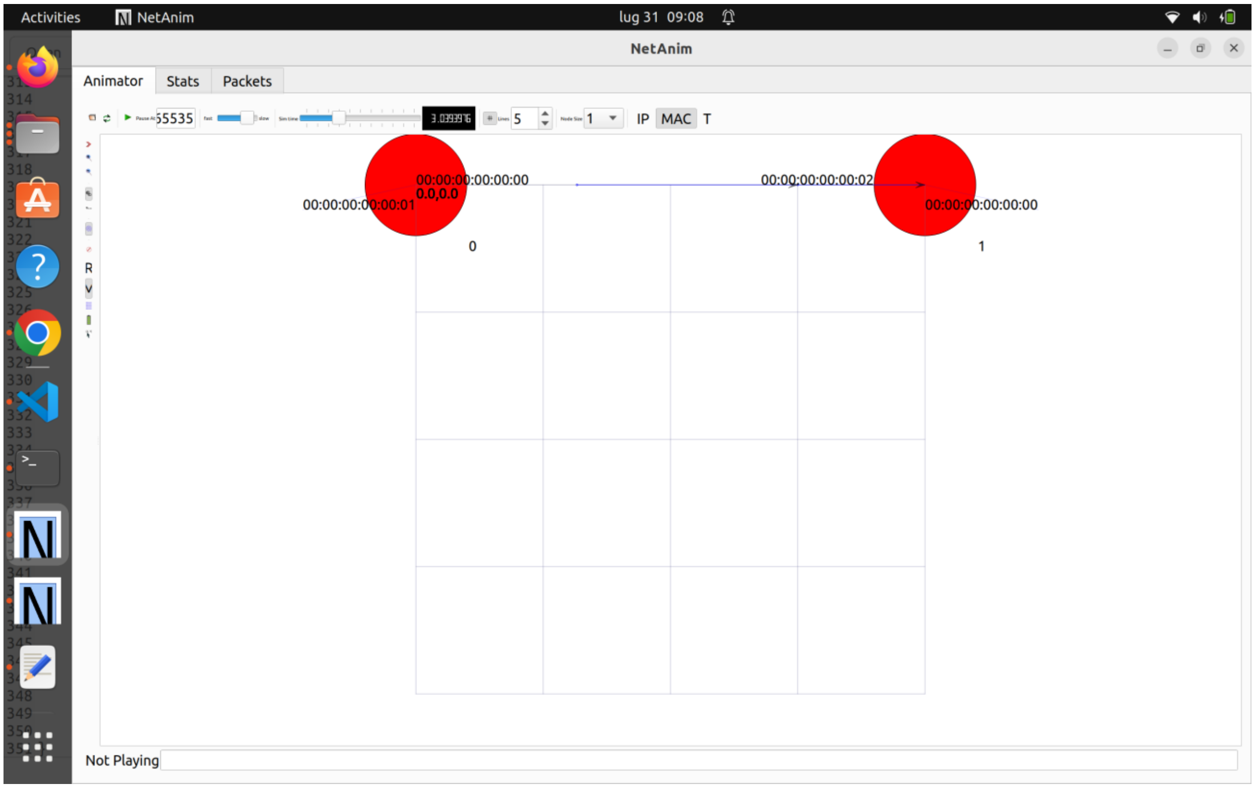
- The simulation is started with Simulator::Run (); and then destroyed with Simulator::Destroy ();.



https://www.youtube.com/watch?v=XZ9nYZUPJ3Y

- **Node 0** is acting as the **client**. It has the TCP client application installed, which sends data to the server.
- **Node 1** is acting as the **server**. It has the TCP server application (PacketSink) installed, which receives data from the client.

This configuration corresponds to the standard client-server model, where the client initiates a connection and sends data to the server, and the server listens for incoming connections and processes the data it receives.



In the provided NetAnim visualization, the elements displayed represent the nodes and the network traffic between them. Here's a breakdown of what you're seeing:

1. **Nodes**:
   - There are two red circles representing the two nodes in your simulation. Each node has a unique identifier (0 and 1).
   - The labels next to each node (e.g., 00:00:00:00:00:00) are MAC addresses, which uniquely identify the network interfaces of the nodes.
   - The coordinates (e.g., 0,0,0,0 and 0,1) indicate the position of each node. This representation is more meaningful when the nodes have mobility, but in this case, they are stationary.
2. **Connection**:
   - The line connecting the two nodes represents the point-to-point link between them. It indicates the data flow between the nodes.
   - The arrow indicates the direction of traffic from node 0 to node 1.
3. **Traffic**:
   - The color and size of the arrow can indicate the amount of data being transferred or the activity level. However, in this screenshot, it appears to be a basic representation.
4. **Simulation Controls**:

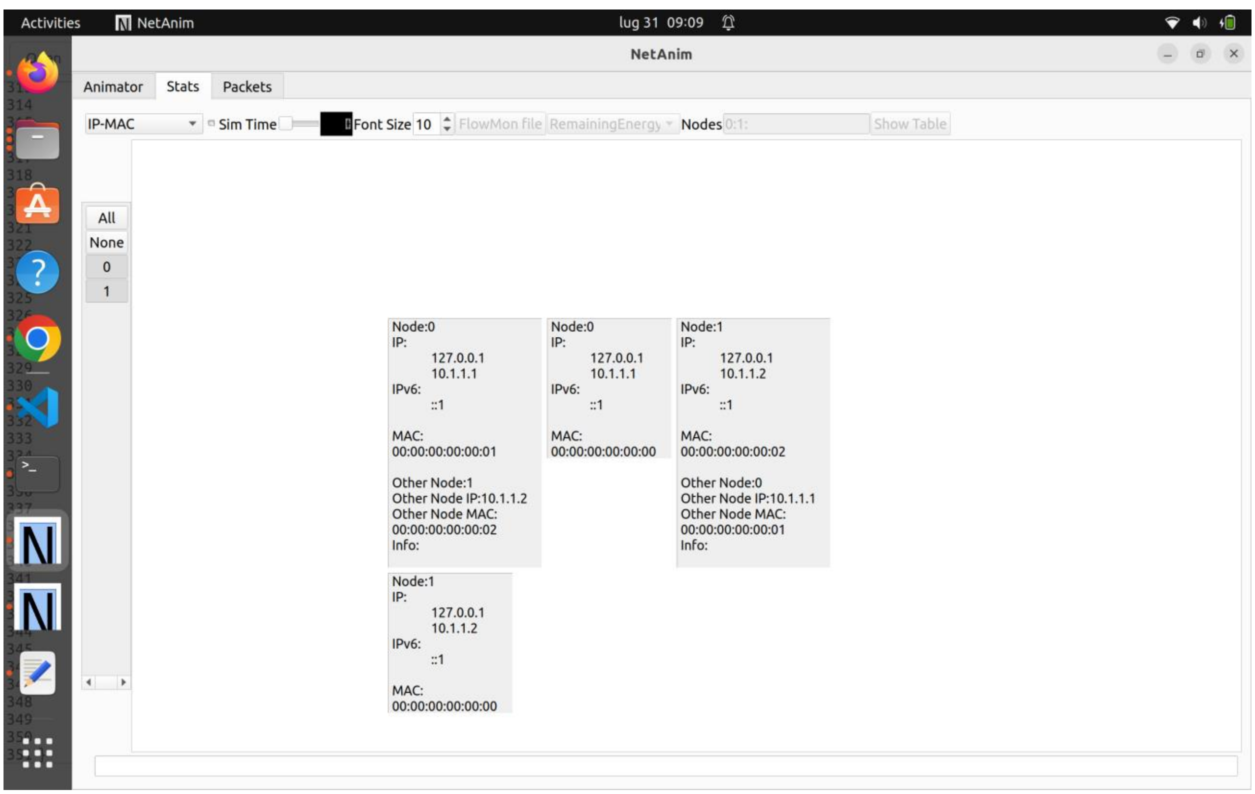
- At the top, there are controls to play, pause, and adjust the speed of the simulation. You can use these controls to observe the network activity over time.

5.Node Details:

- On the top right, you can switch between different views, such as IP, MAC, and other attributes of the nodes.

Key Points:

- The large red circles are nodes in the simulation.
- The line between them represents the data flow over a point-to-point connection.
- The nodes do not have mobility models set, so they remain stationary.



In this NetAnim visualization, the information panel for the nodes is displayed. Here's a breakdown of the details shown:

Nodes Overview

1.Node 0:

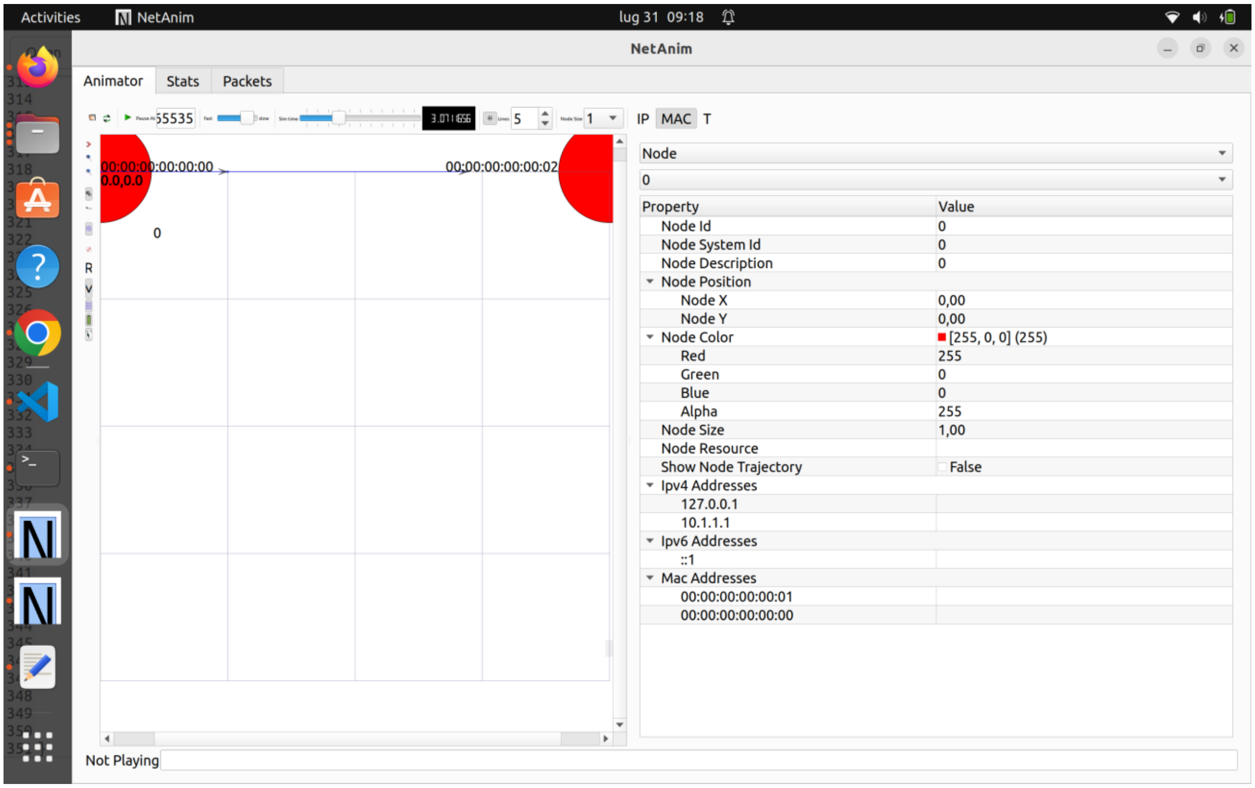
- **IP Addresses**:
    - **127.0.0.1**: This is the loopback IP address, which is a standard address used for testing the network interface.
    - **10.1.1.1**: This is the IP address assigned to Node 0 in the simulation network.
- **IPv6 Address**: ::1, which is the loopback address in IPv6 (similar to 127.0.0.1 in IPv4).
- **MAC Address**: 00:00:00:00:00:01, which uniquely identifies Node 0's network interface.
- **Other Node**:
    - **Node 1**: Shows the IP (10.1.1.2) and MAC (00:00:00:00:00:02) addresses of the node connected to Node 0.

2. **Node 1**:
    - **IP Addresses**:
        - **127.0.0.1**: The loopback IP address.
        - **10.1.1.2**: The IP address assigned to Node 1 in the simulation network.
    - **IPv6 Address**: ::1.
    - **MAC Address**: 00:00:00:00:00:02.
    - **Other Node**:
        - **Node 0**: Shows the IP (10.1.1.1) and MAC (00:00:00:00:00:01) addresses of the node connected to Node 1.

Additional Details

- The panel displays IP and MAC addresses for both IPv4 and IPv6.
- The **Other Node** section for each node lists the connected node's IP and MAC addresses. This information helps in identifying the point-to-point connections between nodes.
- The data shows that the nodes are connected as expected with the configured addresses.

This information helps in understanding the network topology and the configuration of each node's network interfaces. It confirms that the nodes are properly connected and have been assigned the correct IP addresses, as specified in the simulation script.



Left Panel (Network Animation)

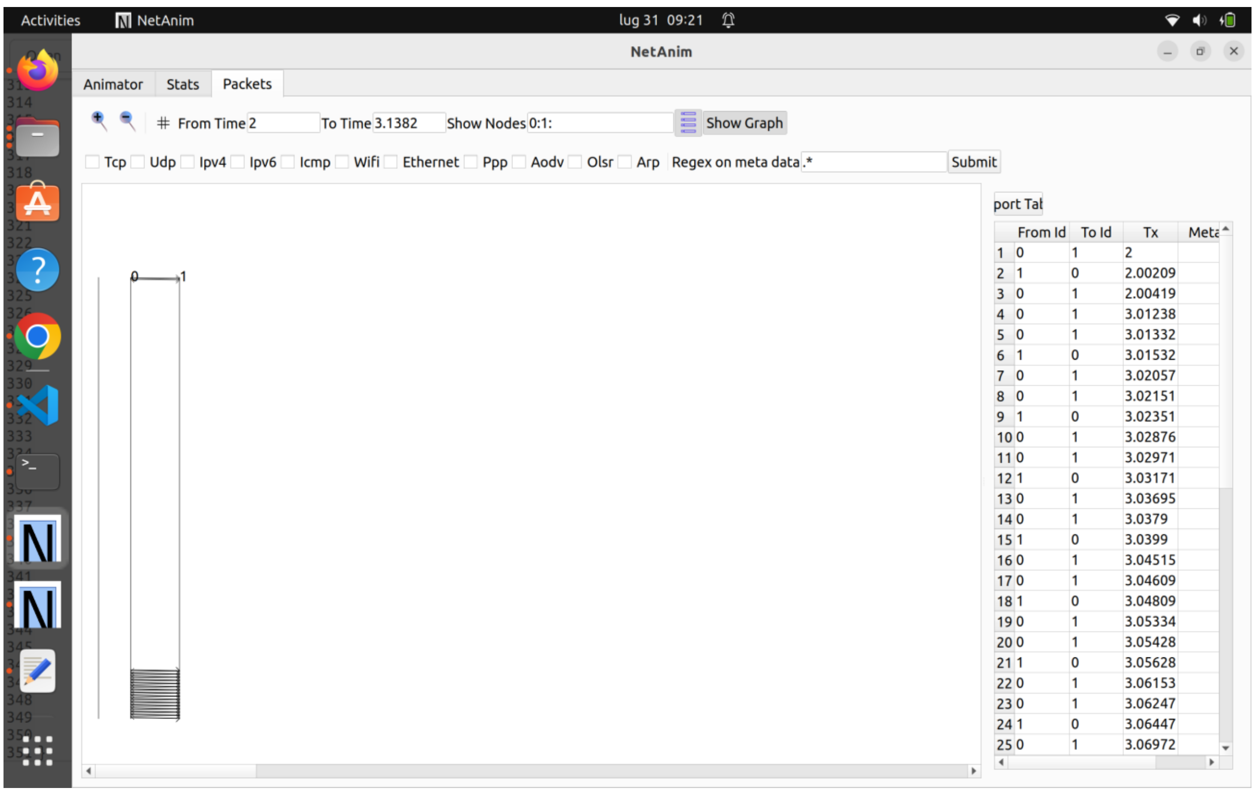
- **Red Circles**: These represent the nodes in the network. The large size indicates the scale or zoom level in the animation. Node 0 is on the left, and Node 1 is on the right.
- **Arrow**: This represents the data flow from Node 0 to Node 1, indicating that data is being sent from the client to the server.

Right Panel (Node Properties)

- **Node**: 0
  - **Node Id**: 0
  - **Node System Id**: 0 (usually used internally by ns-3)
  - **Node Description**: 0 (not specified in this simulation)
  - **Node Position**: (0.00, 0.00) indicating that Node 0 is at the origin point.
  - **Node Color**: [255, 0, 0] (255) indicates the node's color is red, which corresponds to the RGB color code.
  - **Node Size**: 1.00 (the scale used in the animation)
  - **Show Node Trajectory**: False indicates that the trajectory or movement path is not being displayed (because it's not set, the nodes are stationary).
  - **IPv4 Addresses**:

        - 127.0.0.1 (local loopback address)
        - 10.1.1.1 (assigned by the simulation for communication with Node 1)
    - **IPv6 Addresses**: ::1 (loopback address for IPv6)
    - **MAC Addresses**:
        - 00:00:00:00:00:01 (the unique identifier for the network interface of Node 0)
        - 00:00:00:00:00:00 (another MAC address, which may represent a default or secondary interface)

This visualization indicates the configuration and communication setup in the network simulation. Node 0 acts as the client, sending data to Node 1 (the server) over a point-to-point link, with properties defined in the simulation code. The color coding and sizes are used for easy identification and visualization in the animation tool



shows the **Packets** tab in NetAnim, which provides detailed information about packet transmissions between nodes in the simulation. Here's a breakdown of the key elements:

Left Panel (Packet Flow Diagram)

- **Vertical Lines**: Represent the timeline of packet transmissions. The vertical line on the left corresponds to Node 0, and the line on the right corresponds to Node 1.

- **Horizontal Lines with Arrows**: Indicate the flow of packets between nodes. The direction of the arrow shows the direction of the packet transmission. In this case, the arrows show packets being sent from Node 0 to Node 1 and back, indicating two-way communication.

Right Panel (Packet Details Table)

- **Columns**:
  - **From Id**: The ID of the source node (0 or 1).
  - **To Id**: The ID of the destination node (1 or 0).
  - **Tx**: The time at which the packet was transmitted.
  - **Meta**: This column typically contains metadata about the packet, but the specific content isn't shown in this screenshot.
- **Rows**: Each row represents a packet transmission event, detailing the source and destination nodes and the transmission time.

Timeline and Filtering

- **From Time/To Time**: Allows you to filter the displayed packet events within a specific time range (2 to 3.1382 seconds in this case).
- **Show Nodes**: Specifies which nodes' interactions are displayed (Node 0 and Node 1).
- **Check Boxes**: Allow filtering by protocol type (Tcp, Udp, Ipv4, etc.). In this screenshot, only the Tcp protocol is selected, meaning only TCP packet events are displayed.

Explanation

The packet flow diagram and the table provide a detailed view of the packet exchange between Node 0 and Node 1. This visualization helps understand the timing and flow of data packets in the simulation, showing that data packets are being sent back and forth between the client (Node 0) and the server (Node 1). The exact timings can be analyzed to study the network's behavior, such as latency, throughput, and packet delivery.

Summarization

Question

ok quindi simulare tcp del bando e del canale simulare nel ns3 via mezzo di canale di comunicazione con ritardo e capacita del canale .. abbiamo fatto tutto li sopra? con tcp file transfer e tcp simulation??

Answer Sì, con i due script "tcp-file-transfer" e "tcp-simulation", hai già simulato una connessione TCP tra due nodi tramite un canale di comunicazione con ritardo e capacità definiti.

Dettagli delle Simulazioni

12. **Configurazione del Canale di Comunicazione**:
    - In entrambi gli script, è stato utilizzato un PointToPointHelper per definire il canale tra i due nodi. Sono stati impostati attributi come DataRate per definire la capacità del canale e Delay per simulare il ritardo del canale.
13. **Applicazioni TCP**:
    - **tcp-file-transfer**: Questo script può aver simulato il trasferimento di un file da un nodo all'altro utilizzando il protocollo TCP. Può includere l'uso di applicazioni come PacketSink per il server e OnOffHelper per il client, simulando la trasmissione di dati.
    - **tcp-simulation**: Probabilmente simula una configurazione simile, ma con variazioni nei parametri o nel comportamento delle applicazioni per studiare diversi scenari di rete.
14. **Animazione e Visualizzazione**:
    - Con l'uso di NetAnim, hai visualizzato l'animazione delle simulazioni, osservando il flusso dei pacchetti tra i nodi. L'integrazione di AnimationInterface ha permesso la generazione di file di animazione (animation.xml) per visualizzare i risultati.

Considerazioni Finali

Hai simulato correttamente un ambiente di rete TCP su ns-3, considerando aspetti come:

- Ritardo di propagazione nel canale.
- Capacità del canale.
- Flusso di pacchetti tra client e server.

-Tcp file transfer

1. Ritardo di propagazione nel canale:

- Nella simulazione, il ritardo di propagazione è stato configurato utilizzando l'attributo `"Delay"` del `PointToPointHelper`. Per esempio, se il ritardo è stato impostato su `"2ms"`, significa che ogni pacchetto impiega 2 millisecondi per viaggiare dal client al server attraverso il canale di comunicazione.

2. Capacità del canale:

- La capacità del canale è stata configurata tramite l'attributo `"DataRate"` del `PointToPointHelper`. Ad esempio, se è stata impostata su `"5Mbps"`, ciò significa che il canale può trasmettere dati fino a un massimo di 5 Megabit al secondo. Questa capacità limita la quantità di dati che possono essere inviati in un dato momento, influenzando così la velocità di trasferimento effettiva.

3. Flusso di pacchetti tra client e server:

- Durante la simulazione, il flusso di pacchetti è stato osservato utilizzando applicazioni come `OnOffHelper` per il client e `PacketSink` per il server. Il `OnOffHelper` ha generato pacchetti TCP e li ha inviati al server, mentre il `PacketSink` ha ricevuto questi pacchetti. Il flusso di pacchetti è stato visualizzato utilizzando NetAnim, mostrando il trasferimento dei pacchetti dal client al server e viceversa. La visualizzazione includeva il numero di pacchetti trasmessi e ricevuti, nonché i tempi di trasmissione.

TCP simulation

- **Ritardo di propagazione nel canale**:
- Nella simulazione TCP, il ritardo di propagazione è stato configurato tramite l'attributo "Delay" del PointToPointHelper. Ad esempio, se il valore è stato impostato su "2ms", ogni pacchetto inviato dal client al server (e viceversa) impiega 2 millisecondi per attraversare il canale di comunicazione. Questo ritardo rappresenta il tempo necessario affinché un segnale viaggi da un'estremità all'altra del canale.
- **Capacità del canale**:

- La capacità del canale è stata impostata utilizzando l'attributo "DataRate" del PointToPointHelper. Ad esempio, se è stato specificato "5Mbps", il canale può supportare un flusso di dati massimo di 5 Megabit al secondo. Questa impostazione determina la quantità massima di dati che possono essere trasmessi attraverso il canale in un dato momento, influenzando la larghezza di banda disponibile per la comunicazione tra il client e il server.
- **Flusso di pacchetti tra client e server**:
- Il flusso di pacchetti tra client e server è stato gestito attraverso un'applicazione client e un'applicazione server. Il client, configurato con un OnOffHelper, ha generato pacchetti TCP e li ha inviati al server a intervalli specifici, mentre il server, configurato con un PacketSinkHelper, ha ricevuto questi pacchetti. NetAnim è stato utilizzato per visualizzare questo flusso, mostrando i pacchetti in transito e la loro ricezione. La simulazione ha registrato l'invio e la ricezione dei pacchetti, inclusi i tempi di trasmissione e ricezione, consentendo di analizzare il throughput e altri parametri di rete.