CSE 421

Lab 2: Introduction to Packet Analysis and Network Simulation

ID: ____________________

Introduction:

In real life, network monitoring and packet analysis is of great importance to observe the network state and identify any anomalous network event. Wireshark is a tool that can be used to monitor our network and also analysis traces of past network events. In this lab, we will get familiar with the wireshark interface and learn basic packet analysis.

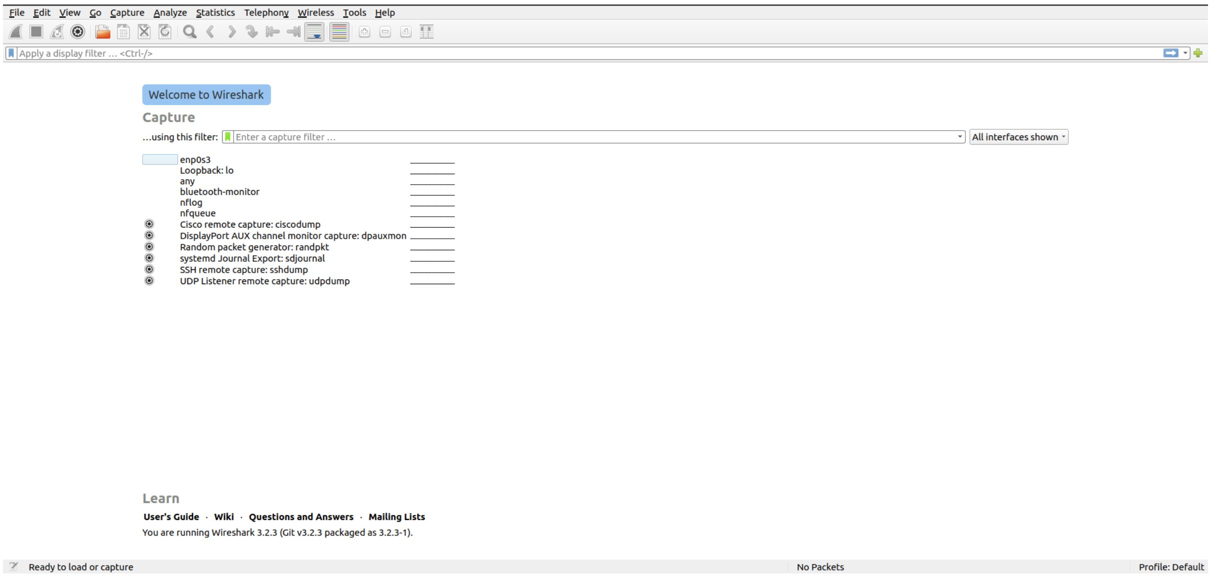
In networking research and protocol testing, simulating a network is one of the core works. We can try out existing networking protocols and measure their performance. Also, we can devise our own modified protocol and test its performance against the existing ones. In this lab, we will also get familiar with an open source network simulator, ns3. We will learn how to install the simulator and run a basic simulation.

Objectives:

1. Getting familiar with wireshark
    a. Learn about basic wireshark interface
    b. Basic filtering of packets
    c. Explore a simple **HTTP** stream
2. Getting familiar with network simulation
    a. Installing and running ns3
    b. Understanding a simple simulation script
    c. Visualizing our simulation
    d. Collecting network traces and observing the events

Task 1: Using Wireshark

1. Step 1: Download and install wireshark
    a. Go to https://www.wireshark.org/
    b. Download and install the tool for the operating system you are using
2. Open wireshark if it is installed
    a. You will see a window like the following



Apply a display filter … <Ctrl-/>

Capture

…using this filter: Enter a capture filter …

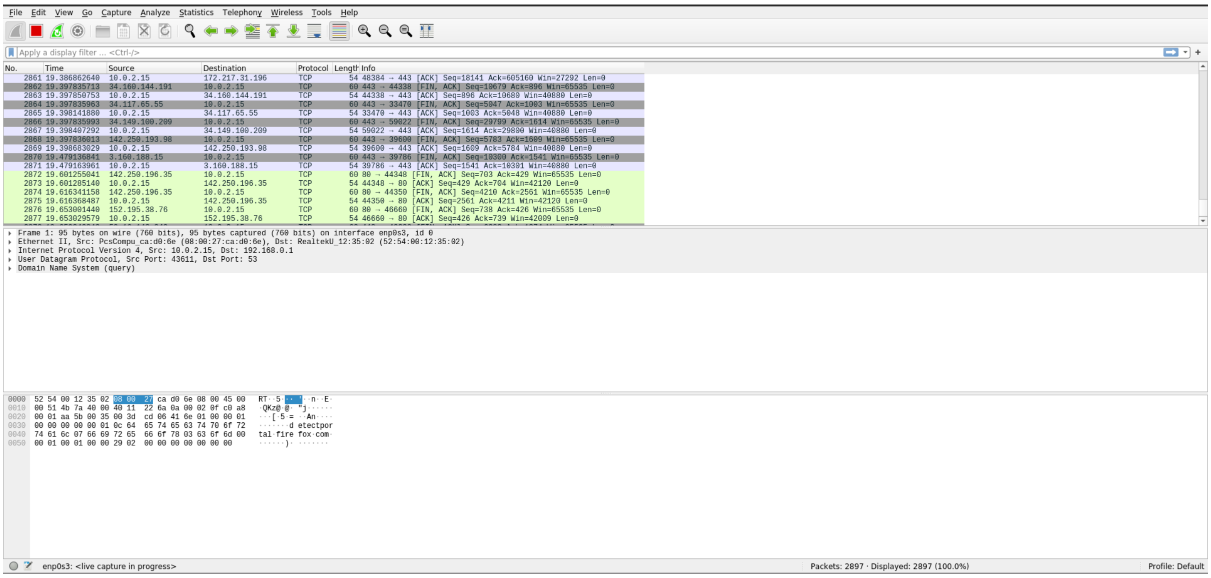
enp0s3
Loopback: lo
any
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

Learn

You are running Wireshark 3.2.3 (Git v3.2.3 packaged as 3.2.3-1).

Ready to load or capture · No Packets · Profile: Default

b. You will see a list of network interfaces in your device. We will select our ethernet interface/ wifi interface, whatever we are using. In our case, we will select **enp0s3** and start capturing. To start capturing, we can double click the intended interface on which we want to capture the packets. We will see an interface like the following:



```
No.   Time           Source            Destination       Protocol Length Info
 2861 19.386862640   10.0.2.15         172.217.31.196    TCP      54 48384 → 443 [ACK] Seq=18141 Ack=605160 Win=27292 Len=0
 2862 19.397835713   34.160.144.191    10.0.2.15         TCP      60 443 → 44338 [FIN, ACK] Seq=10679 Ack=896 Win=65535 Len=0
 2863 19.397850753   10.0.2.15         34.160.144.191    TCP      54 44338 → 443 [ACK] Seq=896 Ack=10680 Win=40880 Len=0
 2864 19.397835963   34.117.65.55      10.0.2.15         TCP      60 443 → 33470 [FIN, ACK] Seq=5047 Ack=1003 Win=65535 Len=0
 2865 19.398141880   10.0.2.15         34.117.65.55      TCP      54 33470 → 443 [ACK] Seq=1003 Ack=5048 Win=40880 Len=0
 2866 19.397835993   34.149.100.209    10.0.2.15         TCP      60 443 → 59022 [FIN, ACK] Seq=29799 Ack=1614 Win=65535 Len=0
 2867 19.398407292   10.0.2.15         34.149.100.209    TCP      54 59022 → 443 [ACK] Seq=1614 Ack=29800 Win=40880 Len=0
 2868 19.397836013   142.250.193.98    10.0.2.15         TCP      60 443 → 39600 [FIN, ACK] Seq=5783 Ack=1609 Win=65535 Len=0
 2869 19.398683029   10.0.2.15         142.250.193.98    TCP      54 39600 → 443 [ACK] Seq=1609 Ack=5784 Win=40880 Len=0
 2870 19.479136841   3.160.188.15      10.0.2.15         TCP      60 443 → 39786 [FIN, ACK] Seq=10300 Ack=1541 Win=65535 Len=0
 2871 19.479163961   10.0.2.15         3.160.188.15      TCP      54 39786 → 443 [ACK] Seq=1541 Ack=10301 Win=40880 Len=0
 2872 19.601255041   142.250.196.35    10.0.2.15         TCP      60 80 → 44348 [FIN, ACK] Seq=703 Ack=429 Win=65535 Len=0
 2873 19.601285140   10.0.2.15         142.250.196.35    TCP      54 44348 → 80 [ACK] Seq=429 Ack=704 Win=42120 Len=0
 2874 19.616341158   142.250.196.35    10.0.2.15         TCP      60 80 → 44350 [FIN, ACK] Seq=4210 Ack=2561 Win=65535 Len=0
 2875 19.616368487   10.0.2.15         142.250.196.35    TCP      54 44350 → 80 [ACK] Seq=2561 Ack=4211 Win=42120 Len=0
 2876 19.653001440   152.195.38.76     10.0.2.15         TCP      60 80 → 46660 [FIN, ACK] Seq=738 Ack=426 Win=65535 Len=0
 2877 19.653029579   10.0.2.15         152.195.38.76     TCP      54 46660 → 80 [ACK] Seq=426 Ack=739 Win=42009 Len=0
```

```
▸ Frame 1: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface enp0s3, id 0
▸ Ethernet II, Src: PcsCompu_ca:d0:6e (08:00:27:ca:d0:6e), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.0.1
▸ User Datagram Protocol, Src Port: 43611, Dst Port: 53
▸ Domain Name System (query)
```

```
0000  52 54 00 12 35 02 08 00  27 ca d0 6e 08 00 45 00   RT··5···'··n··E·
0010  00 51 4b 7a 40 00 40 11  22 6a 0a 00 02 0f c0 a8   ·QKz@·@·"j······
0020  00 01 aa 5b 00 35 00 3d  cd 06 41 6e 01 00 00 01   ···[·5·=··An····
0030  00 00 00 00 00 01 0c 64  65 74 65 63 74 70 6f 72   ·······d etectpor
0040  74 61 6c 07 66 69 72 65  66 6f 78 03 63 6f 6d 00   tal·fire fox·com·
0050  00 01 00 01 00 00 29 02  00 00 00 00 00 00 00      ······)· ·······
```

enp0s3: <live capture in progress>   Packets: 2897 · Displayed: 2897 (100.0%)   Profile: Default

Here, there are three sub windows. The top sub window shows us the list of packets. The middle sub window shows us the headers and data of each layer of any selected packet. The bottom sub window shows us the raw bytes that each packet contains and also the decoded form of it.

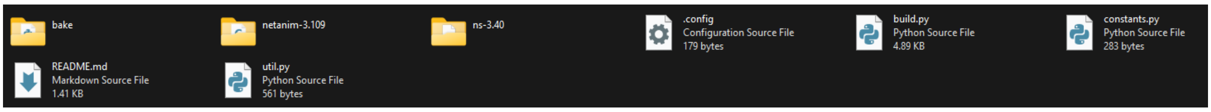
c. We can observe that there are a lot of things going on. That means, our device is interacting through the network interface. We can focus on things by filtering out packets that are of our interest. For example, if we want to filter out http data, we can type ‘http’ on the text box on top and press enter. Only the http packets will be shown and other packets will be filtered out.

Home Task:

Visit any http website while monitoring the network traffic. Now filter only the http packets. Observe the headers of any http request and response packet and explain how the layers are operating in this case.

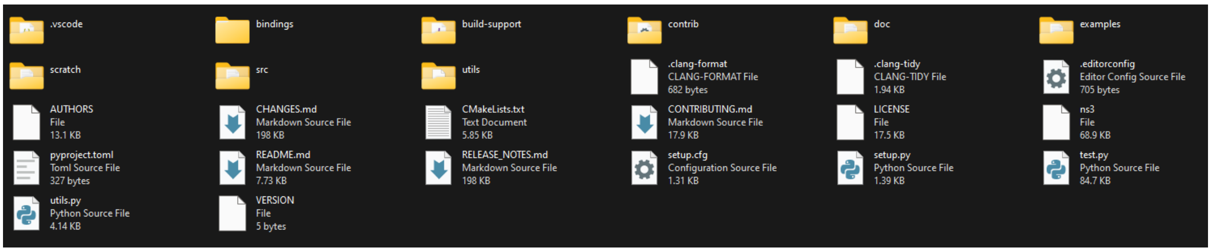
Task 2: Using NS-3

1. Download NS3
    a. Go to https://www.nsnam.org/releases/ns-3-40/ and download the latest version of ns3
    b. Extract the contents and you will see the contents like this:



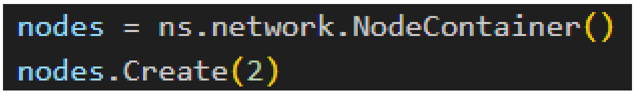
bake	netanim-3.109	ns-3.40	.config Configuration Source File 179 bytes	build.py Python Source File 4.89 KB	constants.py Python Source File 283 bytes
README.md Markdown Source File 1.41 KB	util.py Python Source File 561 bytes				

    c. Move inside ns-3.40 directory. The contents will look like this:



.vscode	bindings	build-support	contrib	doc	examples
scratch	src	utils	.clang-format CLANG-FORMAT File 682 bytes	.clang-tidy CLANG-TIDY File 1.94 KB	.editorconfig Editor Config Source File 705 bytes
AUTHORS File 13.1 KB	CHANGES.md Markdown Source File 198 KB	CMakeLists.txt Text Document 5.85 KB	CONTRIBUTING.md Markdown Source File 17.9 KB	LICENSE File 17.5 KB	ns3 File 68.9 KB
pyproject.toml Toml Source File 327 bytes	README.md Markdown Source File 7.73 KB	RELEASE_NOTES.md Markdown Source File 198 KB	setup.cfg Configuration Source File 1.31 KB	setup.py Python Source File 1.39 KB	test.py Python Source File 84.7 KB
utils.py Python Source File 4.14 KB	VERSION File 5 bytes				

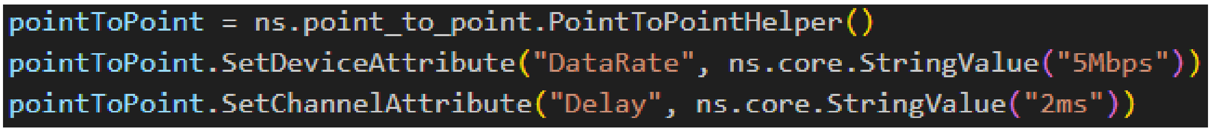
2. Install the necessary packages
    a. **sudo apt install g++ python3 cmake ninja-build git**
        i. **If you find any error giving this command, update and upgrade your apps first and then try the above command. To update and upgrade give command,**
            - **sudo apt update**
            - **sudo apt upgrade**
            - **sudo apt install g++ python3 cmake ninja-build git**
    b. **python3 -m pip install --user cppyy==2.4.1**
        i. **If you see that pip is not installed, first install pip and then try the above command. To install pip give command,**
            - **sudo apt install python3-pip**
            - **python3 -m pip install --user cppyy==2.4.1**
3. Configure ns3 with python support
    a. **First go to ns-allinone-3.40/ns-3.40 directory in terminal and then command,**
        i. **./ns3 configure --enable-python-bindings**
4. Build ns3
    a. **./ns3 build**
5. Run your first script
    a. **./ns3 shell**
    b. **python3 examples/tutorial/first.py**
6. Try to understand our first simulation script
    a. Create nodes/devices and connect them
        i. Creating nodes:



```
nodes = ns.network.NodeContainer()
nodes.Create(2)
```

We create a NodeContainer and create multiple nodes inside that NodeContainer so that we can manage all the nodes at once through the NodeContainer

        ii. Create channel and set attributes:

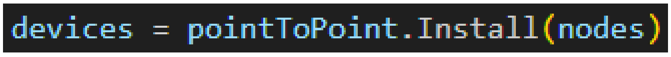


```
pointToPoint = ns.point_to_point.PointToPointHelper()
pointToPoint.SetDeviceAttribute("DataRate", ns.core.StringValue("5Mbps"))
pointToPoint.SetChannelAttribute("Delay", ns.core.StringValue("2ms"))
```

We need to create a channel that will be used for the communication. In this case, we are using a point to point channel by getting a PointToPointHelper object. This PointToPointHelper will manage all the low level transmission related works.

Then we set some attributes of this channel. Here we are setting the DataRate of our channel as 5Mbps and the delay of the channel as 2ms.

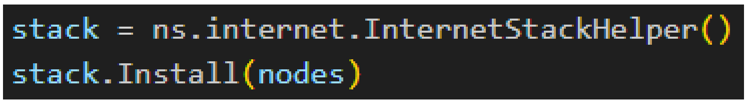
iii. Connect the created nodes using the point-to-point channel:



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

Installing the channel to the nodes gives us the NetDevice that is used for network communication.

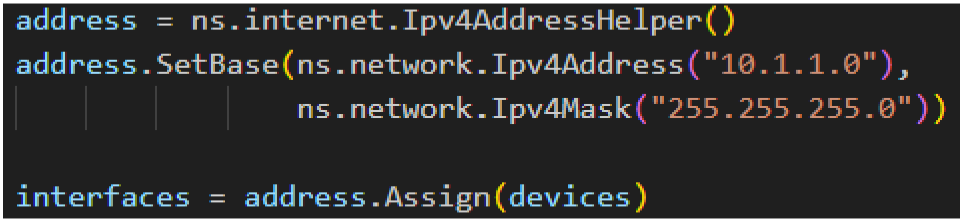
b. Install the internet stack on the nodes:



```
stack = ns.internet.InternetStackHelper()
stack.Install(nodes)
```

We need to install the network stack on the nodes so that the nodes follow the common network layering and protocols. The InternetStackHelper does the necessary works to install the network stack on the nodes.

c. Assign IP addresses to devices:



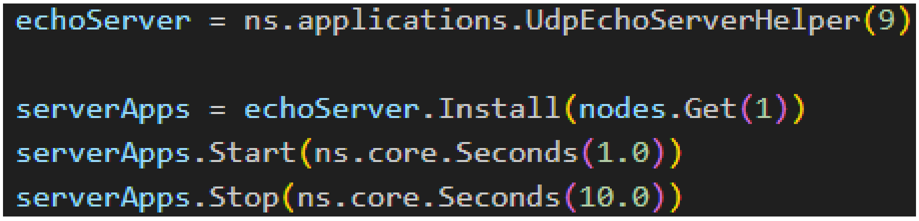
```
address = ns.internet.Ipv4AddressHelper()
address.SetBase(ns.network.Ipv4Address("10.1.1.0"),
                ns.network.Ipv4Mask("255.255.255.0"))

interfaces = address.Assign(devices)
```

We need to uniquely identify the devices in the network for communication. For this reason, we need to assign IP addresses to each devices. So, we first create a network with a network address and subnet mask. Then, we assign IP addresses to each devices. The Ipv4AddressHelper does all the necessary works to do that.

d. Creating server and client application:

i. Configuring the server:

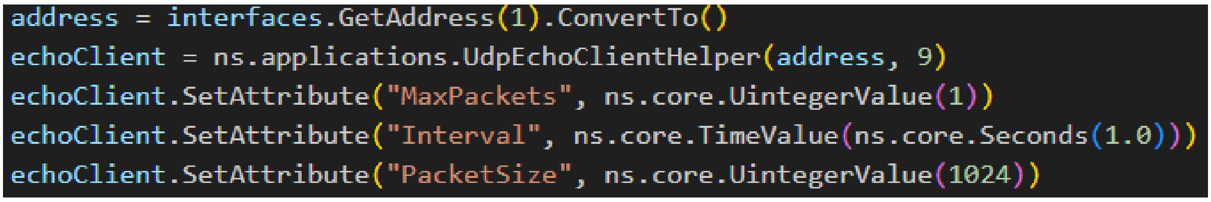


```
echoServer = ns.applications.UdpEchoServerHelper(9)

serverApps = echoServer.Install(nodes.Get(1))
serverApps.Start(ns.core.Seconds(1.0))
serverApps.Stop(ns.core.Seconds(10.0))
```

We first create a server application that listens on its port number 9 and accepts connections from any host. Then we install the server application on the second node (nodes.Get(1), indexing starts from 0). Then we define when the server application will start and when it will stop.

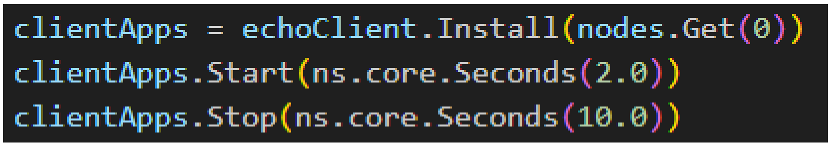
ii. Configuring the client:



```
address = interfaces.GetAddress(1).ConvertTo()
echoClient = ns.applications.UdpEchoClientHelper(address, 9)
echoClient.SetAttribute("MaxPackets", ns.core.UintegerValue(1))
echoClient.SetAttribute("Interval", ns.core.TimeValue(ns.core.Seconds(1.0)))
echoClient.SetAttribute("PacketSize", ns.core.UintegerValue(1024))
```

We have to configure the client application and define what it will do. In this case, it tries to connect to the server by first fetching the address of the server. Then we are creating a client application by telling it to connect to the server address with port number 9. Then we define some attributes of the client application. We set the **MaxPacket** to fix how many packets the client will send the server. Then, we set the **Interval** to fix the delay between successive packet sending. Then, we set the **PacketSize** and tell the client application to send 1024 bytes of data to the server.

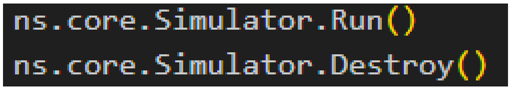
iii. Install the client application:



```
clientApps = echoClient.Install(nodes.Get(0))
clientApps.Start(ns.core.Seconds(2.0))
clientApps.Stop(ns.core.Seconds(10.0))
```

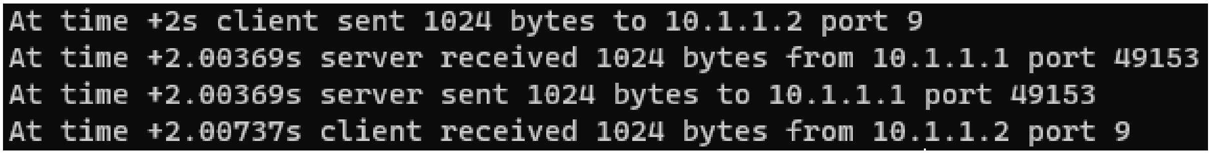
We now install the client application on the first node and set the start and stop time of the client application.

e. Running the simulation:



```
ns.core.Simulator.Run()
ns.core.Simulator.Destroy()
```

7. Observing the output:



```
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

Here we can see the traces of the network events that are performed by the server and client. We can see that 1024 bytes of data is being sent and received. Also, port 9 is used by the server as we had configured. We can also see the time in the simulation when the events happened. The transmission started at 2s because we configured the client to start at 2s.

8. Visualizing the simulation:

a. Installing NetAnim:

NetAnim is a simulation visualization tool that comes bundled with ns3. To use NetAnim, we need to perform some steps.

i. Install prerequisite packages by running the following command:

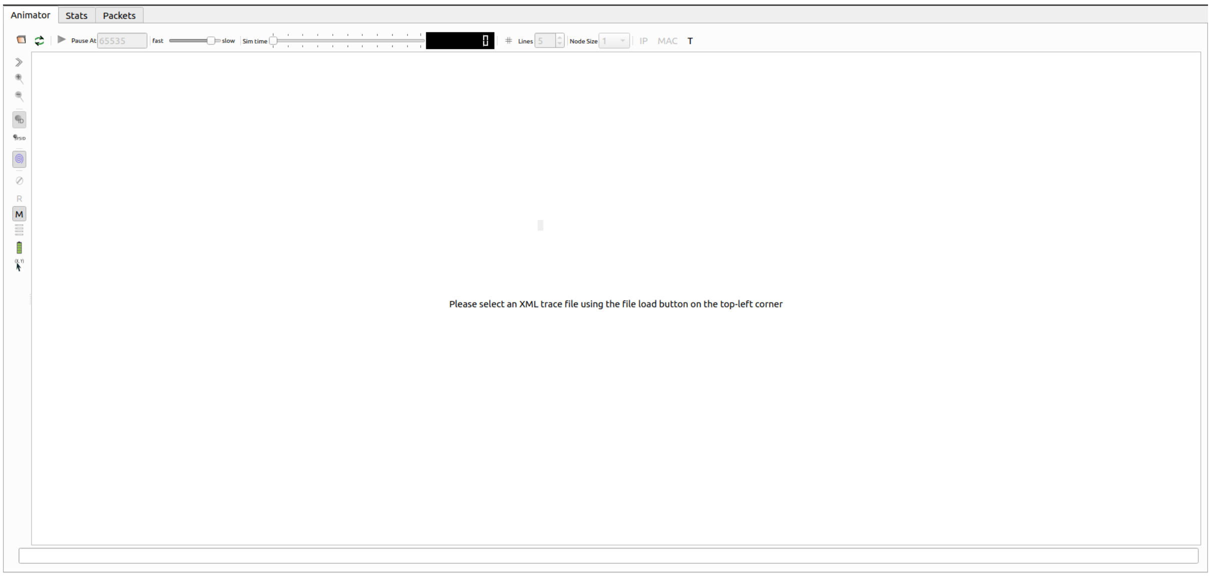
- sudo apt install qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools

ii. Build NetAnim:

- Move inside the **netanim-3.109** directory.
- Run the following commands:
    a. qmake NetAnim.pro
    b. make

iii. Run NetAnim:

- **./NetAnim**
- The interface should look like the following:



Please select an XML trace file using the file load button on the top-left corner

b. Generating the visualization script:
   i. Add the following line of code **before calling** the Run() method:
      - **anim = ns.netanim.AnimationInterface("first.xml")**
   ii. This will generate a “first.xml” file after running the simulation. This file contains the necessary scripts to run the visualization of our simulation.

c. Open the “first.xml” file in NetAnim and play the simulation. You will see a visual representation of the network events happened in our simulation.

9. Collecting network trace:
   We can collect packet captures from all our nodes in the simulation. These packet captures can help to understand the network events and troubleshoot the network.
   a. Add the following line of code **before calling** the Run() method:
      i. pointToPoint.EnablePcapAll("first")
   b. This will generate a pcap file for every node in our simulation. As we had only two nodes in our simulation, we will get two pacp files after running the simulation.
   c. Open the pcap files and observe the packets.
      i. For this we have to istall wireshark. So give command
         - **sudo apt install wireshark**
      ii. Then double click on the pcap files.

10. Getting network metrics and flow statistics:
    a. Add the following code block instead of calling the Run() method:

```
flowmon_helper = ns.flow_monitor.FlowMonitorHelper()
monitor = flowmon_helper.InstallAll()
monitor = flowmon_helper.GetMonitor()

ns.core.Simulator.Stop(ns.core.Seconds(20.0))
ns.core.Simulator.Run()

def print_stats(st):
    print("Tx Bytes: ", st.txBytes)
    print("Rx Bytes: ", st.rxBytes)
    print("Tx Packets: ", st.txPackets)
    print("Rx Packets: ", st.rxPackets)
    print("Lost Packets: ", st.lostPackets)
    if st.rxPackets > 0:
        print("Mean Delay: ", (st.delaySum.GetSeconds() / st.rxPackets))
        print("Throughput: ", (st.rxBytes*8)/18)

monitor.CheckForLostPackets()
classifier = flowmon_helper.GetClassifier()

for flow_id, flow_stats in monitor.GetFlowStats():
    t = classifier.FindFlow(flow_id)
    proto = {6: 'TCP', 17: 'UDP'} [t.protocol]
    print ("FlowID: %i (%s %s/%s --> %s/%i)" % \
        (flow_id, proto, t.sourceAddress, t.sourcePort, t.destinationAddress,
t.destinationPort))
    print_stats(flow_stats)
```

    b. Run the simulation and observe the output.

Home Task:

Now **vary the packet size and observe the metrics**. Take packet size [128,256,512,1024,2048] bytes and collect the throughputs for each of them. Plot the throughput vs packet size in this case and explain the graph.