IAS-NASI-INSA

SUMMER RESEARCH FELLOWSHIP PROGRAMME 2024

Simulating and Enhancing Network Protocols in NS-3

Bindu Madhavi V, ENGS1411

Department of Computer Science and Engineering

Global Academy of Technology

Bengaluru, Karnataka

Under the Guidance of

Dr. Pavan Kumar C

Department of Computer Science and Engineering

Indian Institute of Information Technology Dharwad, Karnataka

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr Pavan Kumar C for his continuous guidance. Being my first summer fellowship, the knowledge and experience gained during these 2 months have been enriching and fruitful.

I thank IAS-NASI-INSA for providing such a wonderful opportunity and for extending their constant support throughout the programme.

Lastly, I thank the Indian Institute of Information Technology, Dharwad for providing accommodation and other essential requirements thereby making my stay comfortable.

TABLE OF CONTENTS

ABSTRACT

Protocols play a major role in determining how entities communicate across computer networks. Network analysis is important to monitor the data transfer, and identify traffic patterns and weaknesses in the network infrastructure to ensure Quality of Service (QoS). All these activities can be monitored with the help of a network simulator like NS-3. The main goal here is to use NS-3 to assess the performance metrics like throughput, delay, jitter, and packet loss of existing protocols, identify bottlenecks and make informed decisions about protocols' improvements.

INTRODUCTION

NS-3 is an open-source, discrete-event network simulator specifically designed for Internet systems. It is a popular choice for researchers and educators due to its ease of use, speed and accuracy compared to other simulators. While the core simulation scripts are written in C++, NS-3 provides Python scripting support as well. An NS-3 user will obtain the ns-3 source code, compile it into shared or static libraries, and link the libraries to the main() programs that he or she authors. The main() program is where the specific simulation scenario configuration is performed.

Some of the key features of NS-3 include,

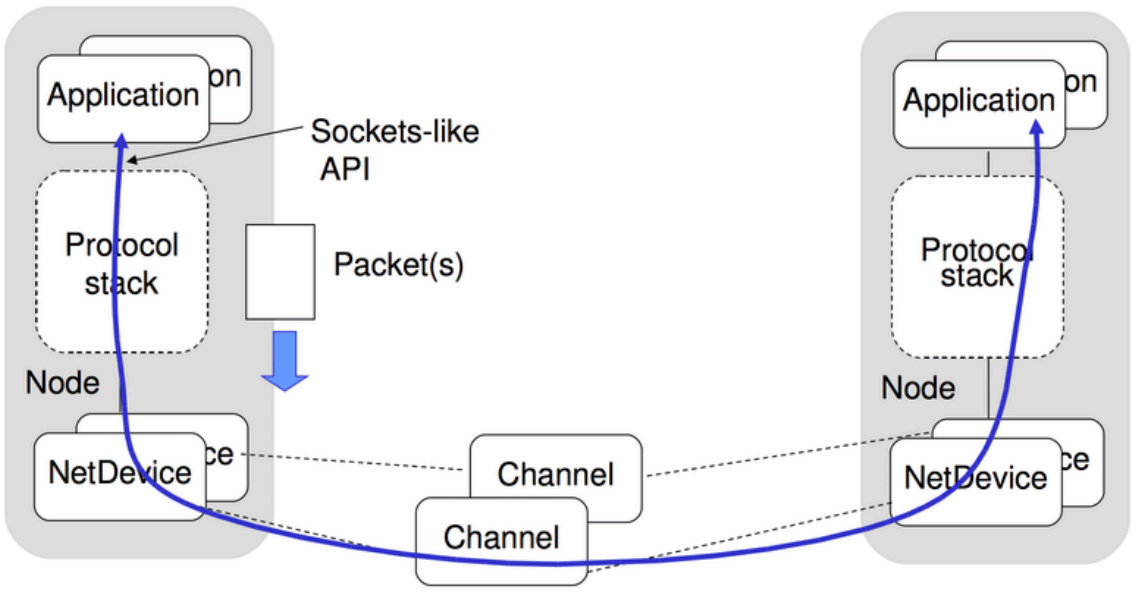
- Discrete-event Simulation - Focuses on modelling network events that occur at specific points in time. This allows for efficient and accurate representation of complex network dynamics.
- Extensible Architecture - NS-3 is built as a collection of libraries that can be combined to create customised simulations. New models and functionalities can be easily added.
- Object-Oriented Programming - The core of NS-3 is written in C++, leveraging object-oriented programming principles promoting code reusability and maintainability.
- Protocol Support - NS-3 supports a wide range of networking protocols, including routing protocols, transport protocols and application layer protocols. This allows us to analyse the performance of existing protocols or test new ones in a controlled environment.
- Visualization - NS-3 can be integrated with external tools for network visualization like NetAnim and for data analysis like Gnuplot helping in deeper analysis.

BASIC CONCEPTS

Network simulator

It is a tool or a software program that allows for analysing the relationships between different components connected in the network.

NS-3 architecture



The basic components of NS-3 include the following,

- Node - A node is a shell of a computer to which applications, protocol stacks and net devices are added.
- Application - creates or receives data sent between nodes.
- Protocol stack - Protocols are the rules governing how network entities should communicate. Mainly TCP/IP stack which has Layer 3 (IPV4, IPV6, ARP) and Layer 4 (TCP, UDP, ICMP) protocols.
- Net device - transmits and receives over the channel.
- Channels - represent the physical medium through which data travels between the nodes.
- Helper API - provides a set of classes and methods that make common operations easier. Containers group similar objects together while helper classes make the simulation scripts look nicer and easier to read.

NS-3 packets

Each network packet contains a byte buffer, a list of tags and metadata.

- Buffer - bit-by-bit (serialised) representation of headers and trailers.
- Tags - set of arbitrary, user-provided data structures (Eg: per-packet cross-layer messages or flow identifiers)
- Metadata - describes types of headers and trailers that have been serialised.

Simulation basics

Simulation time moves discretely from event to event.

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

NS-3 objects

Most objects inherit from ns::Object. Provides a range of useful properties for simulation like Smart pointers, Object aggregation, Run-time type information, Attributes and Trace subsystem. NS-3 objects are created with CreateObject<Class> (constructor arguments).

1. Smart pointers
   A smart pointer provides a form of garbage collection. It behaves like a normal pointer but does not lose memory when the reference count reaches zero. They can be used as,
   Ptr<MyClass> p = CreateObject<MyClass> ();
   p->method ();

2. Object aggregation
   Objects can be dynamically aggregated to each other. It avoids huge classes that encompass all possible functionality. It can be used as,
   node>AggregateObject(mobility);
   Ptr<MobilityModel> mob = node>GetObject<MobilityModel> ();

3. Run-time type information
   It is a mechanism that exposes information about an object's datatype at runtime.
   All NS-3 objects must implement TypeId GetTypeId(void).
   TypeId informs about attributes, runtime type information and trace sources.

4. Attributes

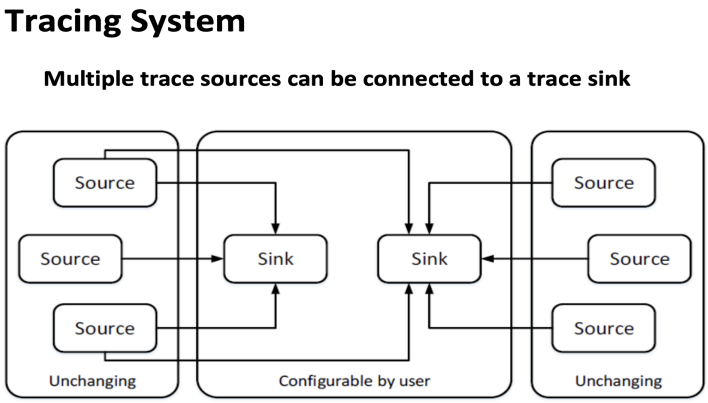
   Each ns3 object has a set of attributes that makes it easy to verify the parameters of a simulation like name, help, type and initial value.

   e.g. TcpSocket::m_cwnd;

   Users would like to,

   - Know what are all the attributes that affect the simulation at run time.
   - Set a default initial value for a variable as well as set or get the current value of a variable.
   - Initialize the value of a variable when a constructor is called.

5. Trace Subsystem



The way to collect data from the experiments and compare the results against previous implementations or different systems. Users can implement specialized trace sinks and connect to existing trace sources using pointers to functions.

- Trace Sources - These are objects within the simulation that signal significant events and provide access to related data.
- Trace Sinks - These are functions that receive callbacks triggered by trace sources. They determine what information is recorded and how it's formatted for output. Trace sinks can write data to files, display it on the console, or perform custom analysis.
- Connection Mechanism - A uniform method exists to connect trace sources and sinks. This allows you to specify which sources provide data to which sinks, enabling you to customise the information you collect.

Packet Tags

Packet tags are small chunks of information. Tags are used to attach context information to a packet. Eg: NetDevice attaches the destination MAC address when queuing, and retrieves it when dequeuing for transmission.

Mobility and Position

All nodes have to be created before the simulation starts. Position Allocators set up the initial position of nodes like List, Grid, and Random position. Mobility models specify how nodes will move i.e,

- Constant position, constant velocity/acceleration.
- Routes Mobility using Google API.

FlowMonitor

FlowMonitor is a network monitoring framework which detects all flows passing through the network. It stores metrics for analysis such as bitrates,
duration, delays, packet sizes and packet loss ratios.

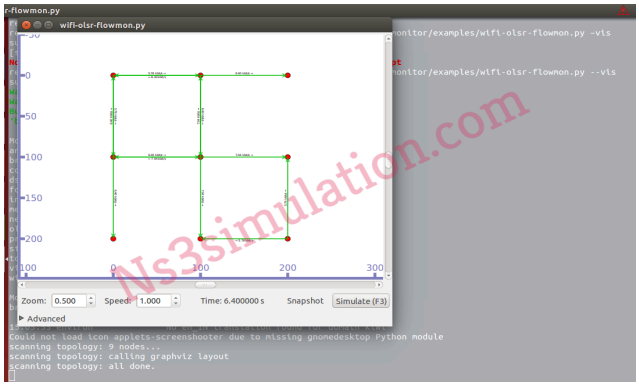
Logging

Logging is used to trace code execution logic. Some of the commonly used methods are,

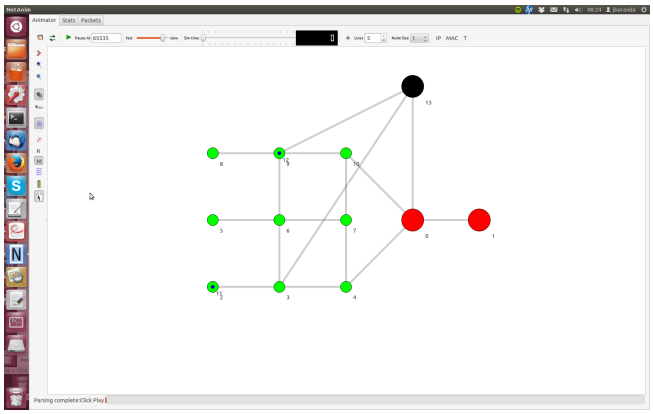
- NS_LOG_ERROR() - serious error messages only
- NS_LOG_WARN() - warning messages
- NS_LOG_DEBUG() - rare ad-hoc debug messages
- NS_LOG_INFO() - informational messages (Eg: banners)
- NS_LOG_FUNCTION() - function tracing
- NS_LOG_PARAM() - parameters to functions
- NS_LOG_LOGIC() - control flow tracing within functions

Visualisation

Some of the commonly used visualisation tools include Pyviz and NetAnim.
Pyviz is a live simulation visualiser (no trace files). It is useful for debugging mobility model behaviour and packets being dropped.



On the other hand, NetAnim animates packets over wired links and wireless links based on trace files.



Steps of an NS-3 simulation script

1. Handle command line arguments
2. Set default attribute values and random seed
3. Create nodes
4. Configure physical and MAC layers
5. Set up network stack, routing and addresses
6. Configure and install applications
7. Set up initial positions and mobility
8. Set up data collection
9. Schedule user-defined events and start the simulation

NS-3 emulation modes

TapBridge provides common emulation functionality and has different modes.

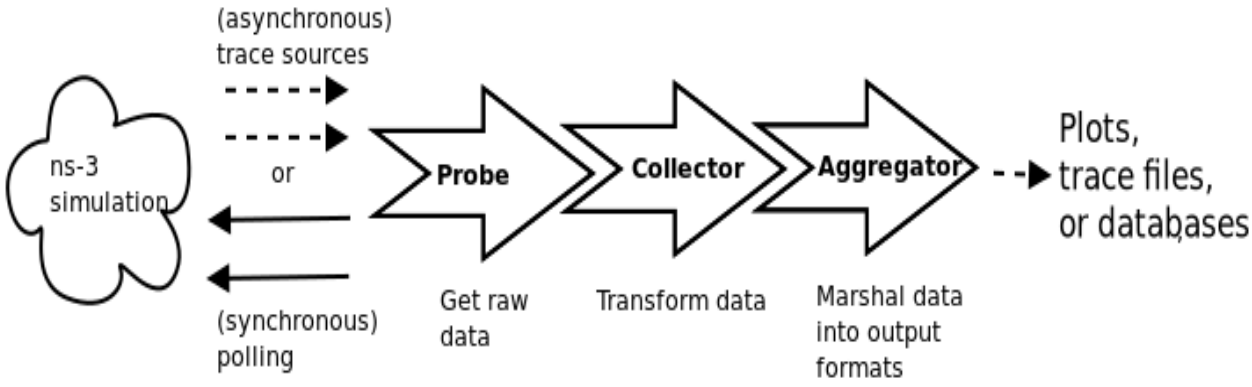
1. LocalDevice mode - NS-3 creates virtual interfaces within the emulator itself. Local processes are then attached to these interfaces.

2. BridgedDevice mode - This mode is useful when we want the VMs to control the network interfaces themselves. Here, NS-3 connects to existing virtual devices, like Linux bridges.

Data Collection Framework (DCF)

NS-3 provides a built-in DCF, designed to efficiently gather and process data generated during simulations. It includes probes, collectors, aggregators and adaptors.

- Probes capture the specific data we're interested in.
- Collectors receive data from probes and might perform preliminary processing or filtering.
- Aggregators further process the collected data, calculating statistics like averages, minimums, or maximums.
- Adaptors format the processed data for export into the desired output format.



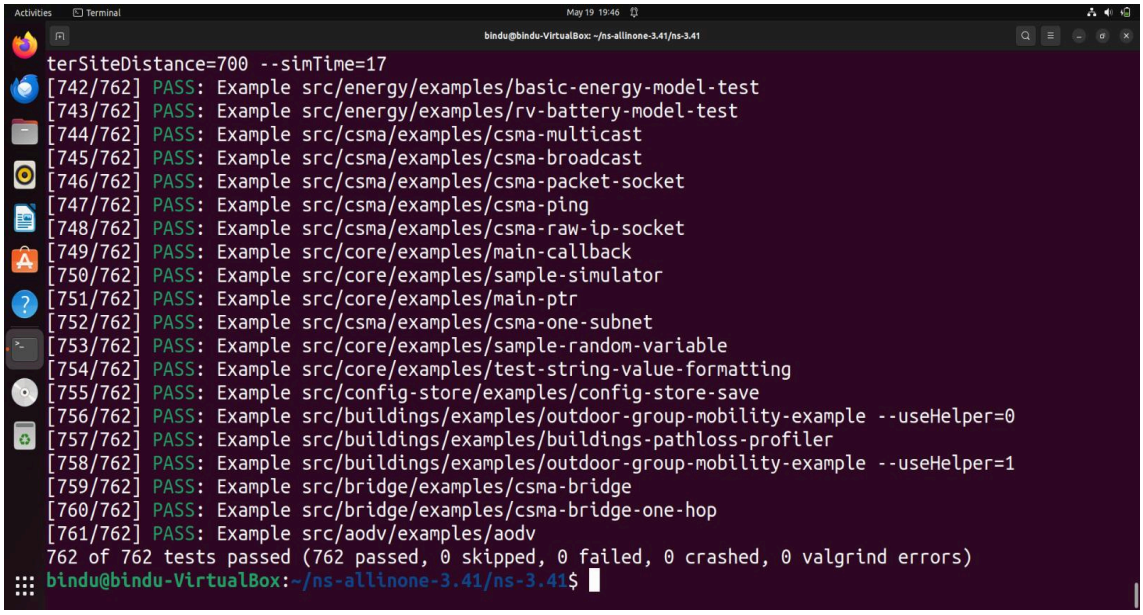
NATURE OF WORK CARRIED OUT

NS-3 Installation on Ubuntu 22.04

1. Type the following command in the terminal,
   $ sudo apt update
   $ sudo apt install g++ python3 cmake ninja-build git gir1.2-goocanvas-2.0 python3-gi python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0 ipython3 tcpdump wireshark sqlite sqlite3 libsqlite3-dev qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools openmpi-bin openmpi-common openmpi-doc libopenmpi-dev doxygen graphviz imagemagick python3-sphinx dia imagemagick texlive dvipng latexmk texlive-extra-utils texlive-latex-extra texlive-font-utils libeigen3-dev gsl-bin libgsl-dev libgslcblas0 libxml2 libxml2-dev libgtk-3-dev lxc-utils lxc-templates vtun uml-utilities ebtables bridge-utils libxml2 libxml2-dev libboost-all-dev ccache
2. Download version ns-3.41 from the website, NS-3.
3. Move the downloaded file to the home folder and extract it there.
4. A separate folder called ns-allinone-3.41 is created in the home folder with all the necessary software.
5. Open the terminal and type in the following commands,
   $ cd ns-allinone-3.41
   $ ./build.py --enable-examples --enable-tests
   [The above line will install all the packages for ns3 along with NetAnim, flow monitor, applications etc.]
   Depending on your system, it can take anywhere between 20 minutes to 2 hours.
6. Once the installation is completed. check whether the software is working by typing,
   $ cd ns-3.41/
   $ ./ns3 run hello-simulator

Testing NS-3

The build installed can be checked by running unit tests using the following command,
$ ./test.py
This command should run several hundred unit tests. If they pass, you have made a successful initial build of NS-3.



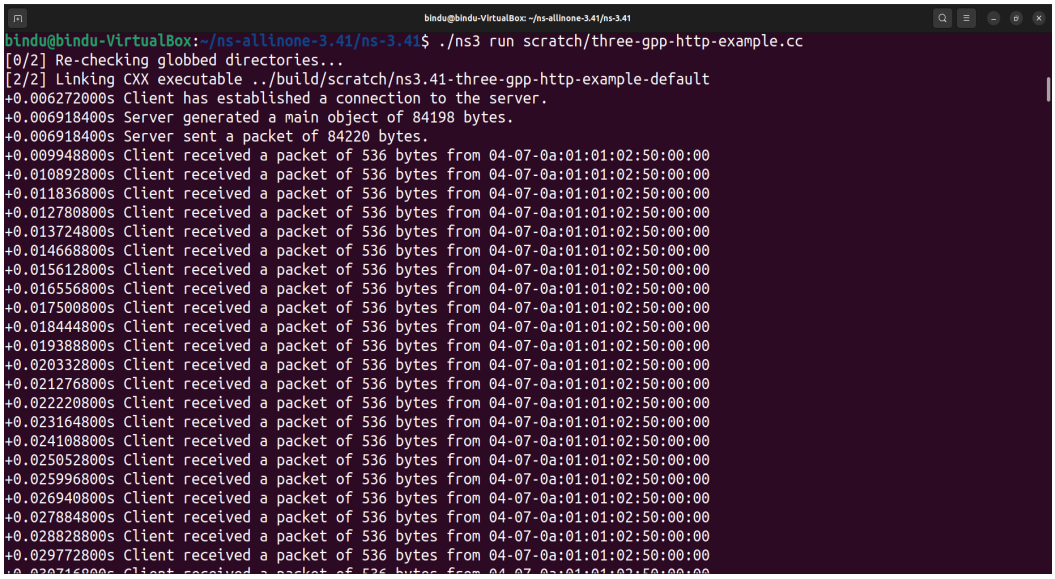
Running the programs on NS-3

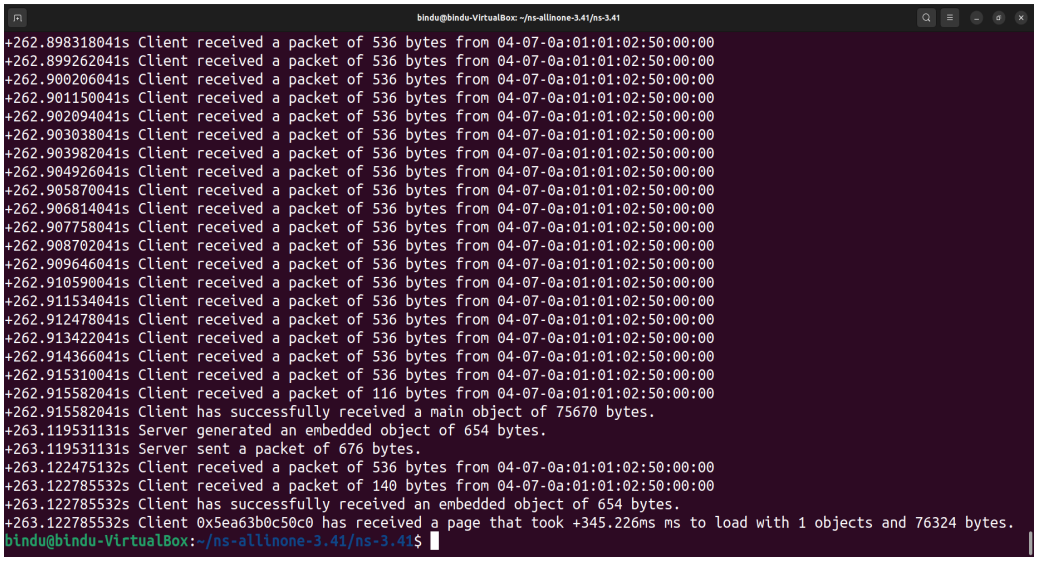
1. Scratch folders have extra permissions, modules and linkages to run .cc files and .py files. So all .cc/.py files have to be copied to the scratch folder. You can navigate to the scratch folder using,
   $ cd ns-allinone-3.41/ns-3.41/scratch
2. To run the program,
   $ cd ns-allinone-3.41/ns-3.41
   $ cd ./ns3 run scratch/<program_name>

Programs executed

1. 3GPP HTTP Application
   Design - The traffic generator simulates web browsing traffic using HTTP. It consists of one or more *ThreeGppHttpClient* apps which connect to a *ThreeGppHttpServer app*. The client transmits request objects to demand a service from the server. Depending on the type of request received, the server transmits either,
   - A main object, the HTML file of the webpage, or
   - An embedded object, Eg: an image referenced by the HTML file.

Output - HTTP object packet bytes sent match the total bytes received by the client, and the ThreeGppHttpHeader matches the expected packet.

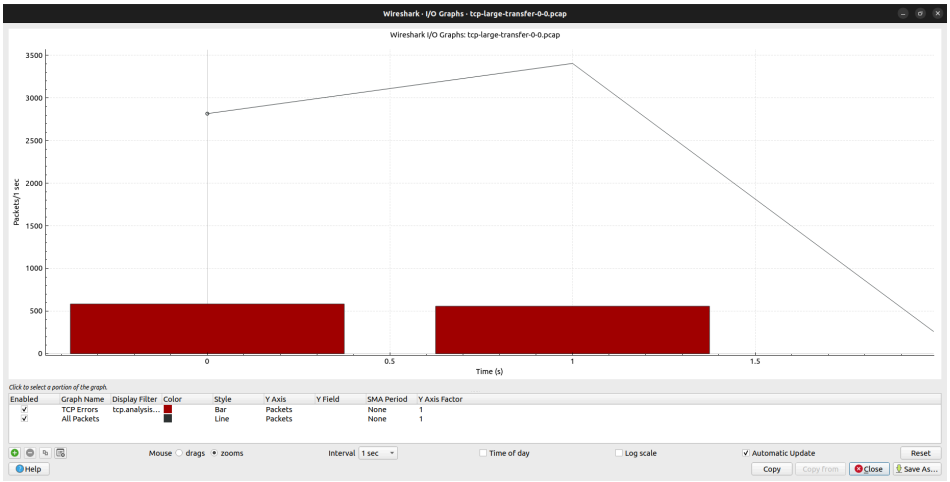


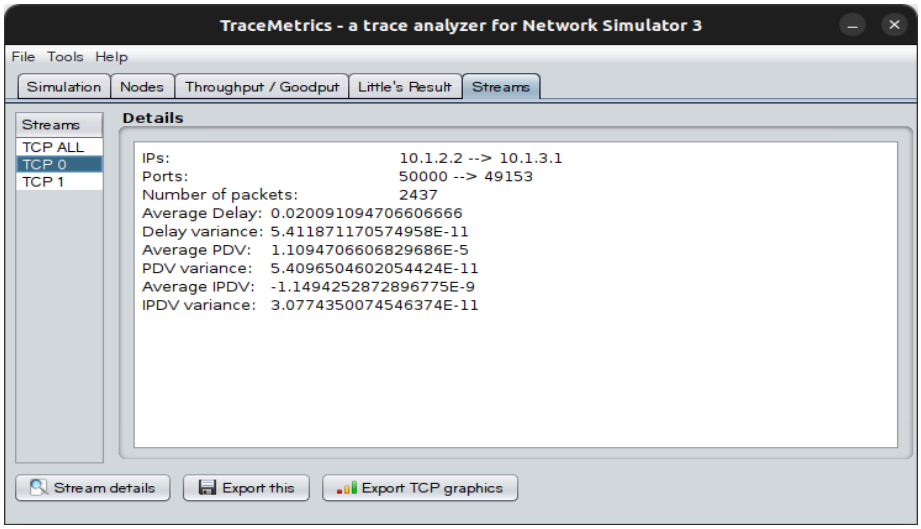


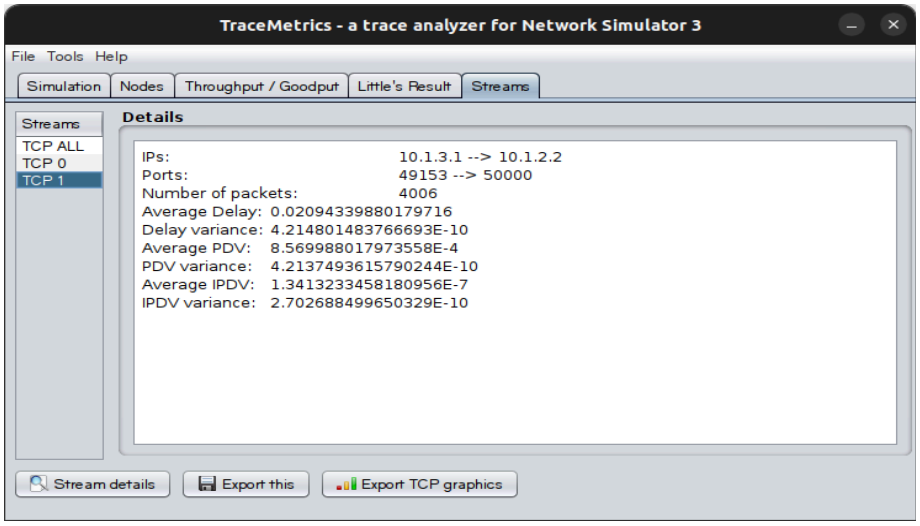
2. Bulk data transfer using TCP

   Design - The number of bytes to send is 2000000. The *writeSize* is 1040 bytes. Three nodes are explicitly created. The first container contains node 0 and node 1 [n0 and n1], while the second container contains n1 and n2. This reflects channel connectivity and will be used to install and connect the network interfaces with a channel. *SndBuffSize = 4096*. In this simulation, we send 2000000 bytes over a connection to server port 50000 at time 0. TCP default MSS is 536 and delayed ACK count is 2. Here we observe SYN exchange, a lot of data segments, ACKS and FIN exchange.

Output - Four pcap files and one trace file are created in this program. Out of which a glimpse of the I/O graph of the first pcap file and the trace file is given below.







3. Point-to-point network

   Design - Two wired nodes connect through a point-to-point network that handles a data rate of 50mbps and a delay of 5ms. The server (1st node) and client (second node) exchange at least 10 packets in a total simulation time of 20 seconds. The maximum packet size is 1024 bytes and 512 bytes.

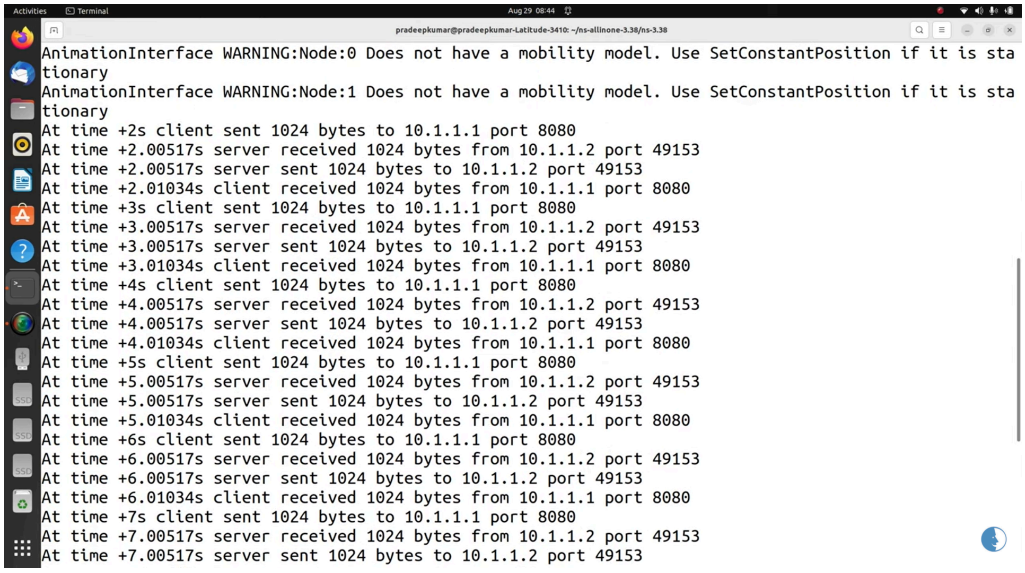
   Code for Tracemetrics and Animation

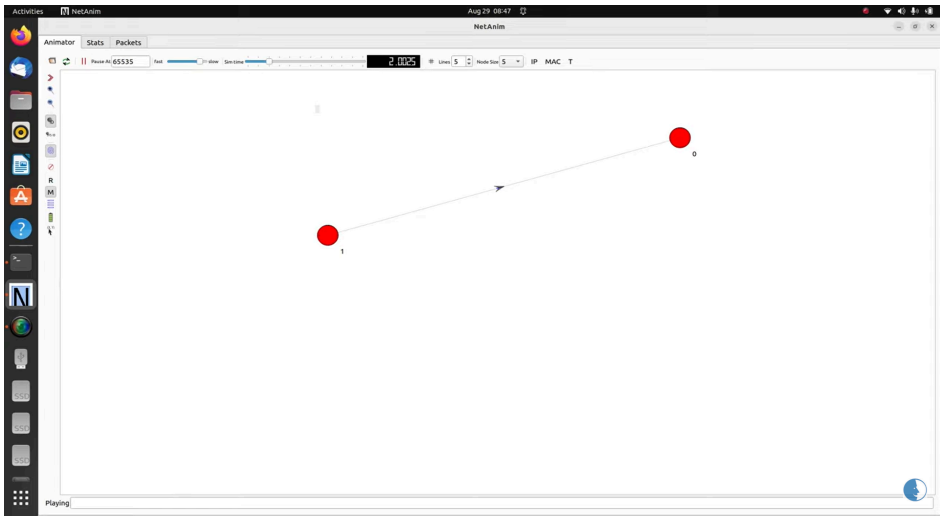
```
//Ascii trace
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("1024
.tr"));
//NetAnim
AnimationInterface anim("p2p.xml");
```

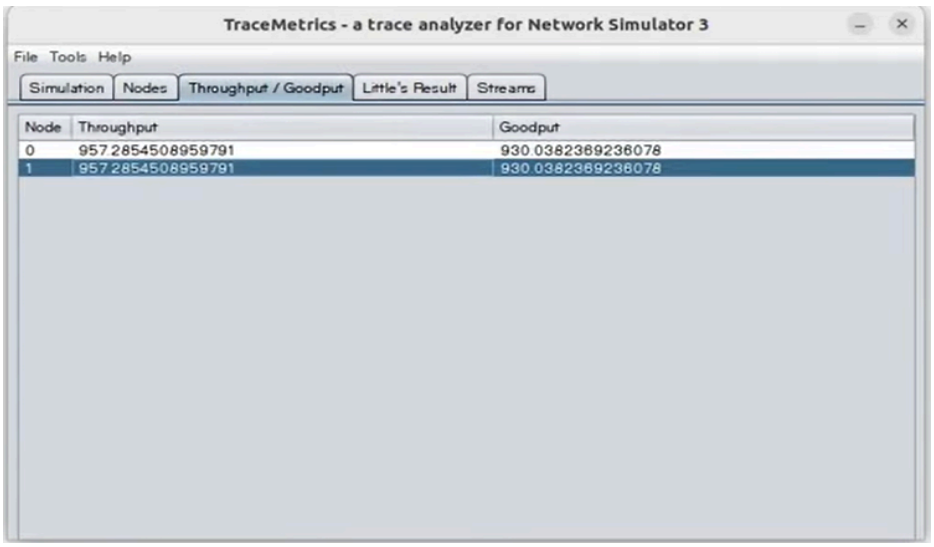
   Code for plotting the throughput values of each node of 2 simulations

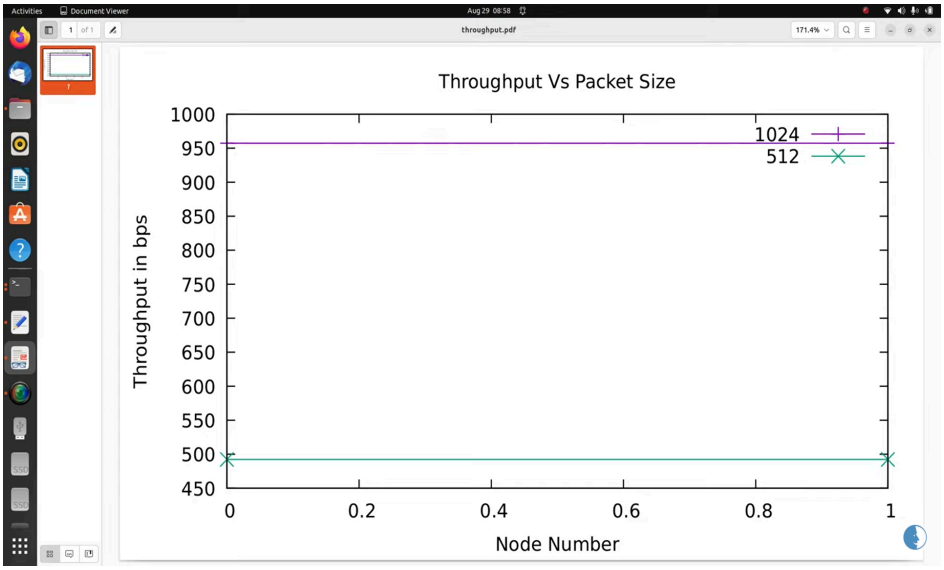
```
set terminal pdf
set output "throughput.pdf"
set title "Throughput Vs PacketSize"
set xlabel "Node Number"
set ylabel "Throughput in bps"
plot "throughput.txt" using 1:2 with linespoints title
"1024","throughput.txt" using 1:3 with linespoints
title "512"
```

Output - Both client and server have exchanged packets. The exchange of packets is visualized using NetAnim, throughput calculated is depicted in the trace file figure. A graph plotted between throughput and packet size using Gnuplot is shown in the last figure.


Activities
Terminal
Aug 29 08:44
pradeepkumar@pradeepkumar-Latitude-3410: ~/ns-allinone-3.38/ns-3.38
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
At time +2s client sent 1024 bytes to 10.1.1.1 port 8080
At time +2.00517s server received 1024 bytes from 10.1.1.2 port 49153
At time +2.00517s server sent 1024 bytes to 10.1.1.2 port 49153
At time +2.01034s client received 1024 bytes from 10.1.1.1 port 8080
At time +3s client sent 1024 bytes to 10.1.1.1 port 8080
At time +3.00517s server received 1024 bytes from 10.1.1.2 port 49153
At time +3.00517s server sent 1024 bytes to 10.1.1.2 port 49153
At time +3.01034s client received 1024 bytes from 10.1.1.1 port 8080
At time +4s client sent 1024 bytes to 10.1.1.1 port 8080
At time +4.00517s server received 1024 bytes from 10.1.1.2 port 49153
At time +4.00517s server sent 1024 bytes to 10.1.1.2 port 49153
At time +4.01034s client received 1024 bytes from 10.1.1.1 port 8080
At time +5s client sent 1024 bytes to 10.1.1.1 port 8080
At time +5.00517s server received 1024 bytes from 10.1.1.2 port 49153
At time +5.00517s server sent 1024 bytes to 10.1.1.2 port 49153
At time +5.01034s client received 1024 bytes from 10.1.1.1 port 8080
At time +6s client sent 1024 bytes to 10.1.1.1 port 8080
At time +6.00517s server received 1024 bytes from 10.1.1.2 port 49153
At time +6.00517s server sent 1024 bytes to 10.1.1.2 port 49153
At time +6.01034s client received 1024 bytes from 10.1.1.1 port 8080
At time +7s client sent 1024 bytes to 10.1.1.1 port 8080
At time +7.00517s server received 1024 bytes from 10.1.1.2 port 49153
At time +7.00517s server sent 1024 bytes to 10.1.1.2 port 49153


Activities
NetAnim
Aug 29 08:47
NetAnim
Animator
Stats
Packets
0
1
Playing


TraceMetrics - a trace analyzer for Network Simulator 3
File Tools Help
Simulation
Nodes
Throughput / Goodput
Little's Result
Streams
Node
Throughput
Goodput
0
957.2854508959791
930.0382369236078
1
957.2854508959791
930.0382369236078



4. Comparison of MANET Routing Protocols

Design - The simulation runs for 200s, of which the first 50 are used for start-up time. The number of nodes is 50. Nodes move according to *RandomWaypointMobilityModel* with a speed of 20 m/s and no pause time within a 300x1500 m region. The WiFi is in ad hoc mode with a 2 Mb/s rate (802.11b) and a Friis loss model. The transmit power is set to 7.5 dBm. There are 10 source/sink data pairs sending UDP data at an application rate of 2.048 Kb/s each. This is typically done at a rate of 4 64-byte packets per second. Application data is started at a random time between 50 and 51 seconds and continues to the end of the simulation.

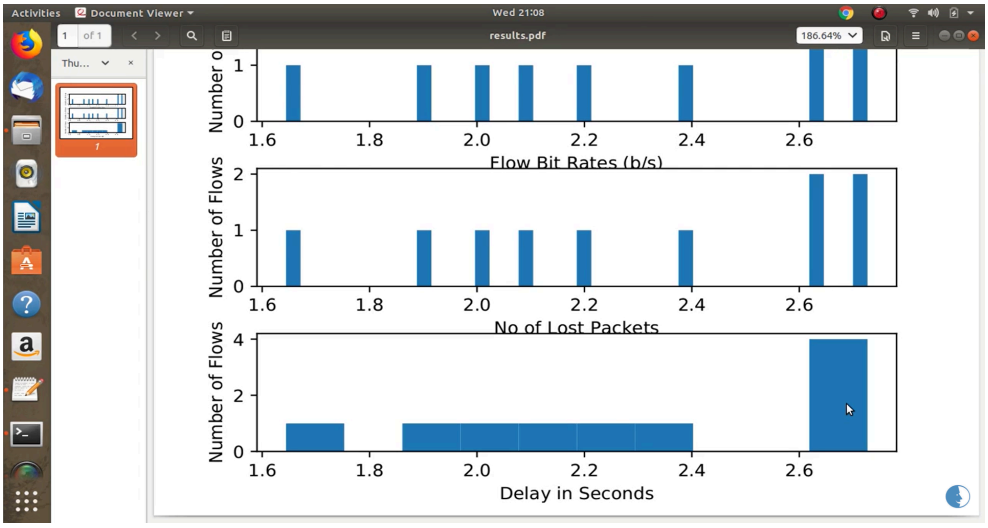
The protocols used for comparison are AODV, OLSR, DSDV and DSR. By default, OLSR is used, but specifying a value of 2 for the protocol will cause AODV to be used, a value of 3 will cause DSDV to be used and a value of 4 will cause DSR to be used.

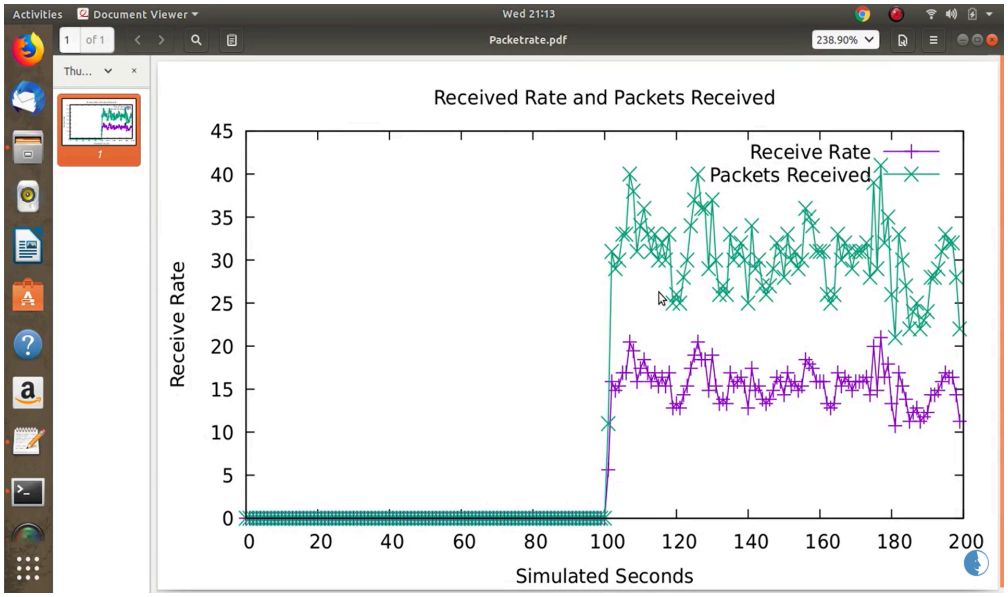
Code for plotting a graph between simulated seconds and receive rate

```
set terminal pdf
set output "Packetrate.pdf"
set title "Receive Rate"
set xlabel "Simulated Seconds"
set ylabel "Receive Rate"
plot "AODV.csv" using 1:2 with linespoints title "Receive
```

```
Rate", "AODV.csv" using 1:3 with linespoints title "Packets
Received"
```

Output - Three files will be created i.e., CSV, mob and flowmon files.



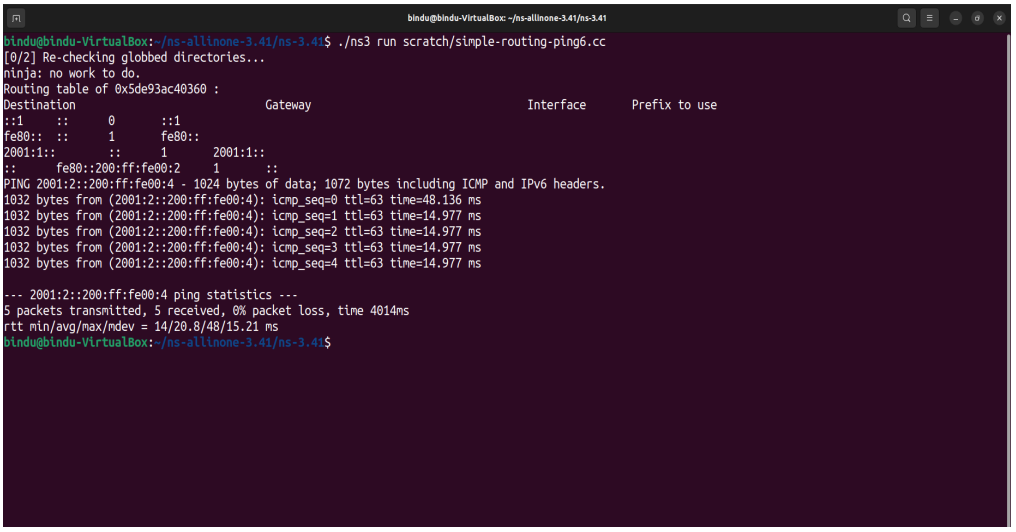


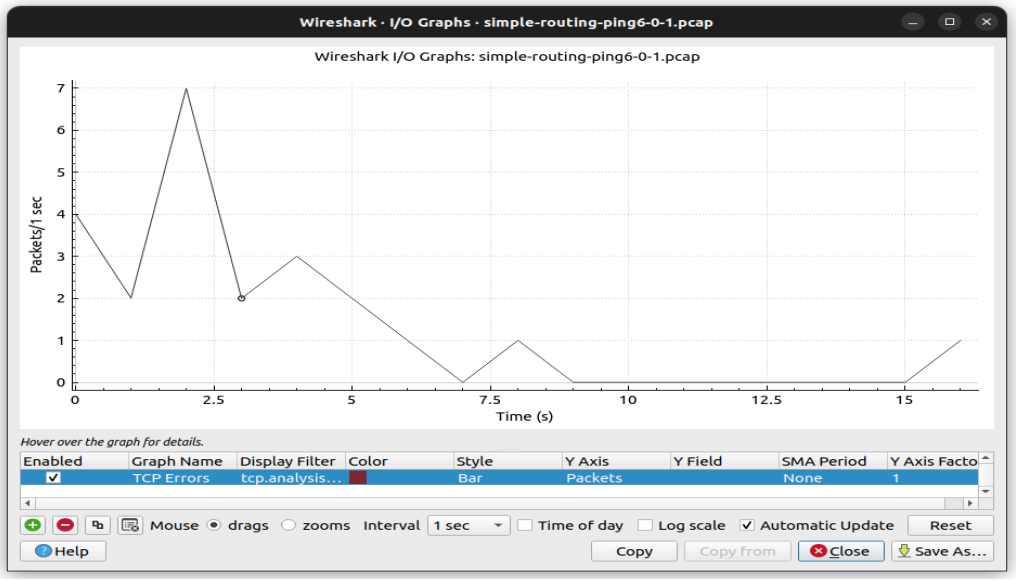
5. Simulating a network with IPV6 routing and sending ping requests between two nodes.

   Design - A simple network topology with three nodes: two hosts (n0, n1) and one router (r) is defined. *CsmaHelper* creates channels between nodes using the CSMA protocol. It configures data rate and delay for the channels. *InternetStackHelper* installs the IPv6 protocol stack on all nodes, enabling communication using IPv6 addresses. *Ipv6AddressHelper* assigns IPv6 addresses

to network interfaces on each node. The program configures routing by setting default routes and enabling forwarding on specific interfaces. *Ping6Helper* creates a Ping6 application on the n0 to send ICMPv6 echo requests to n1. The router forwards the ping requests based on the routing table configuration. The program uses *StackHelper* to manage IPv6 addresses and print routing tables.

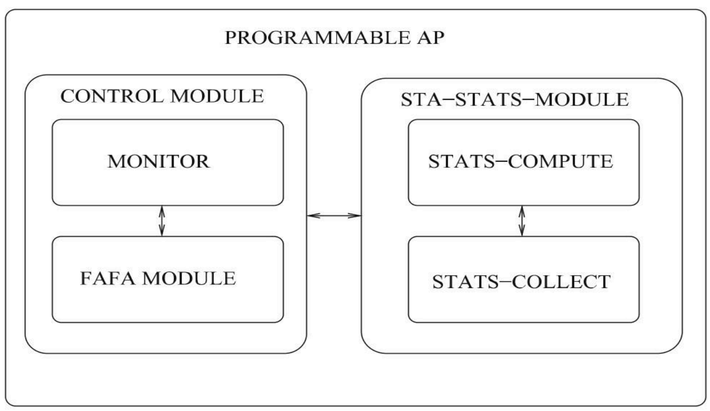
Output - The routing table printed is shown in the first image. Four pcap files are generated, out of which the I/O graph of the first pcap file is shown in the second image.





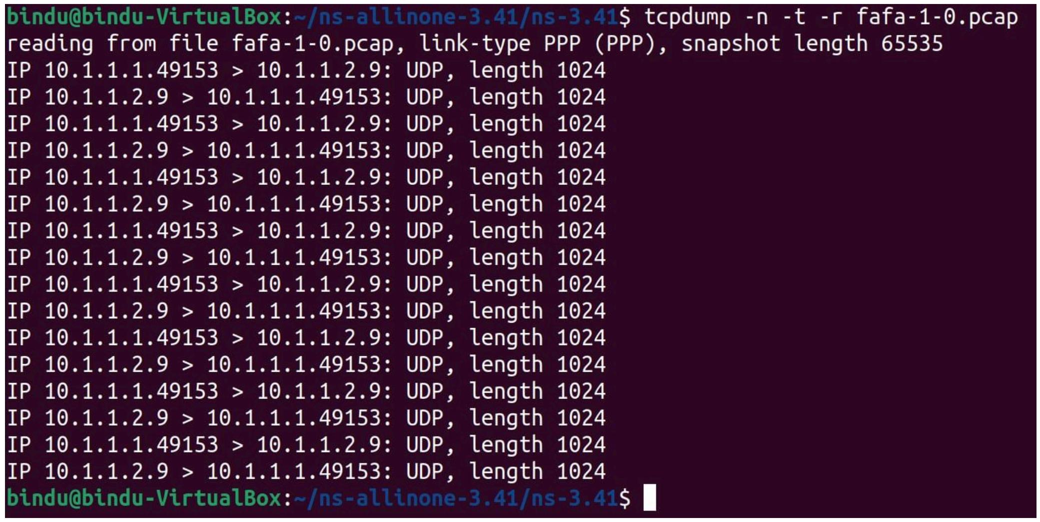
6. Fairness and Applications' transport protocol aware frame aggregation using programmable WLANs (FAFA)

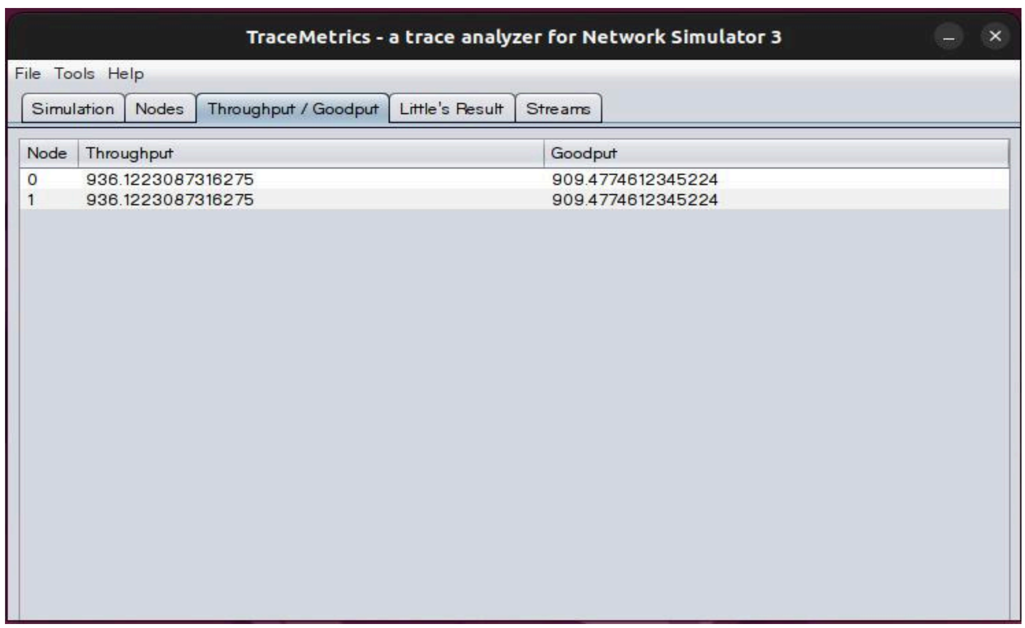
   Design - The *CONTROL* module contains a *Monitor* and a variety of *WLAN control algorithms* like association, rate control, power management, aggregation control, etc. The *CONTROL* module can interact with the *STA-STATS* module to access its connected STAs' important details like *SNR, average transmission rate, aggregation size (AGG_SIZE),* and performance metrics such as *average throughput, delay, and jitter. STA-STATS* module contains STATS-COLLECT and *STATS-COMPUTE* sub-modules. The *STATS-COLLECT* continuously monitors the various STAs' uplink and downlink flows going through AP. Besides, the *STATS-COMPUTE* interacts with the *STATS-COLLECT* module and computes important metrics at both STAs and BSS levels. *STATS-COMPUTE* module monitors STAs traffic and computes STA level details like *average SNR, AGG_SIZE* and *Frame Success Delivery rate.*



   Output - Two pcap files and a trace file are generated.

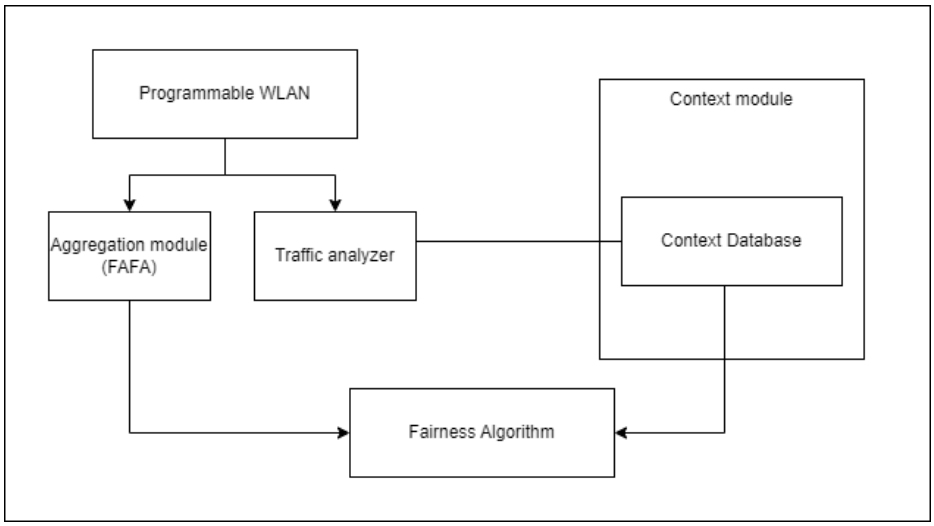
```
bindu@bindu-VirtualBox:~/ns-allinone-3.41/ns-3.41$ tcpdump -n -t -r fafa-0-0.pcap
reading from file fafa-0-0.pcap, link-type PPP (PPP), snapshot length 65535
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
bindu@bindu-VirtualBox:~/ns-allinone-3.41/ns-3.41$
```





7. Context-level fairness (CFL) in programmable WLANs
   This is a newly proposed idea as an extension to FAFA because it showcased only user-specific fairness and application-specific fairness.
   CFL Offers distributed control i.e, shares the workload among devices, potentially reducing strain on the Access Point (AP). It also offers adaptability where devices can adjust fairness based on local conditions and user needs.

Design of Programmable-AP (Pro-AP)



Following is the breakdown of each block of the design,

- STAs in the *programmable WLAN* are the source of traffic pointing to both the *Aggregation module* and *Traffic analyzer*.
- The *Traffic analyzer* connects to the *Context Database* in the *Context module* representing data storage of extracted context information.
- The *Fairness Algorithm* receives input from both the context database and *Aggregation module* for real-time network data.
- Decisions from the *Fairness Algorithm* influence the *Aggregation module* for context-aware frame aggregation.
- Aggregated frames are then transmitted.

Proposed Algorithm

1. Initialization - User information and context from the context database.
2. User Information Update - Update channel SNR for each user.
3. Context-aware weighting - Update weight based on obtained context.
4. Demand-Supply Ratio (DSR) calculation - Calculate DSR for each user.
5. Fairness Evaluation - Calculate Jain's Fairness Index (JFI).
6. FAFA trigger check - FAFA is activated when the fairness metric is below a threshold value.
7. Determination of aggregation size (AGG_SIZE) - Use FAFA/context-aware mode to determine AGG_SIZE and prioritize users with lower weighted DSR for smaller aggregation sizes.
8. Aggregation - Transmit frames according to the determined aggregation sizes.
9. Adaptation - Based on reports/fairness metric changes, the Pro-AP can

dynamically adjust aggregation sizes or switch between FAFA and context-aware modes.

Evaluation Metrics

1. Delay
   D = Time taken for a packet to travel from source to destination in sec
2. Jitter
   J = Absolute_Difference { Delay - Average_Delay }
3. Throughput
   T = Total amt. of data transferred (in bits)/Time interval(sec)
4. Frames-Success Delivery Rate (FSDR)
   *FSDR = (Number of successfully delivered frames) / (Total number of transmitted frames) * 100*
5. Packet overhead
   Packet_Overhead = Total_Packet_Size - Payload_Size

CONCLUSION

This project utilised NS-3, a powerful network simulator to delve into the intricacies of multiple protocols across the network. The simulations provided valuable insights into analysing QoS parameters like delay and throughput. These results are instrumental in optimising network design, developing novel protocols and making informed decisions in the realm of networking. Further research can be conducted to gain a more comprehensive understanding of fairness protocols and their evaluation.

REFERENCES

1. Rudra, A. R., Somayaji, S. L., Singh, S., Mokashi, S. D., Rakshit, A., Khan, D., & Tahiliani, M. P. (2023, June). Linux-like Socket Statistics Utility for ns-3. In *Proceedings of the 2023 Workshop on ns-3* (pp. 121-126).
2. Rangisetti, A. K., Dwivedi, R., & Modem, S. (2023). Fairness and Applications' transport protocol aware frame aggregation using programmable WLANs. *Wireless Networks*, *29*(2), 857-876.
3. Abdelhakim, B. A., & Mohamed, B. A. (2024). Check for updates A Comparative Analysis of MANET Routing Protocols Using NS2 and NS3 Simulators. In *Innovations in Smart Cities Applications Volume 7: The Proceedings of the 8th International Conference on Smart City Applications, Volume 1* (Vol. 1, p. 240). Springer Nature.
4. Liu, G., & Kong, L. (2023). Simulation of Video Streaming Over Wireless Networks with NS-3. *arXiv preprint arXiv:2302.14196*.
5. https://www.nsnam.org/