COMPARISON OF PROPAGATION MODELS

COURSE PROJECT REPORT

Submitted by:

Rajan Kataria (18103076)
Rippendeep Kaur (18103078)
Sanyamdeep Singh (18103083)
Vaishnavi Lokhande (18103093)

Under the guidance of

Asst. Prof. Kunwar Pal
CSE Department, NIT Jalandhar

A report submitted for the partial fulfillment of the course:

ADVANCED COMPUTER NETWORKS LABORATORY| CSPE-352

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY, JALANDHAR

ACADEMIC YEAR: 2020-2021

TABLE OF CONTENTS

S. No.	Title	Page No.
1.	Program Details	1
2.	Group Members' Details	1
3.	Subject Name/Code	1
4.	System Configuration	1
5.	Theory	1-3
6.	Code	4-13
7.	Output	13-15
8.	Analysis of Output from various files	15-23

1. PROGRAM DETAILS

Take any 2-propagation model scenario individually and compare with respect to any predefined program in ns-3.

2. GROUP MEMBERS' DETAILS

Sr. No.	Roll No.	Name	Course	Contact
1.	18103076	Rajan Kataria	B-Tech. (CSE)	rajank.cs.18@nitj.ac.in
2.	18103078	Rippendeep Kaur	B-Tech. (CSE)	rippendeepk.cs.18@nitj.ac.in
3.	18103083	Sanyamdeep Singh	B-Tech. (CSE)	sanyamdeeps.cs.18@nitj.ac.in
4.	18103093	Vaishnavi Lokhande	B-Tech. (CSE)	vaishnavil.cs.18@nitj.ac.in

3. SUBJECT NAME / CODE

Advanced Computer Networks Laboratory (CSPE-352)

4. SYSTEM CONFIGURATION

4.1 **Processor:** Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
4.2 **RAM:** 8.00 GB (7.89 GB usable)
4.3 **System type:** 64-bit operating system, x64-based processor
4.4 **Operating System:** Windows 10 Home Single Language (Version 20H2)

5. THEORY

We have used two propagation loss models; **Friis Propagation Loss model** and **Fixed RSS Loss model** on a simple wifi network containing two nodes (AP wifi node and STA wifi node).

5.1 Friis Propagation Loss Model in ns3:

Detailed Description

The Friis propagation loss model was first described in "A Note on a Simple Transmission Formula", by "Harald T. Friis".

The original equation was described as: $P_r / P_t = (A_r * A_t) / (d^2 \lambda^2)$ with the following equation for the case of an isotropic antenna with no heat loss: $A_{isotr} = \lambda^2 / (4\pi d)^2$

The final equation becomes: $P_r/P_t = \lambda^2 / (4\pi d)^2$

Modern extensions to this original equation are: $(P_r = P_t * G_t * G_r * \lambda^2 / (4\pi d)^2 L)$

With:

- P_r : reception power (W)
- P_t : transmission power (W)
- G_t : transmission gain (unit-less)
- G_r : reception gain (unit-less)
- λ : wavelength (m)
- d : distance (m)
- L : system loss (unit-less)

This model is invalid for small distance values. The current implementation returns the txpower as the rxpower for any distance smaller than MinDistance.

In the implementation, λ is calculated as $\frac{C}{f}$, where $C = 299792458$ m/s is the speed of light in vacuum, and f is the frequency in Hz which can be configured by the user via the Frequency attribute.

Config Paths

ns3::FriisPropagationLossModel is accessible through the following paths with **Config::Set** and **Config::Connect**:

- /ChannelList/[i]/$ns3::WifiChannel/$ns3::YansWifiChannel/PropagationLossModel/$ns3::FriisPropagationLossModel
- /ChannelList/[i]/$ns3::YansWifiChannel/PropagationLossModel/$ns3::FriisPropagationLossModel
- /NodeList/[i]/DeviceList/[i]/$ns3::WifiNetDevice/Channel/$ns3::YansWifiChannel/PropagationLossModel/$ns3::FriisPropagationLossModel

Attributes

- **Frequency**: The carrier frequency (in Hz) at which propagation occurs (default is 5.15 GHz).
  - Set with class: **ns3::DoubleValue**
  - Underlying type: **double** -1.79769e+308:1.79769e+308
  - Initial value: 5.15e+09
- **SystemLoss**: The system loss
  - Set with class: **ns3::DoubleValue**
  - Underlying type: **double** -1.79769e+308:1.79769e+308
  - Initial value: 1
- **MinDistance**: The distance under which the propagation model refuses to give results (m)
  - Set with class: **ns3::DoubleValue**

- Underlying type: **double** -1.79769e+308:1.79769e+308
- Initial value: 0.5

5.2 Fixed RSS Loss Model:

Detailed Description

Return a constant received power level independent of the transmit power.

The received power is constant independent of the transmit power. The user must set received power level through the Rss attribute or public **SetRss()** method. Note that if this loss model is chained to other loss models via **SetNext()** method, it can only be the first loss model in such a chain, or else it will disregard the losses computed by loss models that precede it in the chain.

Config Paths

ns3::FixedRssLossModel is accessible through the following paths with **Config::Set** and **Config::Connect**:

- /ChannelList/[i]/$ns3::WifiChannel/$ns3::YansWifiChannel/PropagationLossModel/$ns3::FixedRssLossModel
- /ChannelList/[i]/$ns3::YansWifiChannel/PropagationLossModel/$ns3::FixedRssLossModel
- /NodeList/[i]/DeviceList/[i]/$ns3::WifiNetDevice/Channel/$ns3::YansWifiChannel/PropagationLossModel/$ns3::FixedRssLossModel

Attributes

- **Rss**: The fixed receiver Rss.
  - Set with class: **ns3::DoubleValue**
  - Underlying type: **double** -1.79769e+308:1.79769e+308
  - Initial value: -150

6. CODE

6.1 FRIIS PROPAGATION LOSS MODEL:

```
/*
*
* Network topology:
*
*   Ap    STA
*   *     *
*   |     |
*   n1     n2
*
*/

#include "ns3/config.h"
#include "ns3/string.h"
#include "ns3/log.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/mobility-helper.h"
#include "ns3/on-off-helper.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/mobility-model.h"
#include "ns3/packet-sink.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/tcp-westwood.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor.h"
#include "ns3/flow-monitor-helper.h"

NS_LOG_COMPONENT_DEFINE("Friis_Propagation_Loss_Model");

using namespace ns3;

Ptr<PacketSink> sink;     /* Pointer to the packet sink application */
uint64_t lastTotalRx = 0; /* The value of the last total received bytes */

void CalculateThroughput()
{
   /***************** Return the simulator's virtual time*************/
```

```
    Time now = Simulator::Now();

    /*************** Convert Application RX Packets to MBits. *************/
    double cur = (sink->GetTotalRx() - lastTotalRx) * (double)8 / 1e5;
    std::cout << now.GetSeconds() << "s: \t" << cur << " Mbit/s" << std::endl;

    lastTotalRx = sink->GetTotalRx();
    Simulator::Schedule(MilliSeconds(100), &CalculateThroughput);
}

int main(int argc, char *argv[])
{
    std::cout << "Using FriisPropagationLossModel \n" << std::endl;

    uint32_t payloadSize = 1472;          /* Transport layer payload size in bytes. */
    std::string dataRate = "100Mbps";     /* Application layer datarate. */
    std::string tcpVariant = "TcpNewReno"; /* TCP variant type. */
    std::string phyRate = "HtMcs7";       /* Physical layer bitrate. */
    double simulationTime = 5;            /* Simulation time in seconds. */

    // LogComponentEnable("OnOffApplication", LOG_INFO);
    // LogComponentEnable("PacketSink", LOG_INFO);
    // LogComponentEnable("WifiMac", LOG_INFO);
    // LogComponentEnable("WifiPhy", LOG_INFO);

    tcpVariant = std::string("ns3::") + tcpVariant;
    TypeId tcpTid;
    NS_ABORT_MSG_UNLESS(TypeId::LookupByNameFailSafe(tcpVariant, &tcpTid),
"TypeId " << tcpVariant << " not found");
    Config::SetDefault("ns3::TcpL4Protocol::SocketType",
TypeIdValue(TypeId::LookupByName(tcpVariant)));

    /************* Configure TCP Options ********************/
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));

    WifiMacHelper wifiMac;
    WifiHelper wifiHelper;
    wifiHelper.SetStandard(WIFI_STANDARD_80211n_2_4GHZ);

    /***************** Set up Legacy Channel *****************/
    YansWifiChannelHelper wifiChannel;
    wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
    wifiChannel.AddPropagationLoss("ns3::FriisPropagationLossModel");
```

```
/****************** Setup Physical Layer ******************/
YansWifiPhyHelper wifiPhy;
wifiPhy.SetChannel(wifiChannel.Create());
wifiPhy.SetErrorRateModel("ns3::YansErrorRateModel");
wifiHelper.SetRemoteStationManager("ns3::ConstantRateWifiManager",
                                   "DataMode", StringValue(phyRate),
                                   "ControlMode", StringValue("HtMcs0"));

/****************** Create Nodes ********************/
NodeContainer networkNodes;
networkNodes.Create(2);
Ptr<Node> apWifiNode = networkNodes.Get(0);
Ptr<Node> staWifiNode = networkNodes.Get(1);

/****************** Configure Access Point (AP)***************/
Ssid ssid = Ssid("network");
wifiMac.SetType("ns3::ApWifiMac",
                "Ssid", SsidValue(ssid));

NetDeviceContainer apDevice;
apDevice = wifiHelper.Install(wifiPhy, wifiMac, apWifiNode);

/******************** Configure Station Node (STA)*************/
wifiMac.SetType("ns3::StaWifiMac",
                "Ssid", SsidValue(ssid));

NetDeviceContainer staDevices;
staDevices = wifiHelper.Install(wifiPhy, wifiMac, staWifiNode);

/********************** Mobility model ***************/
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>();
positionAlloc->Add(Vector(0.0, 0.0, 0.0));
positionAlloc->Add(Vector(100.0, 100.0, 0.0));

mobility.SetPositionAllocator(positionAlloc);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(apWifiNode);
mobility.Install(staWifiNode);

/****************** Internet stack ********************/
InternetStackHelper stack;
stack.Install(networkNodes);
```

```
    Ipv4AddressHelper address;
    address.SetBase("10.0.0.0", "255.255.255.0");
    Ipv4InterfaceContainer apInterface;
    apInterface = address.Assign(apDevice);
    Ipv4InterfaceContainer staInterface;
    staInterface = address.Assign(staDevices);

    /* Populate routing table */
    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    /*************** Install TCP Receiver on the access point *************/
    PacketSinkHelper sinkHelper("ns3::TcpSocketFactory",
InetSocketAddress(Ipv4Address::GetAny(), 9));
    ApplicationContainer sinkApp = sinkHelper.Install(apWifiNode);
    sink = StaticCast<PacketSink>(sinkApp.Get(0));

    /*************** Install TCP/UDP Transmitter on the station************/
    OnOffHelper server("ns3::TcpSocketFactory",
(InetSocketAddress(apInterface.GetAddress(0), 9)));
    server.SetAttribute("PacketSize", UintegerValue(payloadSize));
    server.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
    server.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
    server.SetAttribute("DataRate", DataRateValue(DataRate(dataRate)));
    ApplicationContainer serverApp = server.Install(staWifiNode);

    /******************** Start Applications ***************/
    sinkApp.Start(Seconds(0.0));
    serverApp.Start(Seconds(1.0));
    Simulator::Schedule(Seconds(1.1), &CalculateThroughput);

/********************Enable Traces*********************/


    wifiPhy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);
    wifiPhy.EnablePcap("Friis_AP", apDevice);
    wifiPhy.EnablePcap("Friis_STA", staDevices);

    // AsciiTraceHelper ascii;
    // WifiPhy.EnableAsciiAll(ascii.CreateFileStream("trace1.tr"));

    AsciiTraceHelper ascii;
    Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream("Friis.tr");
    wifiPhy.EnableAsciiAll (stream);
```

```
    AnimationInterface anim("Friis.xml");
    AnimationInterface::SetConstantPosition(apWifiNode, 0.0, 0.0);
    AnimationInterface::SetConstantPosition(staWifiNode, 100.0, 100.0);

  /***************FlowMonitor*************************/
  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowHelper;
  flowMonitor = flowHelper.InstallAll();

    /****************** Start Simulation ******************/
    Simulator::Stop(Seconds(simulationTime + 1));
    Simulator::Run();

 /********create xml file from flowmonitor************************/
 flowMonitor->SerializeToXmlFile("flowFriss.xml", true, true);

    double averageThroughput = ((sink->GetTotalRx() * 8) / (1e6 * simulationTime));

    Simulator::Destroy();

    // if (averageThroughput < 50)
    // {
    //     NS_LOG_ERROR("Obtained throughput is not in the expected boundaries!");
    //     exit(1);
    // }

    std::cout << "\nAverage throughput: " << averageThroughput << " Mbit/s" << std::endl;
    return 0;
}
```

6.2 FIXED RSS LOSS MODEL:

```
/*
*
* Network topology:
*
*   Ap    STA
*   *      *
*   |      |
*   n1     n2
*
*/

#include "ns3/config.h"
#include "ns3/string.h"
```


```
#include "ns3/log.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/mobility-helper.h"
#include "ns3/on-off-helper.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/mobility-model.h"
#include "ns3/packet-sink.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/tcp-westwood.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor.h"
#include "ns3/flow-monitor-helper.h"

NS_LOG_COMPONENT_DEFINE("Fixed_Rss_Loss_Model");

using namespace ns3;

Ptr<PacketSink> sink;     /* Pointer to the packet sink application */
uint64_t lastTotalRx = 0; /* The value of the last total received bytes */

void CalculateThroughput()
{
/************* Return the simulator's virtual time. *****************/
   Time now = Simulator::Now();

   /* Convert Application RX Packets to MBits. */
   double cur = (sink->GetTotalRx() - lastTotalRx) * (double)8 / 1e5;
   std::cout << now.GetSeconds() << "s: \t" << cur << " Mbit/s" << std::endl;

   lastTotalRx = sink->GetTotalRx();
   Simulator::Schedule(MilliSeconds(100), &CalculateThroughput);
}

int main(int argc, char *argv[])
{
   std::cout << "Using FixedRssLossModel with receive strength fixed at -50dB\n" << std::endl;

   uint32_t payloadSize = 1472;          /* Transport layer payload size in bytes. */
   std::string dataRate = "100Mbps";     /* Application layer datarate. */
   std::string tcpVariant = "TcpNewReno"; /* TCP variant type. */
```

```
    std::string phyRate = "HtMcs7";        /* Physical layer bitrate. */
    double simulationTime = 5;             /* Simulation time in seconds. */

    // LogComponentEnable("OnOffApplication", LOG_INFO);
    // LogComponentEnable("PacketSink", LOG_INFO);
    // LogComponentEnable("WifiMac", LOG_INFO);
    // LogComponentEnable("WifiPhy", LOG_INFO);

    tcpVariant = std::string("ns3::") + tcpVariant;
    TypeId tcpTid;
    NS_ABORT_MSG_UNLESS(TypeId::LookupByNameFailSafe(tcpVariant, &tcpTid),
"TypeId " << tcpVariant << " not found");
    Config::SetDefault("ns3::TcpL4Protocol::SocketType",
TypeIdValue(TypeId::LookupByName(tcpVariant)));

/************** Configure TCP Options ********************/
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));

    WifiMacHelper wifiMac;
    WifiHelper wifiHelper;
    wifiHelper.SetStandard(WIFI_STANDARD_80211n_2_4GHZ);

/*************** Set up Legacy Channel ********************/
    YansWifiChannelHelper wifiChannel;
    wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
    wifiChannel.AddPropagationLoss("ns3::FixedRssLossModel", "Rss", DoubleValue(-50));

/*************** Setup Physical Layer ********************/
    YansWifiPhyHelper wifiPhy;
    wifiPhy.SetChannel(wifiChannel.Create());
    wifiPhy.SetErrorRateModel("ns3::YansErrorRateModel");
    wifiHelper.SetRemoteStationManager("ns3::ConstantRateWifiManager",
                                       "DataMode", StringValue(phyRate),
                                       "ControlMode", StringValue("HtMcs0"));

/************** Create Nodes ************************/
    NodeContainer networkNodes;
    networkNodes.Create(2);
    Ptr<Node> apWifiNode = networkNodes.Get(0);
    Ptr<Node> staWifiNode = networkNodes.Get(1);

/***************** Configure Access Point (AP)*****************/
    Ssid ssid = Ssid("network");
```

```
    wifiMac.SetType("ns3::ApWifiMac",
                    "Ssid", SsidValue(ssid));

    NetDeviceContainer apDevice;
    apDevice = wifiHelper.Install(wifiPhy, wifiMac, apWifiNode);

/***************** Configure Station Node (STA)***************/
    wifiMac.SetType("ns3::StaWifiMac",
                    "Ssid", SsidValue(ssid));

    NetDeviceContainer staDevices;
    staDevices = wifiHelper.Install(wifiPhy, wifiMac, staWifiNode);

/******************* Mobility model ********************/
    MobilityHelper mobility;
    Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>();
    positionAlloc->Add(Vector(0.0, 0.0, 0.0));
    positionAlloc->Add(Vector(100.0, 100.0, 0.0));

    mobility.SetPositionAllocator(positionAlloc);
    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(apWifiNode);
    mobility.Install(staWifiNode);

 /***************** Internet stack *********************/
    InternetStackHelper stack;
    stack.Install(networkNodes);

    Ipv4AddressHelper address;
    address.SetBase("10.0.0.0", "255.255.255.0");
    Ipv4InterfaceContainer apInterface;
    apInterface = address.Assign(apDevice);
    Ipv4InterfaceContainer staInterface;
    staInterface = address.Assign(staDevices);

/****************** Populate routing table ***************/
    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

/************** Install TCP Receiver on the access point *************/
    PacketSinkHelper sinkHelper("ns3::TcpSocketFactory",
InetSocketAddress(Ipv4Address::GetAny(), 9));
    ApplicationContainer sinkApp = sinkHelper.Install(apWifiNode);
    sink = StaticCast<PacketSink>(sinkApp.Get(0));
```

```
/****************** Install TCP/UDP Transmitter on the station ***********/
   OnOffHelper server("ns3::TcpSocketFactory",
(InetSocketAddress(apInterface.GetAddress(0), 9)));
   server.SetAttribute("PacketSize", UintegerValue(payloadSize));
   server.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
   server.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
   server.SetAttribute("DataRate", DataRateValue(DataRate(dataRate)));
   ApplicationContainer serverApp = server.Install(staWifiNode);

/****************** Start Applications *****************/
   sinkApp.Start(Seconds(0.0));
   serverApp.Start(Seconds(1.0));
   Simulator::Schedule(Seconds(1.1), &CalculateThroughput);

/****************** Enable Traces *********************/

   wifiPhy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);
   wifiPhy.EnablePcap("FixedRSS_AP", apDevice);
   wifiPhy.EnablePcap("FixedRSS_STA", staDevices);

   // AsciiTraceHelper ascii;
   // WifiPhy.EnableAsciiAll(ascii.CreateFileStream("trace1.tr"));

   AsciiTraceHelper ascii;
   Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream("FixedRSS.tr");
   wifiPhy.EnableAsciiAll (stream);

   AnimationInterface anim("FixedRSS.xml");
   AnimationInterface::SetConstantPosition(apWifiNode, 0.0, 0.0);
   AnimationInterface::SetConstantPosition(staWifiNode, 100.0, 100.0);
 /********************* Flow monitor*****************/

 Ptr<FlowMonitor> flowMonitor;
 FlowMonitorHelper flowHelper;
 flowMonitor = flowHelper.InstallAll();

 /******************* Start Simulation ***************/
   Simulator::Stop(Seconds(simulationTime + 1));
   Simulator::Run();
  /**********create xml file from flowMonitor******************/
   flowMonitor->SerializeToXmlFile("flowFixedRSS.xml", true, true);
```

```
    double averageThroughput = ((sink->GetTotalRx() * 8) / (1e6 * simulationTime));

    Simulator::Destroy();

    // if (averageThroughput < 50)
    // {
    //     NS_LOG_ERROR("Obtained throughput is not in the expected boundaries!");
    //     exit(1);
    // }

    std::cout << "\nAverage throughput: " << averageThroughput << " Mbit/s" << std::endl;
    return 0;
}
```

7. OUTPUT

Below are the snapshots of the terminal output of the above mentioned codes.

7.1 FRIIS PROPOGATION LOSS MODEL:

```
rippen@ubuntu20-VirtualBox:~/Downloads/ns-allinone-3.33/ns-3.33$ ./waf --run scratch/lossModel_1.cc
Waf: Entering directory `/home/rippen/Downloads/ns-allinone-3.33/ns-3.33/build'
Waf: Leaving directory `/home/rippen/Downloads/ns-allinone-3.33/ns-3.33/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2.942s)
Using FriisPropagationLossModel

1.1s:   8.47872 Mbit/s
1.2s:   9.53856 Mbit/s
1.3s:   17.3107 Mbit/s
1.4s:   16.8397 Mbit/s
1.5s:   16.9574 Mbit/s
1.6s:   13.8957 Mbit/s
1.7s:   14.4845 Mbit/s
1.8s:   17.664 Mbit/s
1.9s:   12.9536 Mbit/s
2s:     18.0173 Mbit/s
2.1s:   16.3686 Mbit/s
2.2s:   15.4266 Mbit/s
2.3s:   14.9555 Mbit/s
2.4s:   12.9536 Mbit/s
2.5s:   16.1331 Mbit/s
2.6s:   10.7162 Mbit/s
```

```
2.7s:   13.7779 Mbit/s
2.8s:   8.94976 Mbit/s
2.9s:   10.4806 Mbit/s
3s:     14.3667 Mbit/s
3.1s:   14.0134 Mbit/s
3.2s:   12.7181 Mbit/s
3.3s:   14.8378 Mbit/s
3.4s:   15.3088 Mbit/s
3.5s:   14.4845 Mbit/s
3.6s:   14.1312 Mbit/s
3.7s:   14.72 Mbit/s
3.8s:   12.8358 Mbit/s
3.9s:   11.305 Mbit/s
4s:     14.0134 Mbit/s
4.1s:   14.1312 Mbit/s
4.2s:   13.8957 Mbit/s
4.3s:   14.4845 Mbit/s
4.4s:   15.4266 Mbit/s
4.5s:   14.4845 Mbit/s
4.6s:   18.0173 Mbit/s
4.7s:   13.7779 Mbit/s
4.8s:   17.7818 Mbit/s
4.9s:   16.8397 Mbit/s
5s:     13.4246 Mbit/s
5.1s:   14.1312 Mbit/s
5.2s:   16.6042 Mbit/s
5.3s:   18.7238 Mbit/s
5.4s:   18.2528 Mbit/s
5.5s:   12.9536 Mbit/s
5.6s:   12.9536 Mbit/s
5.7s:   19.7837 Mbit/s
5.8s:   14.1312 Mbit/s
5.9s:   20.8435 Mbit/s

Average throughput: 14.6493 Mbit/s
```

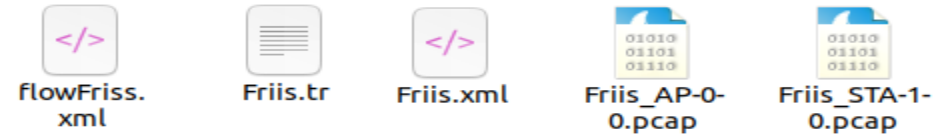
7.2 FIXED RSS LOSS MODEL:

```
rippen@ubuntu20-VirtualBox:~/Downloads/ns-allinone-3.33$ cd ns-3.33
rippen@ubuntu20-VirtualBox:~/Downloads/ns-allinone-3.33/ns-3.33$ ./waf --run scratch/lossModel_2.cc
Waf: Entering directory `/home/rippen/Downloads/ns-allinone-3.33/ns-3.33/build'
[2737/2789] Compiling scratch/lossModel_2.cc
[2738/2789] Compiling scratch/mysecond_2.cc
[2740/2789] Compiling scratch/subdir/scratch-simulator-subdir.cc
[2748/2789] Linking build/scratch/subdir/subdir
[2749/2789] Linking build/scratch/lossModel_2
[2750/2789] Linking build/scratch/mysecond_2
Waf: Leaving directory `/home/rippen/Downloads/ns-allinone-3.33/ns-3.33/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (9.297s)
Using FixedRssLossModel with receive strength fixed at -50dB

1.1s:   44.0422 Mbit/s
1.2s:   56.5248 Mbit/s
1.3s:   55.2294 Mbit/s
1.4s:   55.3472 Mbit/s
1.5s:   52.7565 Mbit/s
1.6s:   54.7584 Mbit/s
1.7s:   52.6387 Mbit/s
1.8s:   49.2237 Mbit/s
1.9s:   49.6947 Mbit/s
2s:     48.7526 Mbit/s
2.1s:   51.2256 Mbit/s
2.2s:   48.5171 Mbit/s
2.3s:   51.6966 Mbit/s
2.4s:   55.1117 Mbit/s
2.5s:   52.2854 Mbit/s
2.6s:   55.2294 Mbit/s
2.7s:   52.0499 Mbit/s
2.8s:   52.0499 Mbit/s
2.9s:   48.7526 Mbit/s
3s:     54.9939 Mbit/s
3.1s:   55.2294 Mbit/s
3.2s:   52.1677 Mbit/s
3.3s:   55.936 Mbit/s
```

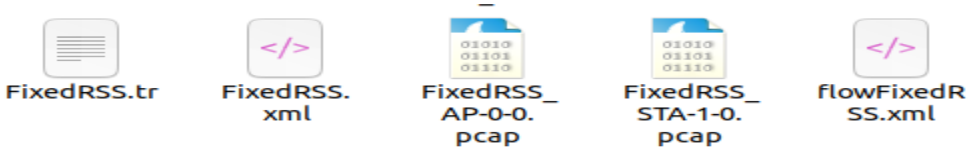
```
3.4s:    42.9824 Mbit/s
3.5s:    55.3472 Mbit/s
3.6s:    44.8666 Mbit/s
3.7s:    55.5827 Mbit/s
3.8s:    55.2294 Mbit/s
3.9s:    52.2854 Mbit/s
4s:      55.465 Mbit/s
4.1s:    51.6966 Mbit/s
4.2s:    48.5171 Mbit/s
4.3s:    55.2294 Mbit/s
4.4s:    55.465 Mbit/s
4.5s:    52.0499 Mbit/s
4.6s:    55.5827 Mbit/s
4.7s:    46.7507 Mbit/s
4.8s:    55.1117 Mbit/s
4.9s:    55.465 Mbit/s
5s:      45.3376 Mbit/s
5.1s:    49.1059 Mbit/s
5.2s:    54.6406 Mbit/s
5.3s:    49.4592 Mbit/s
5.4s:    48.7526 Mbit/s
5.5s:    44.8666 Mbit/s
5.6s:    45.1021 Mbit/s
5.7s:    46.3974 Mbit/s
5.8s:    55.3472 Mbit/s
5.9s:    55.465 Mbit/s

Average throughput: 51.8332 Mbit/s
```

8. ANALYZING OUTPUTS FROM DIFFERENT FILES

The following files were created when Program 1 aka lossModel_1.cc (Friis Propagation Loss Model) was compiled.



Below are the files created after the compilation of program 2, i.e. lossModel_2.cc (Fixed RSS Loss model).



8.1 XML file using NetAnim

To analyse the node structure using animation, in NetAnim (Network Animator), you need to make xml file for your C++ code in ns-3.

This can be formed using the below written code lines in end of the C++ programs as shown. The arguments of SetConstantPosition function show the coordinates of nodes to be shown on the grid in the Network Animator.

lossModel_1.cc (Friis Propagation Loss Model):

```
167     AnimationInterface anim("Friis.xml");
168     AnimationInterface::SetConstantPosition(apWifiNode, 0.0, 0.0);
169     AnimationInterface::SetConstantPosition(staWifiNode, 100.0, 100.0);
```

lossModel_2.cc (Fixed RSS Loss Model):

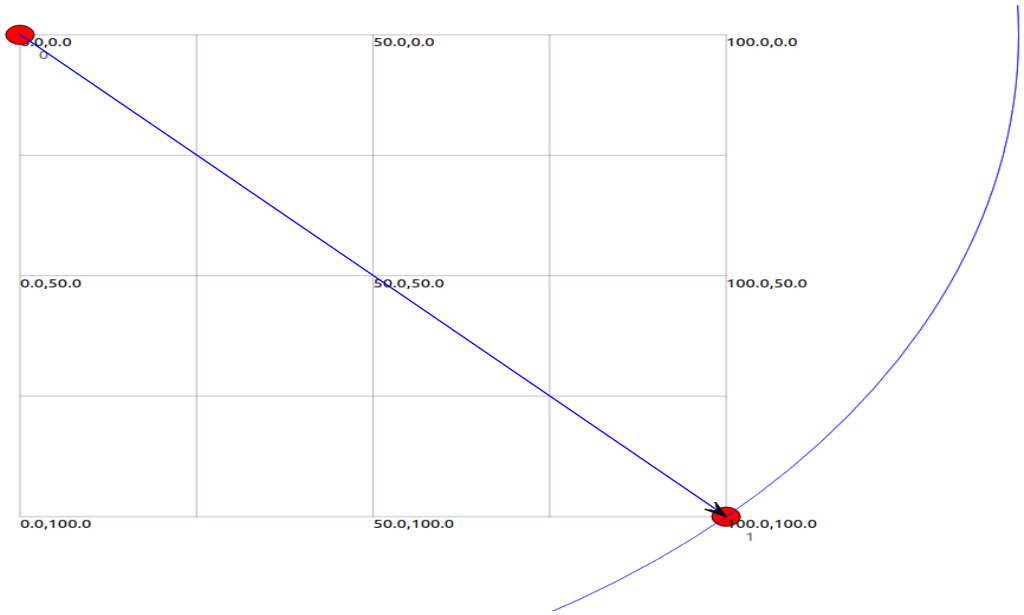
```
166     AnimationInterface anim("FixedRSS.xml");
167     AnimationInterface::SetConstantPosition(apWifiNode, 0.0, 0.0);
168     AnimationInterface::SetConstantPosition(staWifiNode, 100.0, 100.0);
```

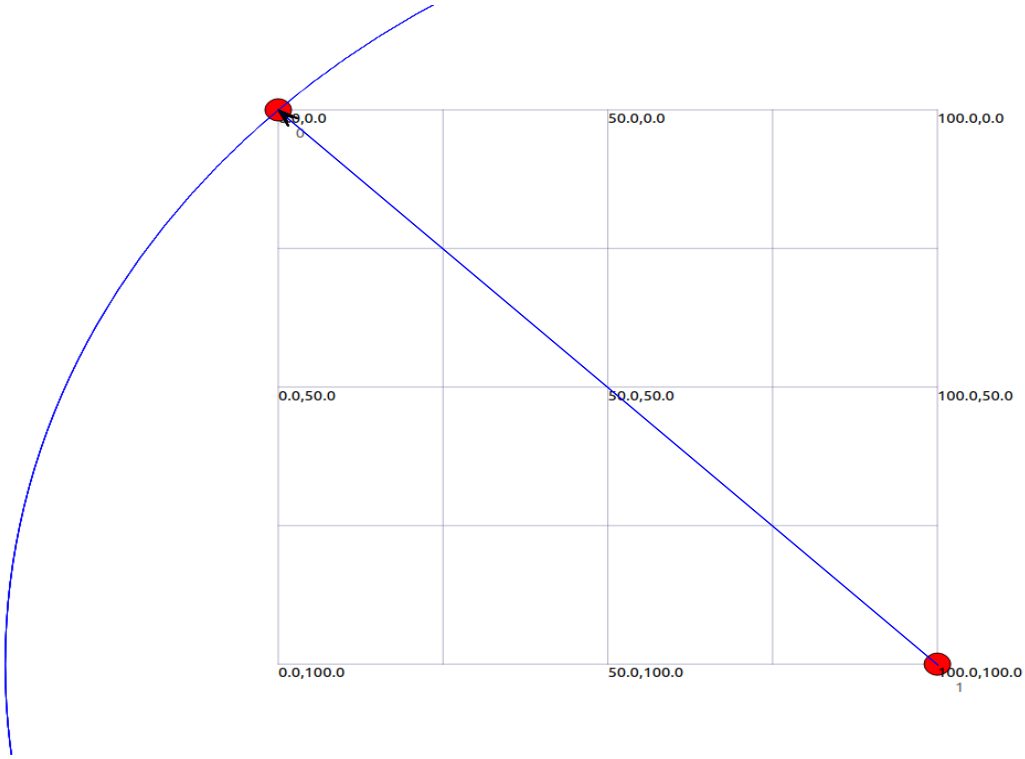
Now to run xml file of your C++ program in NetAnim, follow the below written steps, i.e., go in the netanim-3.108 directory, and write ./NetAnim command as shown:

```
rippen@ubuntu20-VirtualBox:~/Downloads/ns-allinone-3.33$ cd netanim-3.108
rippen@ubuntu20-VirtualBox:~/Downloads/ns-allinone-3.33/netanim-3.108$ ./NetAnim
```

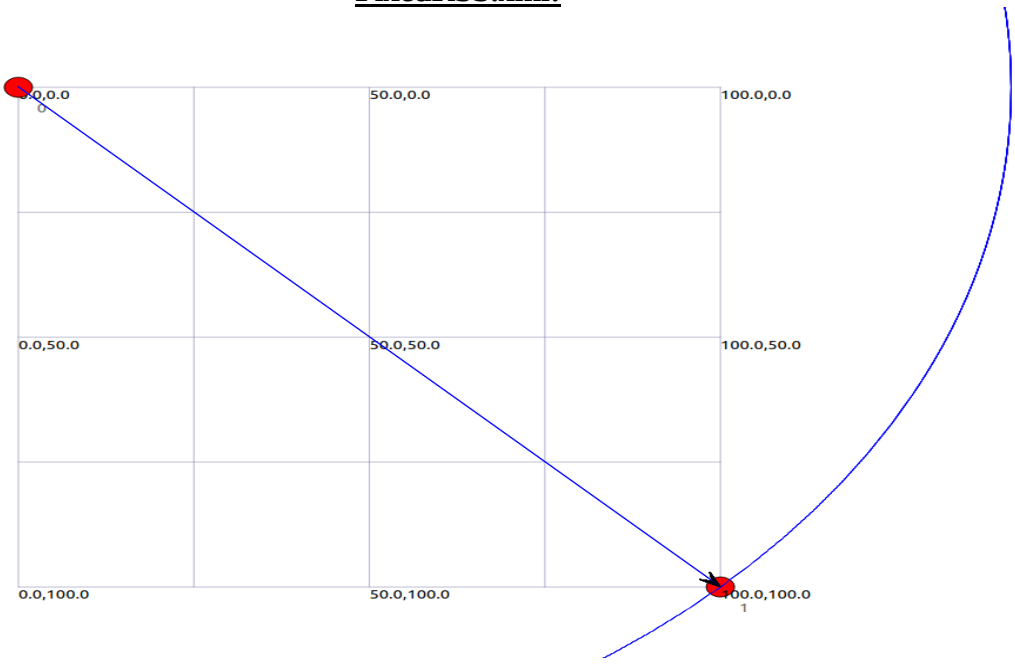
The NetAnim GUI will open, just select your xml file from the directory, and press play button. The animation will play. The screenshots of node 0 (ap Wifi Node) sending packet to the node 1 (sta wifi Node) and node 1 sending acknowledgement back to the node 0 are shown below.

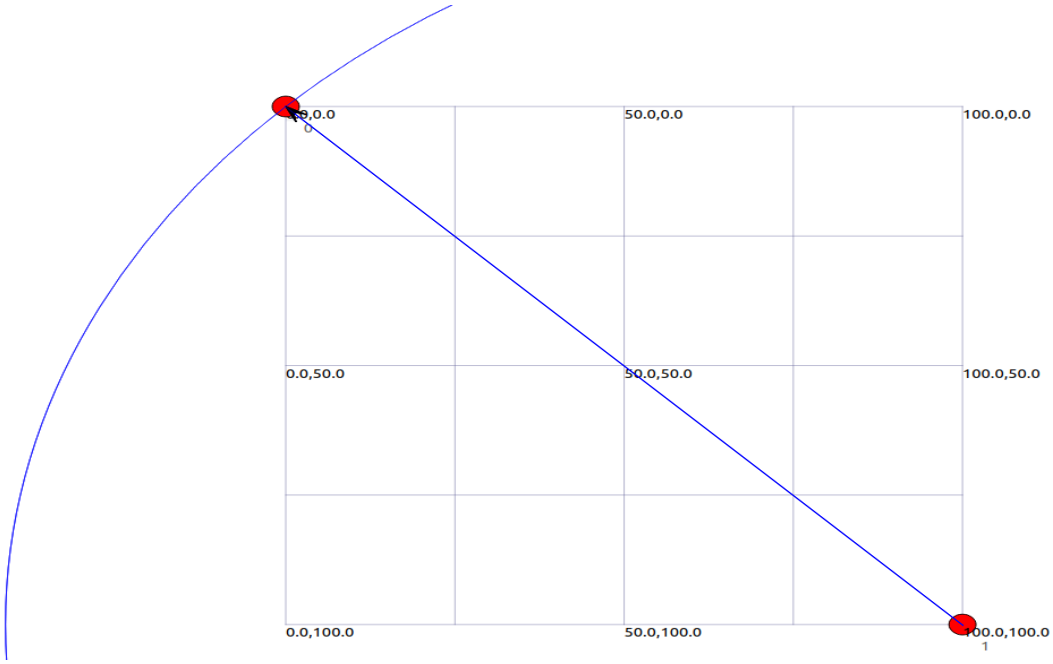
Friis.xml:





FixedRSS.xml:





8.2 .tr file

The ASCII trace files is made using the below mentioned commands:

lossModel_1.cc (Friis Propagation Loss Model):

```
163     AsciiTraceHelper ascii;
164     Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream("Friis.tr");
165     wifiPhy.EnableAsciiAll (stream);
```

lossModel_2.cc (Fixed RSS Loss Model):

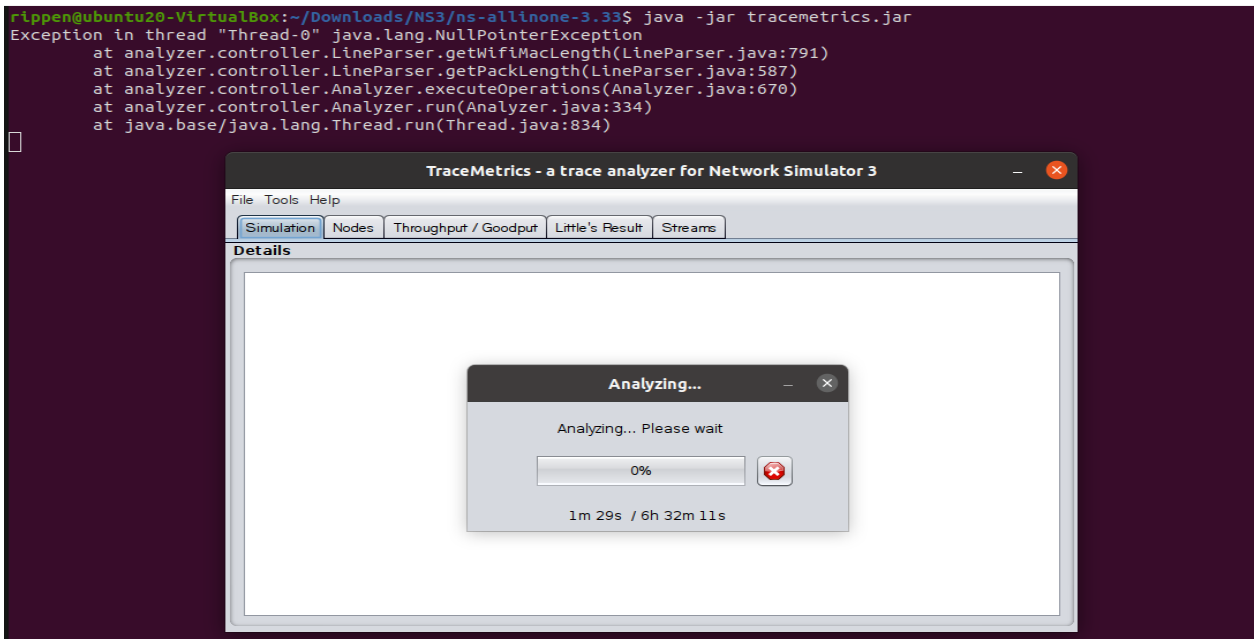
```
162     AsciiTraceHelper ascii;
163     Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream("FixedRSS.tr");
164     wifiPhy.EnableAsciiAll (stream);
```

To run TraceMetrics - trace analyzer, run the following command in the directory where you have unzipped/extracted the tracemetrics.zip file.

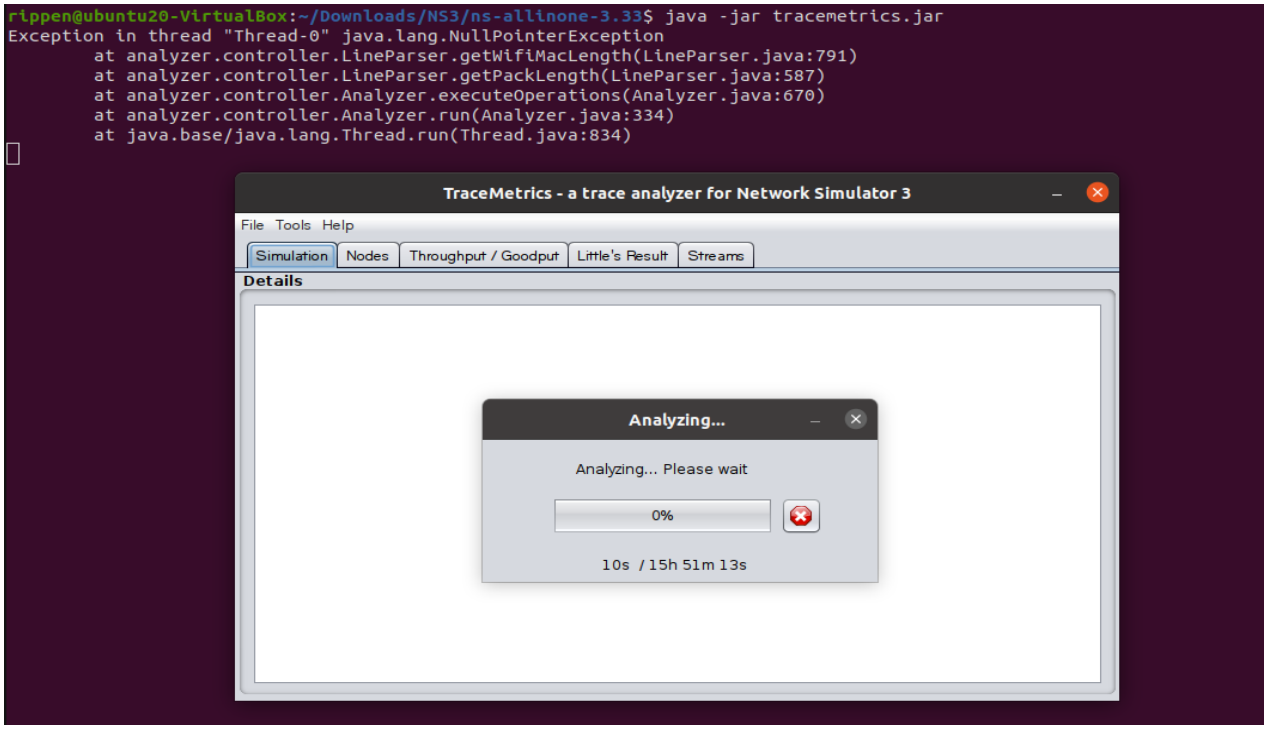
$ java -jar tracemetrics.jar

The GUI of TraceMetrics - a trace analyzer for NS3 will open, you will have to select the trace file created by you in the program using **File** > **Open** from the Menu bar. And then, all the details of Simulation, Nodes, Throughput/Goodput Little's Result, and Streams will be available in the trace analyzer. You can view that by clicking on the respective button.
However in our programs , analysis of .tr files seems taking very much time.

Friis.tr:



FixedRSS.tr:



8.3 .pcap file

The below written commands have been used in the above programs to create pcap files for all the nodes.

lossModel_1.cc (Friis Propagation Loss Model):

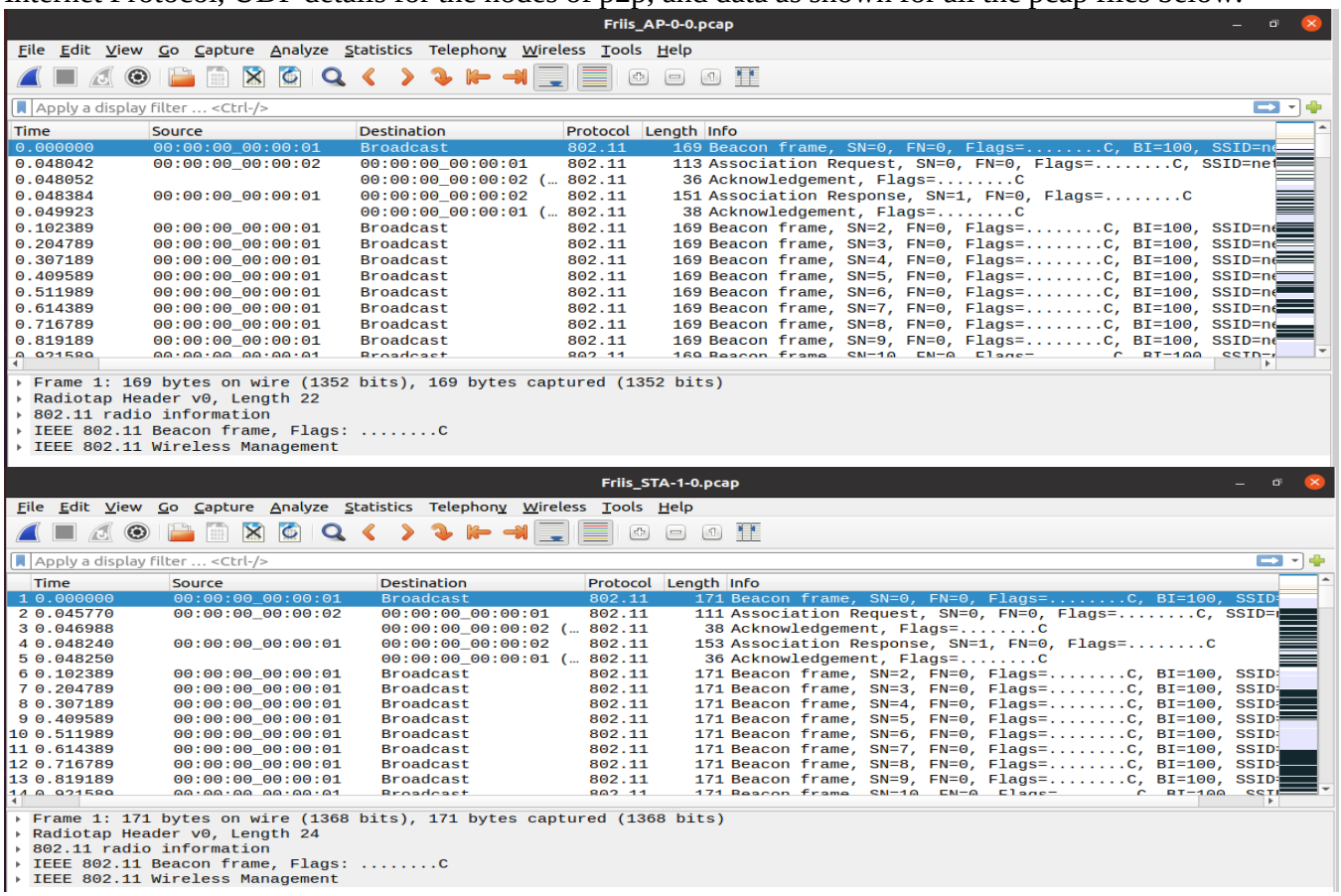
```
156     wifiPhy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);
157     wifiPhy.EnablePcap("Friis_AP", apDevice);
158     wifiPhy.EnablePcap("Friis_STA", staDevices);
```

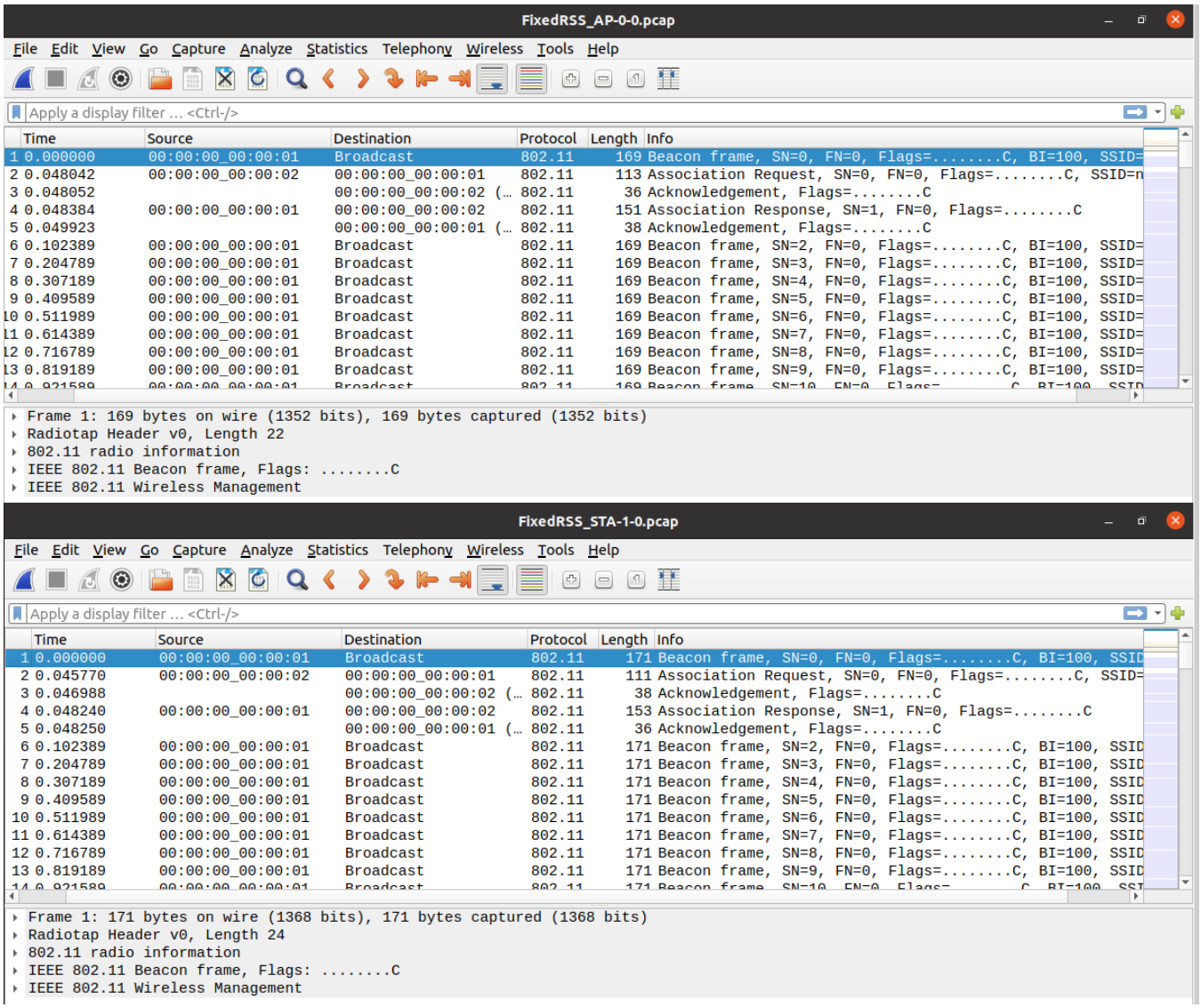
lossModel_2.cc (Fixed RSS Loss Model):

```
155     wifiPhy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);
156     wifiPhy.EnablePcap("FixedRSS_AP", apDevice);
157     wifiPhy.EnablePcap("FixedRSS_STA", staDevices);
```

To analyse pcap file via Wireshark and tcpdump,simply write **wireshark** on the terminal and open Wireshark GUI and then, click **File** > **Open File**. And, then choose the file from the directory, and press enter.

The Wireshark window will show you different analysis of the respective pcap file, which includes Frame, Ethernet details and ARP for the nodes of CSMA ,and Frame, Point-To-Point Protocol, Internet Protocol, UDP details for the nodes of p2p, and data as shown for all the pcap files below.





8.4 XML file using Flow Monitor

The following commands have been used to make the xml files using flow monitor.

lossModel_1.cc (Friis Propagation Loss Model):

```
173  /*****************************************FlowMonitor**********
174  Ptr<FlowMonitor> flowMonitor;
175  FlowMonitorHelper flowHelper;
176  flowMonitor = flowHelper.InstallAll();

182 /**********************create xml file from flowmonitor*********************************
183 flowMonitor->SerializeToXmlFile("flowFriss.xml", true, true);
```

lossModel_2.cc (Fixed RSS Loss Model):

```
173  Ptr<FlowMonitor> flowMonitor;
174  FlowMonitorHelper flowHelper;
175  flowMonitor = flowHelper.InstallAll();
180   /**************************create xml file from flowMonitor*****************
181     flowMonitor->SerializeToXmlFile("flowFixedRSS.xml", true, true);
```

The following are the .xml files created after running the above code:

flowFixedRSS.xml

```
134    <FlowProbe index="0">
135      <FlowStats  flowId="1" packets="22010" bytes="33540300" delayFromFirstProbeSum="+2.48839e+11ns" >
136      </FlowStats>
137      <FlowStats  flowId="2" packets="11005" bytes="572264" delayFromFirstProbeSum="+0ns" >
138      </FlowStats>
139    </FlowProbe>
140    <FlowProbe index="1">
141    </FlowProbe>
142    <FlowProbe index="2">
143      <FlowStats  flowId="1" packets="22070" bytes="33631740" delayFromFirstProbeSum="+0ns" >
144      </FlowStats>
145      <FlowStats  flowId="2" packets="10991" bytes="571536" delayFromFirstProbeSum="+6.15589e+10ns" >
146      </FlowStats>
147    </FlowProbe>
148    <FlowProbe index="3">
149    </FlowProbe>
150  </FlowProbes>
151 </FlowMonitor>
```

flowFriis.xml

```
177   </FlowStats>
178   <Ipv4FlowClassifier>
179     <Flow flowId="2" sourceAddress="10.0.0.1" destinationAddress="10.0.0.2" protocol="6" sourcePort="9" destinationPort="49153">
180       <Dscp value="0x0" packets="3271" />
181     </Flow>
182     <Flow flowId="1" sourceAddress="10.0.0.2" destinationAddress="10.0.0.1" protocol="6" sourcePort="49153" destinationPort="9">
183       <Dscp value="0x0" packets="6319" />
184     </Flow>
185   </Ipv4FlowClassifier>
186   <Ipv6FlowClassifier>
187   </Ipv6FlowClassifier>
188   <FlowProbes>
189     <FlowProbe index="0">
190       <FlowStats  flowId="1" packets="6256" bytes="9531204" delayFromFirstProbeSum="+1.38403e+11ns" >
191       </FlowStats>
192       <FlowStats  flowId="2" packets="3271" bytes="172104" delayFromFirstProbeSum="+0ns" >
193       </FlowStats>
194     </FlowProbe>
195     <FlowProbe index="1">
196     </FlowProbe>
197     <FlowProbe index="2">
198       <FlowStats  flowId="1" packets="6319" bytes="9627216" delayFromFirstProbeSum="+0ns" >
199       </FlowStats>
200       <FlowStats  flowId="2" packets="3270" bytes="172052" delayFromFirstProbeSum="+1.03135e+10ns" >
201       </FlowStats>
```

To analyse the output of the xml files first copy the python file "**flowmon-parse-result.py**" from **ns-3.33**>**src**>**flow-monitor**>**examples** to **ns-3.33**>**scratch** and run the following commands in the terminal.

$ python3 scratch/flowmon-parse-results.py flowFriss.xml
$ python3 scratch/flowmon-parse-results.py flowFixedRSS.xml

8.5 Graphical representation

The output of both the models were taken in different txt files and data file contains the data (output of both the programs) and .plt files were used to create the graphs after writing the following command in ns3 folder:

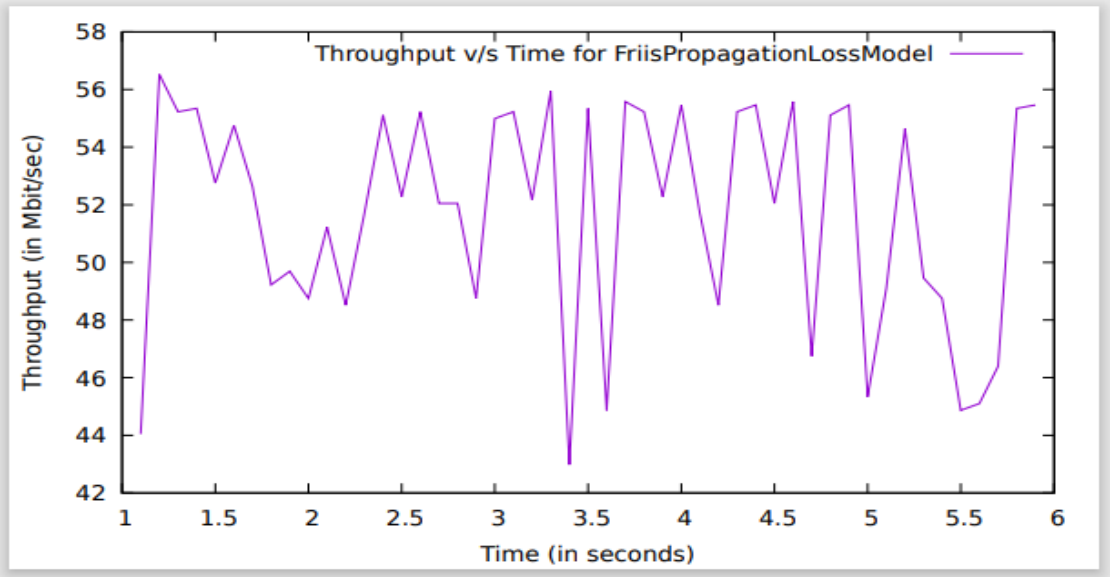
$ gnuplot friiss.plt
$ gnuplot fixedRSS.plt

.data file:

```
File  Edit  Format  View  Help
1.1 44.0422 8.47872
1.2 56.5248 9.53856
1.3 55.2294 17.3107
1.4 55.3472 16.8397
1.5 52.7565 16.9574
1.6 54.7584 13.8957
1.7 52.6387 14.4845
1.8 49.2237 17.664
1.9 49.6947 12.9536
2 48.7526 18.0173
2.1 51.2256 16.3686
2.2 48.5171 15.4266
2.3 51.6966 14.9555
2.4 55.1117 12.9536
2.5 52.2854 16.1331
2.6 55.2294 10.7162
2.7 52.0499 13.7779
2.8 52.0499 8.94976
2.9 48.7526 10.4806
3 54.9939 14.0134
3.1 55.2294 14.0134
3.2 52.1677 12.7181
```

friis.plt

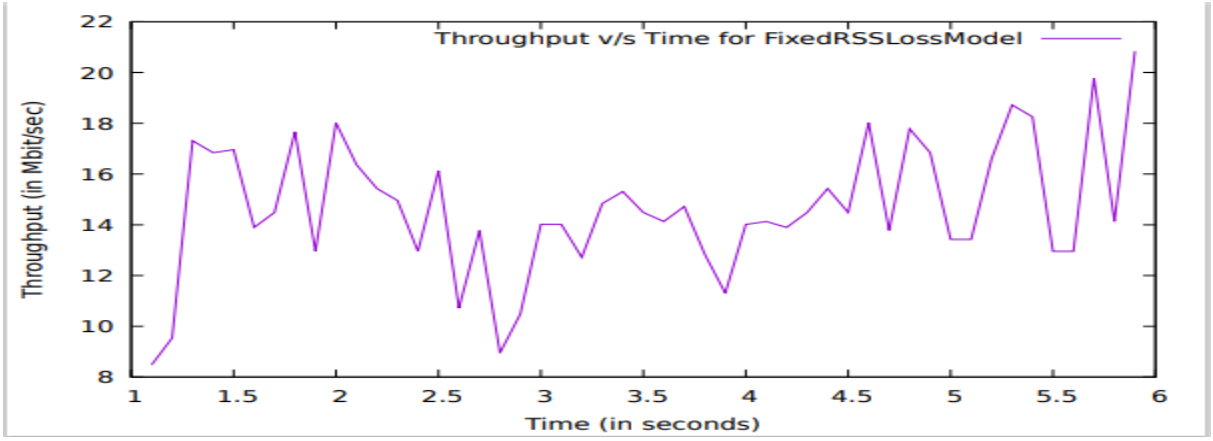
```
File  Edit  Format  View  Help
set terminal pdf
set output "acn_1.pdf"
set xlabel "Time (in seconds)"
set ylabel "Throughput (in Mbit/sec)"
plot "acn_proj_graph.data" using 1:2 with lines title "Throughput v/s Time for FriisPropagationLossModel"
```



fixedRSS.plt:

File Edit Format View Help

```
set terminal pdf
set output "acn_.pdf"
set xlabel "Time (in seconds)"
set ylabel "Throughput (in Mbit/sec)"
plot "acn_proj_graph.data" using 1:3 with lines title "Throughput v/s Time for FixedRSSLossModel"
```

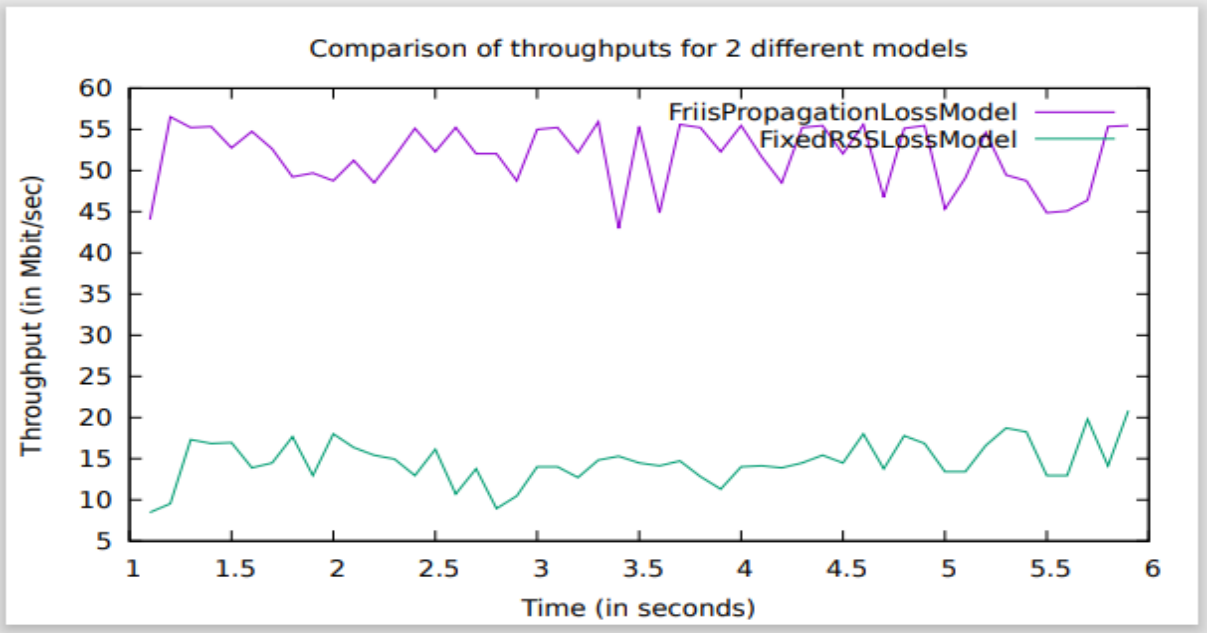


compare.plt:

File Edit Format View Help

```
set terminal pdf
set output "acn_compare.pdf"
set xlabel "Time (in seconds)"
set ylabel "Throughput (in Mbit/sec)"
set title "Comparison of throughputs for 2 different models"
plot "acn_proj_graph.data" using 1:2 title "FriisPropagationLossModel" with lines,\
"acn_proj_graph.data" using 1:3 title "FixedRSSLossModel" with lines
```

COMPARISON PLOT



The average throughput of Friis Propagation Loss Model is 14.6493 Mbit/s and that of Fixed RSS Loss Model 51.8332 Mbit/s.