# ET4394 Wireless Networking- NS3 Assignment
# **Wireless Network Performance Analysis**

Parul Gupta
4505123,
Delft University of Technology

May 8, 2016

## Contents

## 1 Introduction

Wireless Networks are based on IEEE 802.11 use CSMA (Carrier Sense Multiple Access protocol) . Wifi communication is half-duplex shared media configuration, where all stations transmit and receive on the same radio channel. While there are several advantages like lower installation complexities and flexibility in reception, WLANs suffer from problems like lower bandwidth and channel capacity.Thus it should be noted that while deploying a wireless network Client capacity, channel utilization, signal quality, and reliability are much more important factors.
In the basic functioning, the station computers or Wifi clients connect to an AP over a wireless link, and this network is known as Basic Service Set ( BSS). Each BSS is identified by its BSSID which corresponds to an access point MAC address. In such a network,the problem of detection of collision is solved by a collision avoidance mechanism called Distributed Control Mechanism (DCF) as specified in 802.11 specifications. Also a wi-fi station can only transmit across a channel when it is clear. It is an Acknowledged based communication, hence all transmissions are acknowledged, so after a collision a station retries after a random waiting interval.

## 2 Project Description

### 2.1 Objective

The aim of this project is to evaluate the performance of the WIFI IEEE 802.11n based network of 6 nodes (5 stations and 1 access point) similar to one shown in figure-2. UDP protocol is chosen for the communication between all the nodes and hence installed on them,Various parameters are altered in order to reach a suitable conclusion regarding the network performance, they are packetsize, number of nodes , distance between stations and access points, data rate of the Wifi, position allocations in mobility models, different types of mobility models. Subsequently, the effect of variation in the above mentioned parameters was measured and analysed in the following sections.

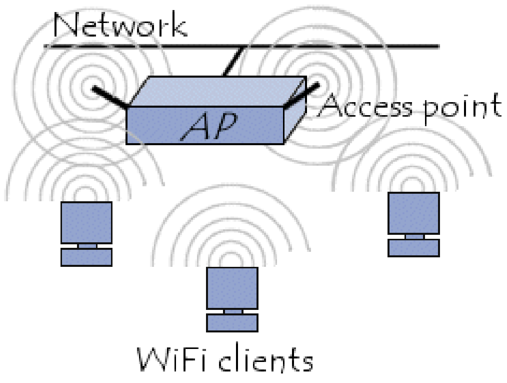


Figure 1: Wifi-Network

### 2.2 My Hypothesis

In theory, we know that throughput and delay are effected by the density of the network, data rate and the packetsize,i.e. payload. Also the way the mobility of the nodes is set is bound to effect the network performance. Cumulatively, we can say the effect of above parameters rules the network performance. For instance, if we increase the number of nodes then it can be understood that the contention between the nodes will increase, hence the average throughput per node will reduce, at the same time the delay will increase. Data rate and packetsize influence the throughput other way round. Increase in the packetsize/datarate ensures that data transferred(packets) per sec increases and hence throughput increases, but it has an counter-effect on the delay, which in this case reduces.

## 3 Project specifications and Implementation

Ns-3 simulator is used for event-based simulation of the wireless topology under test. Ns- 3 is an open source flexible Event scheduler. It makes use of C++ namespace and templates and provides for Python Bindings for most of the Public APIs. The trace output can be further analyzed using Wireshark or tcpdump utilities. It uses “waf” build system and builds a dynamic library which has both a debug and optimized version. Netanim can be used for network visualization.Figure-**??** shows the basis blocks of NS-3.

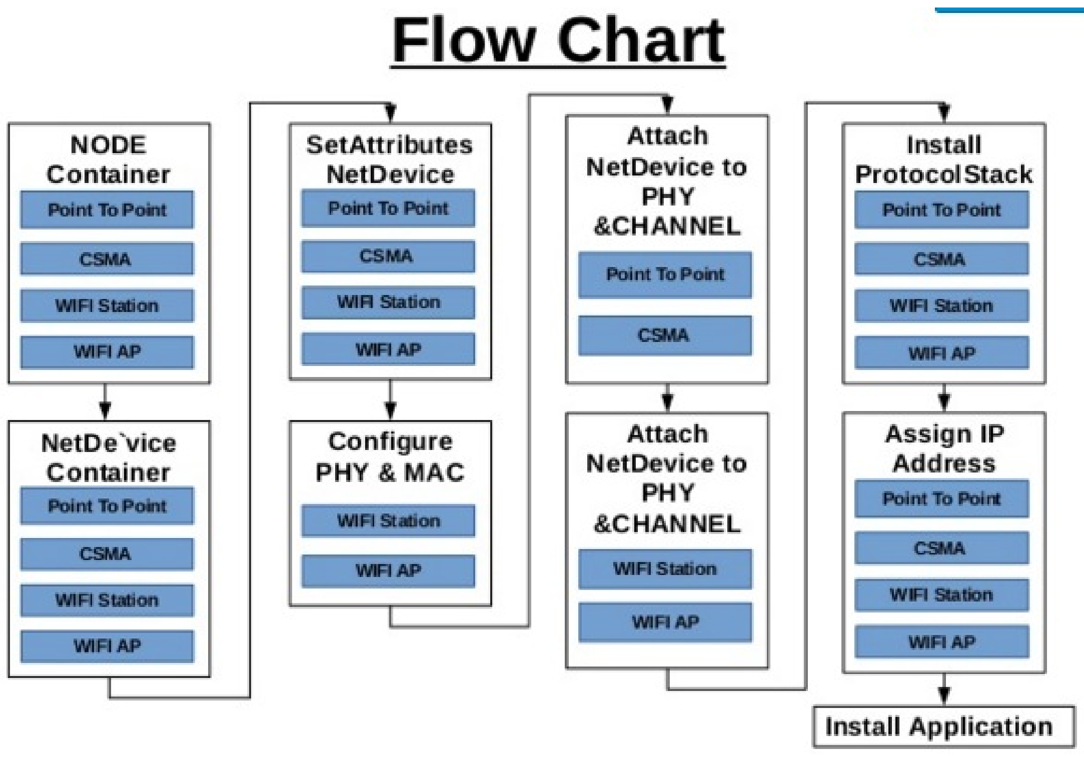


Figure 2: Basic Functioning of NS-3

The workflow of my project consists of various steps :

1. **Creating Wireless Network Topology** : CommandLine class allows processing of command line arguments. With the help of this we can supply various attributes and variables. Helpers can be used to create various wireless channels/physical and mac models. Mobility models should be associated with the wireless nodes ( stations and access points).

   - Create Wifi Nodes : Wi-fi Clients ( work-stations) and Access point.
   - Packet Attributes : Payload Size, UDP/TCP
   - Physical Layer Attributes : Data Rate, Frequency, Channelwidth etc
   - Wi-fi Standard : WIFI_PHY_STANDARD_80211n_2_4GHZ
     WIFI_PHY_STANDARD_80211n_5GHZ
   - Network Interface: Wireless and Layer 2 Protocol is 802.11n
   - Position allocators : Grid Position allocation, Uniform and random disc position allocation
   - Mobility Models : Constant Position Random Waypoint and Random Walk
   - Installing InternetStack on the nodes and Initialising their IP addresses..
   - Installing UDP server and client protocols.
   - Declaring output files for data analysis.

2. **Running Simulation and Corresponding Analysis** :

   - Tracing File : Log every packet receipt, transmit, queue, drop
   - Built-in Statistics Gathering :Throughput,Delay and Overall Network Performance
   - Custom Tracing : Traces specific packets packets/links/nodes Reduces size of trace file and post-analysis time

FlowMonitorHelper class is used along with Ipv4FlowClassifier and FlowMonitor class to monitor the packets sent and received from different nodes by attaching headers to it..Below mentioned are the formulas for throughput and delay for the network under consideration.

$$Throughput = rxBytes * 8.0/(timeLstpacketreceived - timeLstpacketreceived)/1024/nStanodes; \tag{1}$$

$$delay = delaySum/rxPackets; \tag{2}$$

## 4 Results and Discussions

In this section, the impact of various parameters on the network perfomance will be discussed. The Parameters under consideration are varied such as number of nodes, datarate, packetsize, position of nodes and mobility of nodes.
*Note: All the throughput results are in "kbps" and the delay is expressed in microseconds.When three parameters are altered for the sake of graphs other paramters are held constant. For e.g while plotting the graph between no of nodes and average throughput, payload size is also altered, but mobility model is set to "ConstantRateMobility Model" and position allocation is set to "GridPositionAllocator" and datarate is fixed to "DATAMODE VALUE 3". Similar strategy is adopted for the other calculations as well.*

1. **Effect of different packet size on Average Throughput:** According to our hypothesis as the payload or packetsize increases the throughput should increase and the similar behaviour can be observed in our system when packetsize is increased from 100 to 500 and then 1472 with number of nodes varying from 1,2,3,4,and 5 as shown in figure-3. In figure-3payload size1 is 1472, payload size 2 is 500 and payload size 3 is 100.

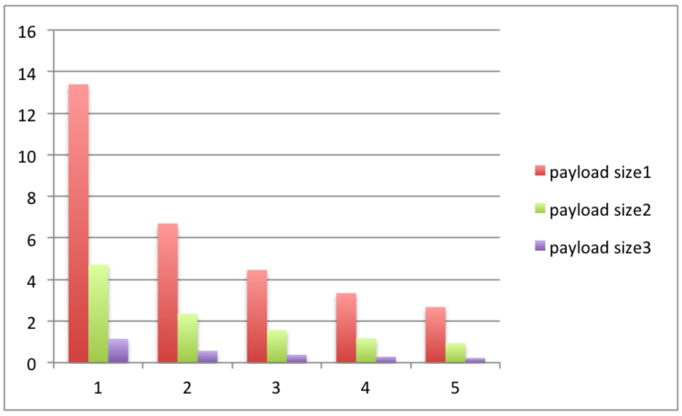


Figure 3: Average Throughput v/s Payload sizes 1472, 500, 100

2. **Effect of different mobility models on Average Throughput and Delay** : As can be seen in figures -4 and 5, the mobility models affect the delay as the number of stations are increased.But in case of throughput , there is almost no impact of change in mobility models if the number of stations is constant. Only the change in the number of stations is affecting the average throughput.Payload Size is 1472, and no of stations are increasing by 1 form 1 to 5.

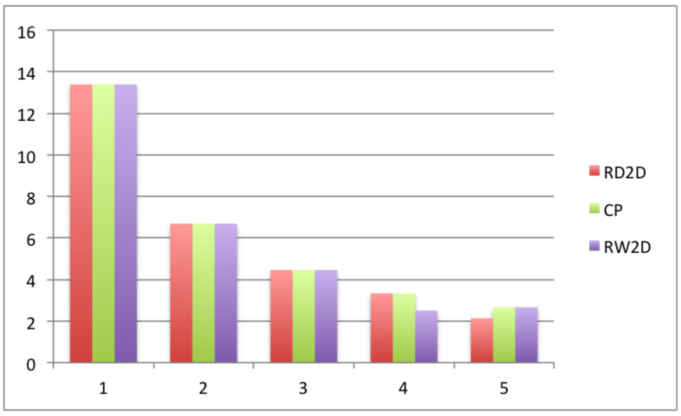


Figure 4: Average Throughput v/s Mobility Models

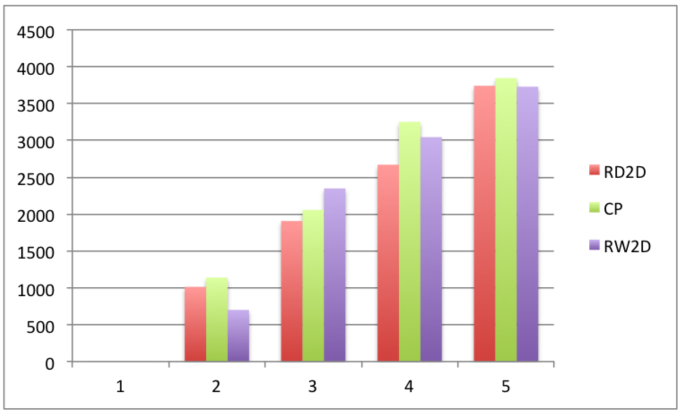


Figure 5: Average Delay v/s Mobility Models

3. **Effect of different position allocation on Average Throughput and Delay**: Different types of orientations can be selected from the NS3, out of which we have simulated our network for Grid Position Allocator- Orientation 1, Uniform Disc Allocator - Orientation 2, Random Box Position Allocator - Orientation 3, and Random Rectangle Position Allocator- Orientation 4 .The figure-6 describes the response of the network to the different types of position allocation.Payload size= 1472 and No os stations varying from 1 to 5.The results of Average Throughput were similar to that Obtained from different types of mobility models for Orientation 1 , Orientation 2 and Orientation 3, average throughput is Plotted against Number of nodes in figure-7.

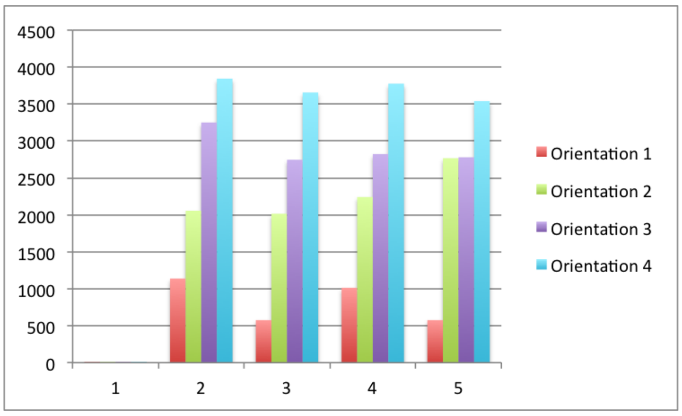


Figure 6: Average Delay v/s Orientations

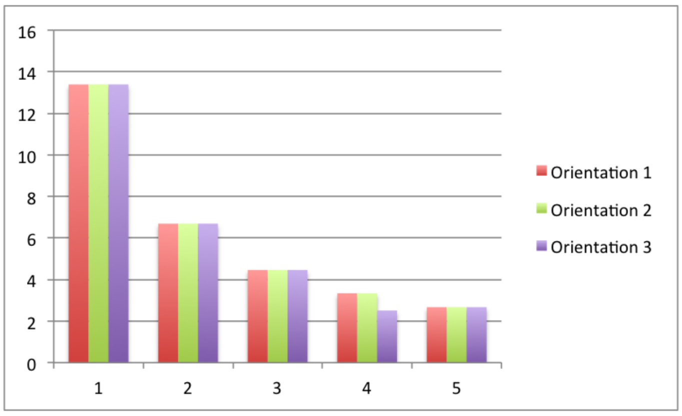


Figure 7: Average Throughput v/s Orientations

4. **Effect of different data rates on the Average Delay** : In this scenario payload size is 1472 and the number os stations are varied with each data rate.As can be seen from the figure-8, for the changing data rates the average delay is increasing. For higher datarate better throughput is observed.In figure-8,on X-axis Datarates are plotted for different number of station. Delay is plotted in Microseconds on Y axis.Delay is found to be zero for all types of datarates when number of station 1. In figure-8Series 1 stands for datarate 1, and so on so forth.

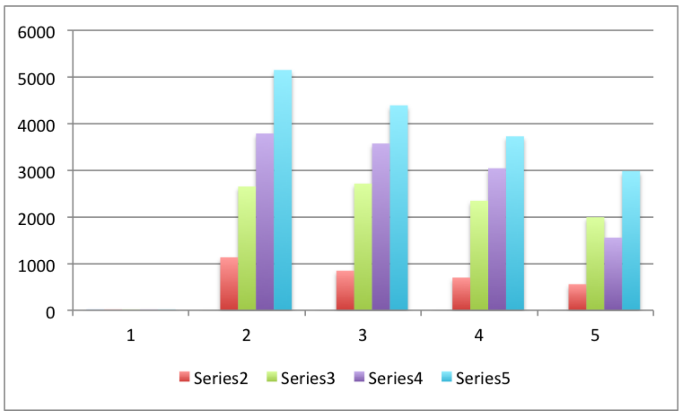


Figure 8: Average delay v/s Data rates

The data rates affect the average throughput per node in a different way with the number of stations increasing for each of the data rate,the throughput reduces. On the other hand , there is almost no impact on average throughput for a given number of stations if the datarates are altered.

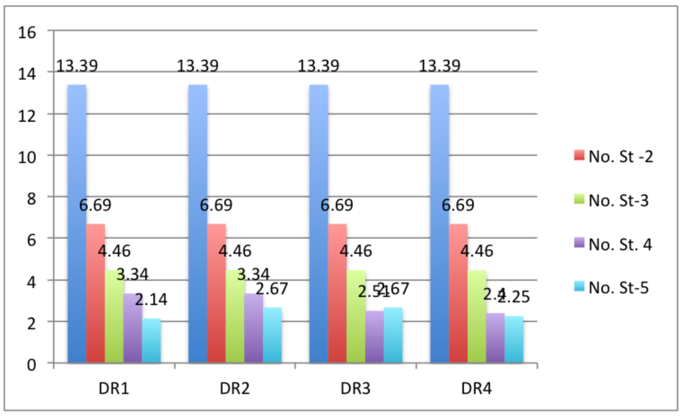
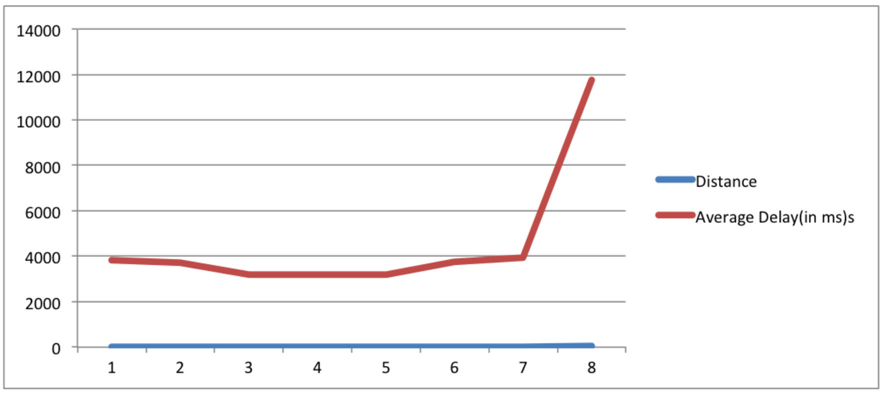
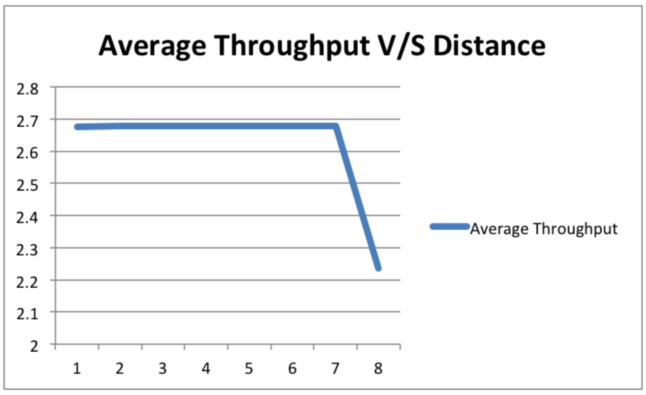


Figure 9: Average throughput v/s Number of Nodes

5. **Effect of different distances between access point and station points on Average Throughput and Delay**: According to our hypothesis it can be seen that the average delay is increasing gradually as the distance from the access point increases. But it is interesting to know that upto a certain distance from the access point, the average throughput of the network is almost constant and their no impact on throughput and after a certain distance it drastically reduces.Figure-10aand figure-10b depict the behavior of system with respect to the current scenario.



(a) Average delay v/s Distance



(b) Average Throughput v/s Distance

6. **Effect of number of nodes on Average throughput and Delay** : As it be observed from figures:11 and 12 the Average throughput decreases with the increase in the number of nodes and average Delay increases with the increase in the number of nodes.

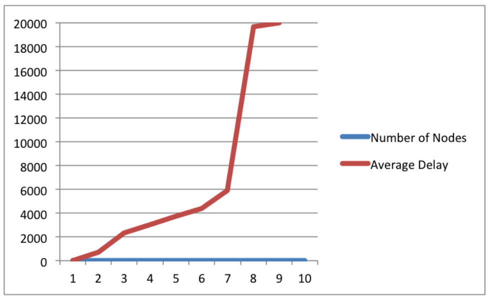


Figure 11: Average delay v/s Number of Nodes

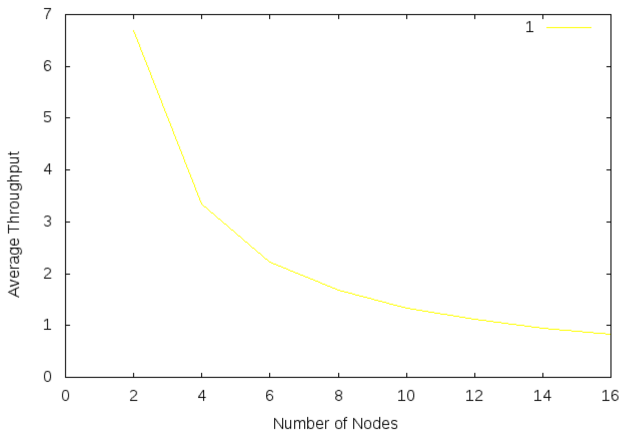


Figure 12: Average Throughput v/s Number of Nodes

## 5 Conclusion

We simulated a network of 6 nodes ( 5 stations and 1 access point) based on Wifi 802.11n with different set of mobility methods, position allocations, payload or packet size and variable distance from access points . The UDP traffic is used for simulation purposes. Following are the conclusions from the above mentioned results :

1. As the payload size increases the performance of the network improves as it was evident when the results for the payload size 1472 bytes were most optimal.
2. The increase number of nodes has an adverse effect on the network performance as the throughtput decrease in a steep fashion while the delay shoots upto infinity. The effect was visible as the average throughput per node reduced.
3. Maximum throughput was achieved for the 11 Mbps data rate although there is not much difference in comparison to other data rates when employed.However a drastic increase in the delay could be observed.
4. The results reveal that Constant Mobility Model performs better in comparison to random Direction 2D Mobility Model and Random Walk 2D Mobility Model.
5. Also as the distance from the AP increase the throughput decreases , thus is combination of bigger payload and proximity to the AP is an ideal combination for an optimal throughput. However, this also increases overall delay of system due to more time to transfer and more frequent collisions. Thus , there lies a trade-off between maximum throughput and bigger packet size.
6. Different position allocation do not affect the performance of the network to a noticeable extent if the number of station is fixed.Similar behaviour is obtained for the mobility models.

## 6 References

1. https://www.nsnam.org/documentation/
2. https://www.nsnam.org/docs/models/html/flow-monitor.html
3. Lecture Slides and Wikipedia
4. Google Groups and Stack Overflow related to NS3

## 7 Code

```
## File: ns3_code3.cc
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/stats-module.h"
#include "ns3/nstime.h"
#include "ns3/object.h"
#include "ns3/uinteger.h"
#include "ns3/traced-value.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/trace-helper.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/wifi-helper.h"
#include "string.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>

 using namespace ns3;
 using namespace std;

NS_LOG_COMPONENT_DEFINE ("PG_NS3_Assignment");

 int main (int argc, char *argv[])
 {
         double distance = 1.0;
         bool udp=true;
     int d = 0 ;
     ofstream log_file("data_values7.csv", std::ios::out | std::ios::app);

     // Creating randomness based on the basis of time

     time_t timex;

     time(&timex);

     RngSeedManager::SetSeed(timex);

     RngSeedManager::SetRun(1);

 //Generating scenario for Grid Position Allocator
         int PositionAllocator = 2;
  for(int mob=1;mob<=3;mob++)
  {
  for(int data_rate=1;data_rate<=5;data_rate++)
  {
  for (int i=1;i<=5;i++){

         cout<<"\n"<<"Distance of Access Point from stations is "<<distance+d<<endl;

  int ap_dis = distance + d ;
  int mobility_model = mob;

         // Defining various variables
```

```
    double time_simulation = 10; // time for simulation
    double freq_wifi = 5.0; //Not using 2.4 GHZ as number of collisions can be more
    int nStanodes=i;
    int nApnodes = 1;   // number of nodes serving as access points
    //bool udp = true;
int DataModevalue= data_rate ;
    bool verbose = false;
     //meters


    double packetSize = 1472;

     // mobility method (1-2)
    //Command line arguments for variables
    CommandLine cmd;
    cmd.AddValue ("freq_wifi", "Working_with_5_GHZ_WIFI_PHY_STANDARD", freq_wifi);
    cmd.AddValue ("time_simulation", "Simulation_time_in_seconds", time_simulation);
    cmd.AddValue ("udp", "UDP_is_used", udp);
    cmd.AddValue("distance","Distance_between_access_point_and_the_stations",distance);
    cmd.AddValue ("nStanodes", "Number_of_stations", nStanodes);
    cmd.AddValue("nApnodes","Number_of_access_points", nApnodes);
    cmd.AddValue ("packetSize", "packetSize_for_UDP", packetSize);
    cmd.AddValue ("DataModevalue", "DataMode_value_for_MAC", DataModevalue);
    cmd.AddValue ("PositionAllocator", "Different_types_of_orientations", PositionAllocato
cmd.AddValue ("mobility_model","Different_types_of_mobility_models",mobility_model);

    cmd.Parse (argc,argv);

    //Enabeling the log component
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);


    //Creating wifi station nodes
    NodeContainer wifiStaNode;
    wifiStaNode.Create (nStanodes);

    //Creating wifi acess point nodes
    NodeContainer wifiApNode;
    wifiApNode.Create (nApnodes);

    //Changing physical layer attributes
    YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
    YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
    phy.SetChannel (channel.Create ());

    //Enabling the guard
    phy.Set ("ShortGuardEnabled", BooleanValue (true));

    //Setting up Wi-Fi standards
    WifiHelper wifi = WifiHelper::Default ();
    if (freq_wifi == 5.0)
        {
       wifi.SetStandard (WIFI_PHY_STANDARD_80211n_5GHZ);
    }
else if (freq_wifi == 2.4)
    {
        wifi.SetStandard (WIFI_PHY_STANDARD_80211n_2_4GHZ);
        Config::SetDefault ("ns3::LogDistancePropagationLossModel::ReferenceLoss",
        DoubleValue (40.046));
    }
else
   {
        std::cout<<"Wrong_freq_wifi_value!"<<std::endl;
```

```
            return 0;
        }
        //wifi.SetStandard (WIFI_PHY_STANDARD_80211n_5GHZ);

        if (verbose)
        {
                wifi.EnableLogComponents ();
        }

        //Initialising MAC layer and Remote Station
        HtWifiMacHelper mac = HtWifiMacHelper::Default ();
        StringValue DataMode = HtWifiMacHelper::DataRateForMcs (DataModevalue);
        wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode",
    DataMode,"ControlMode", DataMode);

    //Creating wifi devices and setting up other related parameters
        Ssid ssid = Ssid ("WN_PG_NS3_80211n");
        mac.SetType ("ns3::StaWifiMac","Ssid", SsidValue (ssid),"ActiveProbing",
    BooleanValue (false));
        NetDeviceContainer staDevice;
        staDevice = wifi.Install (phy, mac, wifiStaNode);

        mac.SetType ("ns3::ApWifiMac","Ssid", SsidValue (ssid));
        NetDeviceContainer apDevice;
        apDevice = wifi.Install (phy, mac, wifiApNode);

        //Setting up channel width
        Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/ChannelWidth",
    UintegerValue (20));

        //Defining different Modility Methods for Nodes
        MobilityHelper mobility;
        switch(PositionAllocator)
        {
        case 1:

            cout<<"\n"<<"-----Random_Disc_Position_Allocator----------"<<endl;
                mobility.SetPositionAllocator ("ns3::RandomDiscPositionAllocator",
   "X", DoubleValue (5.0),
    "Y", DoubleValue (5.0),
"Theta",  StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=6.283]"),
"Rho",StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=200.0]"));
                        break;
                case 2:
           // log_file << "GridPositionAllocator";
            cout<<"\n"<<"-----Grid_Position_Allocator------"<<endl;
                mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                "MinX", DoubleValue (0.0),
                                "MinY", DoubleValue (0.0),
                                "DeltaX", DoubleValue (10.0),
                                "DeltaY", DoubleValue (10.0),
                                "GridWidth", UintegerValue (4),
                                "LayoutType", StringValue ("RowFirst"));
                        break;
        case 3:
            //log_file << "UniformDiscPositionAllocator";
            cout<<"\n"<<"-----Uniform_Disc_Position_Allocator------"<<endl;
            mobility.SetPositionAllocator ("ns3::UniformDiscPositionAllocator",
                "X", DoubleValue (5.0),
                "Y", DoubleValue (5.0),
                "rho", DoubleValue (5.0));
    break;
                default:
                std::cout<<"(0,0)_positions_assigned_to_all_nodes";
        }

//Different types of Mobility models which can be implemented
 //1 : RandomDirection2d
//2 : RandomWalk2d
//3 : ConstantPosition
switch(mobility_model)
{
    case 1:
       std::cout << "RandomDirection2d" << "\t";
       mobility.SetMobilityModel ("ns3::RandomDirection2dMobilityModel", "Bounds",
       RectangleValue (Rectangle (-10, 10, -10, 10)), "Speed",
       StringValue ("ns3::ConstantRandomVariable[Constant=3]"), "Pause",
       StringValue ("ns3::ConstantRandomVariable[Constant=0.4]"));
       break;
    case 2:
        std::cout << "RandomWalk2d" << "\t";
        mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
        "Bounds", RectangleValue (Rectangle (-1000, 1000, -1000, 1000)),
        "Distance", ns3::DoubleValue (300.0));
         break;
    case 3:
         std::cout << "ConstantPosition" << "\t";
         mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
         break;
    default:
          std::cout << "RandomWalk2d" << "\t";
    break;
}



        //mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
        mobility.Install (wifiStaNode);
 //std::cout << "Mobility of nodes configured" << std::endl;
NS_LOG_INFO("Mobility␣of␣station␣nodes␣configured");

    //Varying distance between the access points and the stations
 MobilityHelper mobility_ap;
 Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
 positionAlloc->Add (Vector (ap_dis, 0.0, 0.0));
 mobility_ap.SetPositionAllocator (positionAlloc);
 mobility_ap.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
 mobility_ap.Install (wifiApNode);
 NS_LOG_INFO("Mobility␣of␣access␣point␣configured");




        //Installing Internet Stack on all nodes
        InternetStackHelper stack;
        stack.Install (wifiApNode);
        stack.Install (wifiStaNode);

        //Providing IP address to All Nodes
        Ipv4AddressHelper address;
        address.SetBase ("192.168.1.0", "255.255.255.0");

        Ipv4InterfaceContainer staNodeInterface;
        Ipv4InterfaceContainer apNodeInterface;

        apNodeInterface = address.Assign (apDevice);
        staNodeInterface = address.Assign (staDevice);
        //Installation of UDP protocol
        ApplicationContainer serverApp;
        UdpServerHelper myServer (9);
        serverApp = myServer.Install (wifiApNode);

        //Server start and stop time definition
        serverApp.Start (Seconds (1.0));
        serverApp.Stop (Seconds (time_simulation + 1));


        //Setting up UDP protocol attributes
        UdpClientHelper myClient (apNodeInterface.GetAddress (0), 9);
        myClient.SetAttribute ("Interval", TimeValue (Seconds (1))); //packets/s
        myClient.SetAttribute ("PacketSize", UintegerValue (packetSize));//payload
        myClient.SetAttribute ("MaxPackets", UintegerValue (nStanodes));

        ApplicationContainer clientApp = myClient.Install (wifiStaNode);
        clientApp.Start (Seconds (2.0));
        clientApp.Stop (Seconds (time_simulation + 1));

        //Displaying of IP Address of nodes
        //cout<<"Address for Access Point node: "<<apNodeInterface.GetAddress(0)<<std::endl;
        //cout<<"Address for Nodes\n";
        //for(int i=0;i<nStanodes;++i)
                //{
                //std::cout<<"Node "<<i<<" : "<<staNodeInterface.GetAddress(i)<<std::endl;
        //}

        //Populating routing tables of nodes
        Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
    AnimationInterface anim("ns3_code.xml");

        //Simulation results for station points and access points are captured
        phy.EnablePcap("Simulation_nodes",wifiStaNode);
        phy.EnablePcap("Simulation_apnodes",wifiApNode);


        //Calculation of throughput_udp and Delay
 double throughput_udp[100];
 double delay[100];

 FlowMonitorHelper flowmonitor;
 Ptr<FlowMonitor> monitor = flowmonitor.InstallAll();

    //Specifing run time
 Simulator::Stop (Seconds (time_simulation));

    //Starting the simulation
 Simulator::Run ();
 double false_nodes =0;
 double avg_thr=0,avg_dely=0;
 monitor->CheckForLostPackets (); //Monitoring packet transfer
 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
 (flowmonitor.GetClassifier ());
 map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

 for (map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin ();
 i != stats.end (); ++i)
    {
        throughput_udp[i->first] = ((i->second.rxBytes) * 8.0) /
(i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())
/1024/nStanodes;
        if(throughput_udp[i->first]==0) // unreachable nodes
```

```
          { delay[i->first]=9999999999;
           false_nodes = false_nodes + 1;}

        else
        {
            Time Tdelay (i->second.delaySum / i->second.rxPackets);
            delay[i->first]=Tdelay.ToDouble(Time::US);
        }


       //cout<<"------------------------------------------"<<"\n";
    }

 Simulator::Destroy ();

 // Calculating avarage throughput_udp and delay for simulation scenario

 for (int i=1;i<=nStanodes;++i)
 {
        avg_thr = avg_thr + throughput_udp[i];
        avg_dely = avg_dely + delay[i];
  }
 avg_dely=avg_dely/(nStanodes-false_nodes);
 avg_thr=avg_thr/(nStanodes);
    //Writing Various values to the log file

    log_file << "Number_of_nodes,Average_throughput_udp,
␣␣␣␣Average_delay,Mobility_model,data_rate\n";
    log_file << nStanodes << "," << avg_thr << "," <<
    avg_dely << "," << mobility_model<< ","
   <<DataModevalue<<"\n";
}
}
}
    log_file.close();
    return 0;
}
```