DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL OF
WIRELESS COMMUNICATION LAB
ETEC 463

Maharaja Agrasen Institute of Technology, PSP area,
Sector – 22, Rohini, New Delhi – 110086

(Affiliated to Guru Gobind Singh Indraprastha University,
Dwarka New Delhi)

INDEX OF THE CONTENTS

1. Introduction to Wireless Communication Lab

Context and Aims

This is an advanced course in telecommunications, providing detailed knowledge of the fundamental concepts in wireless communications and in-depth discussions on several selected areas, namely, GSM, CDMA,3G, 4G, WLAN, Spread Spectrum techniques, wideband transmissions. This course is a professional course offered in the Wireless Communication ETEC 405 Outline – Semester 8, 2016-17.

Aims:

- Make the student familiar with the basic principles of information transmission in wireless networks.
- Make the student familiar with wireless transmission techniques and their applications.
- Enable the student to do analysis and design transmission and receiving algorithms.
- Learn Basic Principles of WLAN, GSM, CDMA and Spread Spectrum Techniques.

2. LAB REQUIREMENTS

H/W Detail	Intel i3/C2D Processor/2 GB RAM/500GB HDD/MB/Lan Card/ Key Board/ Mouse/CD Drive/15” Color Monitor/ UPS	24 Nos.
	LaserJet Printer	1 No.
S/W Detail	Fedora/NS3	

3. LIST OF EXPERIMENTS

1. Program in NS 3 to connect WIFI TO BUS(CSMA)
2. Program in NS 3 to create WIFI SIMPLE INFRASTUCTURE MODE
3. Program in NS 3 to create WIFI SIMPLE ADHOC MODE
4. Program in NS 3 to connect WIFI TO WIRED BRIDGING
5. Program in NS 3 to create WIFI TO LTE(4G) CONNECTION
6. Program in NS3 for CREATING A SIMPLE WIFI ADHOC GRID
7. Introduction to GSM Architecture

4. FORMAT OF THE LAB RECORD TO BE PREPARED BY THE STUDENTS

The front page of the lab record prepared by the students should have a cover page as displayed below.

NAME OF THE LAB

Font should be (Size 20", italics bold, Times New Roman)

Faculty name

Student name

Roll No.:

Semester:

Group:

Font should be (12", Times Roman)

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110086

The second page in the record should be the index as displayed below.

LAB NAME

PRACTICAL RECORD

PAPER CODE **:**

Name of the student :

University Roll No. :

Branch :

Section/ Group :

PRACTICAL DETAILS

Experiments according to the lab syllabus prescribed by GGSIPU

Exp. no	Experiment Name	Date of performance	Date of checking	Remarks	Marks

5. MARKING SCHEME FOR THE PRACTICAL EXAMS

There will be two practical exams in each semester.

- Internal Practical Exam
- External Practical Exam

INTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 40

Division of 40 marks is as follows

1. Regularity: 30

    - Performing program in each turn of the lab
    - Attendance of the lab
    - File

2. Viva Voice: 10

NOTE: For the regularity, marks are awarded to the student out of 5 for each experiment performed in the lab and at the end the average marks are giving out of 30.

EXTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch and by an external examiner. In this exam student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 60

Division of 60 marks is as follows

1. Sheet filled by the student: 20
2. Viva Voice: 15
3. Experiment performance: 15
4. File submitted: 10

NOTE:

- Internal marks + External marks = Total marks given to the students
  (40 marks) (60 marks) (100 marks)

- Experiments given to perform can be from any section of the lab.

Introduction to NS3 :

Simple client-server communication

Expected learning outcome: NS-3 simulation basics. Basic client server paradigm. Reading pcap traces.

- **Experiment**:
    1. Create a simple topology of two nodes (Node1, Node2) separated by a point-to-point link.
    2. Setup a UdpClient on one Node1 and a UdpServer on Node2. Let it be of a fixed data rate Rate1.
    3. Start the client application, and measure end to end throughput whilst varying the latency of the link.
    4. Now add another client application to Node1 and a server instance to Node2. What do you need to configure to ensure that there is no conflict?
    5. Repeat step 3 with the extra client and server application instances. Show screenshots of pcap traces which indicate that delivery is made to the appropriate server instance.

TCP variants

- **Expected learning outcome**: TCP internals and the difference between each of the variants. NS-3 tracing mechanism.

- **Experiment**:
    1. Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links.
    2. Install a TCP socket instance on Node1 that will connect to Node3.
    3. Install a UDP socket instance on Node2 that will connect to Node4.
    4. Start the TCP application at time 1s.
    5. Start the UDP application at time 20s at rate Rate1 such that it clogs half the dumbbell bridge's link capacity.
    6. Increase the UDP application's rate at time 30s to rate Rate2 such that it clogs the whole of the dumbbell bridge's capacity.

7. Use the ns-3 tracing mechanism to record changes in congestion window size of the TCP instance over time. Use gnuplot/matplotlib to visualise plots of cwnd vs time.
8. Mark points of fast recovery and slow start in the graphs.
9. Perform the above experiment for TCP variants Tahoe, Reno and New Reno, all of which are available with ns-3.

TCP and router queues

- **Expected learning outcome**: Queues, packet drops and their effect on congestion window size.

- **Experiment**:
    1. As in previous exercise, Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links.
    2. Add drop tail queues of size QueueSize5 and QueueSize6 to Node5 and Node6, respectively.
    3. Install a TCP socket instance on Node1 that will connect to Node3.
    4. Install a TCP socket instance on Node2 that will connect to Node3.
    5. Install a TCP socket instance on Node2 that will connect to Node4.
    6. Start Node1--Node3 flow at time 1s, then measure it's throughput. How long does it take to fill link's entire capacity?
    7. Start Node2--Node3 and Node2--Node4 flows at time 15s, measure their throughput.
    8. Measure packet loss and cwnd size, and plot graphs throughput/time, cwnd/time and packet loss/time for each of the flows.
    9. Plot graph throughput/cwnd and packet loss/cwnd for the first flow. Is there an optimal value for cwnd?
    10. Vary QueueSize5 and QueueSize6. Which one has immediate effect on cwnd size of the first flow? Explain why.

Routing (Optimised Link State Routing)

- **Expected learning outcome**: What are MANETs and how they work. OLSR basics. Routing issues associated with MANETs.

- **Experiment**:
    1. Create a wireless mobile ad-hoc network with three nodes Node1, Node2 and Node3. Install the OLSR routing protocol on these nodes.
    2. Place them such that Node1 and Node3 are just out of reach of each other.
    3. Create a UDP client on Node1 and the corresponding server on Node3.
    4. Schedule Node1 to begin sending packets to Node3 at time 1s.
    5. Verify whether Node1 is able to send packets to Node3.
    6. Make Node2 move between Node1 and Node3 such that Node2 is visible to both A and C. This should happen at time 20s. Ensure that Node2 stays in that position for another 15s.
    7. Verify whether Node1 is able to send packets to Node3.
    8. At time 35s, move Node2 out of the region between Node1 and Node3 such that it is out of each other's transmission ranges again.
    9. Verify whether Node1 is able to send packets to Node3.
    10. To verify whether data transmissions occur in the above scenarios, use either the tracing mechanism or a RecvCallback() for Node3's socket.
    11. Plot the number of bytes received versus time at Node3.
    12. Show the pcap traces at Node 2's Wifi interface, and indicate the correlation between Node2's packet reception timeline and Node2's mobility.

Wifi RTS/CTS

- **Expected learning outcome**: How 802.11 works with and without RTS/CTS. An insight into why its hard to setup efficient wireless networks.

- **Experiment**:
    1. Setup a 5x5 wireless adhoc network with a grid. You may use examples/wireless/wifi-simple-adhoc-grid.cc as a base.
    2. Install the OLSR routing protocol.

3. Setup three UDP traffic flows, one along each diagonal and one along the middle (at high rates of transmission).
4. Setup the ns-3 flow monitor for each of these flows.
5. Now schedule each of the flows at times 1s, 1.5s, and 2s.
6. Now using the flow monitor, observe the throughput of each of the UDP flows. Furthermore, use the tracing mechanism to monitor the number of packet collisions/drops at intermediary nodes. Around which nodes are most of the collisions/drops happening?
7. Now repeat the experiment with RTS/CTS enabled on the wifi devices.
8. Show the difference in throughput and packet drops if any.

Wifi Channels

- **Expected learning outcome**: How Radio channel models affect transmission. An insight into why its important to correctly model the channel.

- **Experiment**:
    1. Setup a 2-nodes wireless adhoc network. Place the nodes at a fixed distance in a 3d scenario.
    2. Install all the relevant network stacks, up to and including UDP.
    3. Setup a CBR transmission between the nodes, one acting as a server and one as a client. Take the iperf[1] behaviour as an example.
    4. Setup counters and outputs for packets sent and received.
    5. Schedule the simulation to run for enough time to obtain statistically relevant results (suggestion: analyze some test results and reduce the simulation time accordingly).
    6. Repeat the simulation varying the distance between the nodes from a minimum of 1meter to the point where the nodes can't transmit/receive anymore.
    7. Repeat the above varying the channel models and the transmission/receive parameters like node's position above the ground, transmission power, etc.
    8. Show the differences between the various channel models, and comment them. Identify the channel model that is more appropriate for each case (indoor, outdoor, LoS, NLoS, etc.).

1. WIFI TO BUS(CSMA) CONNECTION

```
// Default Network Topology
//
// Number of wifi or csma nodes can be increased up to 250
//                          |
//               Rank 0   |   Rank 1
// -------------------------|----------------------------
//   Wifi 10.1.3.0
//                AP
// *    *    *    *
// |    |    |    |    10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                 point-to-point  |    |    |    |
//                                 ================
//                                   LAN 10.1.2.0


#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

int
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  bool tracing = false;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);
```

```
// Check for valid number of csma or wifi nodes
// 250 should be enough, otherwise IP addresses
// soon become an issue
if (nWifi > 250 || nCsma > 250)
  {
    std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::endl;
    return 1;
  }

if (verbose)
  {
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }

NodeContainer p2pNodes;
p2pNodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi;
```

```
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

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

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);

MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);

InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
```

```
  Ipv4InterfaceContainer csmaInterfaces;
  csmaInterfaces = address.Assign (csmaDevices);

  address.SetBase ("10.1.3.0", "255.255.255.0");
  address.Assign (staDevices);
  address.Assign (apDevices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps =
    echoClient.Install (wifiStaNodes.Get (nWifi - 1));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Stop (Seconds (10.0));

  if (tracing == true)
    {
      pointToPoint.EnablePcapAll ("third");
      phy.EnablePcap ("third", apDevices.Get (0));
      csma.EnablePcap ("third", csmaDevices.Get (0), true);
    }

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

2. WIFI SIMPLE INFRASTUCTURE MODE

This script configures two nodes on an 802.11b physical layer, with 802.11b NICs in infrastructure mode, and by default, the station sends one packet of 1000 (application) bytes to the access point. The physical layer is configured to receive at a fixed RSS (regardless of the distance and transmit power); therefore, changing position of the nodes has no effect.

There are a number of command-line options available to control the default behavior. The list of available command-line options can be listed with the following command:
./waf --run "wifi-simple-infra --help" For instance, for this configuration, the physical layer will stop successfully receiving packets when rss drops below -97 dBm. To see this effect, try running:

./waf --run "wifi-simple-infra --rss=-97 --numPackets=20"
./waf --run "wifi-simple-infra --rss=-98 --numPackets=20"/ ./waf --run "wifi-simple-infra --rss=-99 --numPackets=20"

Note that all ns-3 attributes (not just the ones exposed in the below script) can be changed at command line; see the documentation.

This script can also be helpful to put the Wifi layer into verbose logging mode; this command will turn on all wifi logging:
./waf --run "wifi-simple-infra --verbose=1"
When you are done, you will notice two pcap trace files in your directory. If you have tcpdump installed, you can try this:
tcpdump -r wifi-simple-infra-0-0.pcap -nn -tt

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("WifiSimpleInfra");

void ReceivePacket (Ptr<Socket> socket)
{
```

```
  while (socket->Recv ())
    {
      NS_LOG_UNCOND ("Received one packet!");
    }
}

static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                             uint32_t pktCount, Time pktInterval )
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &GenerateTraffic,
                           socket, pktSize,pktCount-1, pktInterval);
    }
  else
    {
      socket->Close ();
    }
}


int main (int argc, char *argv[])
{
  std::string phyMode ("DsssRate1Mbps");
  double rss = -80;  // -dBm
  uint32_t packetSize = 1000; // bytes
  uint32_t numPackets = 1;
  double interval = 1.0; // seconds
  bool verbose = false;

  CommandLine cmd;

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
  cmd.AddValue ("rss", "received signal strength", rss);
  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
  cmd.AddValue ("numPackets", "number of packets generated", numPackets);
  cmd.AddValue ("interval", "interval (seconds) between packets", interval);
  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);

  cmd.Parse (argc, argv);
  // Convert to time object
  Time interPacketInterval = Seconds (interval);

  // disable fragmentation for frames below 2200 bytes
```

```
  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
StringValue ("2200"));
  // turn off RTS/CTS for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue
("2200"));
  // Fix non-unicast data rate to be the same as that of unicast
  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                      StringValue (phyMode));

  NodeContainer c;
  c.Create (2);

  // The below set of helpers will help us to put together the wifi NICs we want
  WifiHelper wifi;
  if (verbose)
    {
      wifi.EnableLogComponents ();  // Turn on all Wifi logging
    }
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
  // This is one parameter that matters when using FixedRssLossModel
  // set it to zero; otherwise, gain will be added
  wifiPhy.Set ("RxGain", DoubleValue (0) );
  // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
  wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  // The below FixedRssLossModel will cause the rss to be fixed regardless
  // of the distance between the two stations, and the transmit power
  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue (rss));
  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a mac and disable rate control
  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));

  // Setup the rest of the mac
  Ssid ssid = Ssid ("wifi-default");
  // setup sta.
  wifiMac.SetType ("ns3::StaWifiMac",
                   "Ssid", SsidValue (ssid),
                   "ActiveProbing", BooleanValue (false));
```

```
  NetDeviceContainer staDevice = wifi.Install (wifiPhy, wifiMac, c.Get (0));
  NetDeviceContainer devices = staDevice;
  // setup ap.
  wifiMac.SetType ("ns3::ApWifiMac",
                   "Ssid", SsidValue (ssid));
  NetDeviceContainer apDevice = wifi.Install (wifiPhy, wifiMac, c.Get (1));
  devices.Add (apDevice);

  // Note that with FixedRssLossModel, the positions below are not
  // used for received signal strength.
  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
  positionAlloc->Add (Vector (0.0, 0.0, 0.0));
  positionAlloc->Add (Vector (5.0, 0.0, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (c);

  InternetStackHelper internet;
  internet.Install (c);

  Ipv4AddressHelper ipv4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (devices);

  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (0), tid);
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
  recvSink->Bind (local);
  recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

  Ptr<Socket> source = Socket::CreateSocket (c.Get (1), tid);
  InetSocketAddress remote = InetSocketAddress (Ipv4Address ("255.255.255.255"), 80);
  source->SetAllowBroadcast (true);
  source->Connect (remote);

  // Tracing
  wifiPhy.EnablePcap ("wifi-simple-infra", devices);

  // Output what we are doing
  NS_LOG_UNCOND ("Testing " << numPackets  << " packets sent with receiver rss " << rss
);

  Simulator::ScheduleWithContext (source->GetNode ()->GetId (),
                                  Seconds (1.0), &GenerateTraffic,
```

```
                                source, packetSize, numPackets, interPacketInterval);

  Simulator::Stop (Seconds (30.0));
  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

3. WIFI SIMPLE ADHOC MODE

This script configures two nodes on an 802.11b physical layer, with 802.11b NICs in adhoc mode, and by default, sends one packet of 1000 (application) bytes to the other node. The physical layer is configured to receive at a fixed RSS (regardless of the distance and transmit power); therefore, changing position of the nodes has no effect.

There are a number of command-line options available to control the default behavior. The list of available command-line options can be listed with the following command: ./waf --run "wifi-simple-adhoc --help" For instance, for this configuration, the physical layer will stop successfully receiving packets when rss drops below -97 dBm To see this effect, try running:

./waf --run "wifi-simple-adhoc --rss=-97 --numPackets=20"
./waf --run "wifi-simple-adhoc --rss=-98 --numPackets=20"
./waf --run "wifi-simple-adhoc --rss=-99 --numPackets=20"

Note that all ns-3 attributes (not just the ones exposed in the below script) can be changed at ommand line; see the documentation.

This script can also be helpful to put the Wifi layer into verbose logging mode; this command will turn on all wifi logging:

./waf --run "wifi-simple-adhoc --verbose=1"

When you are done, you will notice two pcap trace files in your directory. If you have tcpdump installed, you can try this:
tcpdump -r wifi-simple-adhoc-0-0.pcap -nn -tt

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("WifiSimpleAdhoc");
```

```
void ReceivePacket (Ptr<Socket> socket)
{
  while (socket->Recv ())
    {
      NS_LOG_UNCOND ("Received one packet!");
    }
}

static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                             uint32_t pktCount, Time pktInterval )
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &GenerateTraffic,
                           socket, pktSize,pktCount-1, pktInterval);
    }
  else
    {
      socket->Close ();
    }
}


int main (int argc, char *argv[])
{
  std::string phyMode ("DsssRate1Mbps");
  double rss = -80;  // -dBm
  uint32_t packetSize = 1000; // bytes
  uint32_t numPackets = 1;
  double interval = 1.0; // seconds
  bool verbose = false;

  CommandLine cmd;

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
  cmd.AddValue ("rss", "received signal strength", rss);
  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
  cmd.AddValue ("numPackets", "number of packets generated", numPackets);
  cmd.AddValue ("interval", "interval (seconds) between packets", interval);
  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);

  cmd.Parse (argc, argv);
  // Convert to time object
  Time interPacketInterval = Seconds (interval);
```

```
  // disable fragmentation for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
StringValue ("2200"));
  // turn off RTS/CTS for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue
("2200"));
  // Fix non-unicast data rate to be the same as that of unicast
  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                      StringValue (phyMode));

  NodeContainer c;
  c.Create (2);

  // The below set of helpers will help us to put together the wifi NICs we want
  WifiHelper wifi;
  if (verbose)
    {
      wifi.EnableLogComponents ();  // Turn on all Wifi logging
    }
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
  // This is one parameter that matters when using FixedRssLossModel
  // set it to zero; otherwise, gain will be added
  wifiPhy.Set ("RxGain", DoubleValue (0) );
  // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
  wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  // The below FixedRssLossModel will cause the rss to be fixed regardless
  // of the distance between the two stations, and the transmit power
  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue (rss));
  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a mac and disable rate control
  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));
  // Set it to adhoc mode
  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

  // Note that with FixedRssLossModel, the positions below are not
  // used for received signal strength.
```

```
  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
  positionAlloc->Add (Vector (0.0, 0.0, 0.0));
  positionAlloc->Add (Vector (5.0, 0.0, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (c);

  InternetStackHelper internet;
  internet.Install (c);

  Ipv4AddressHelper ipv4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (devices);

  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (0), tid);
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
  recvSink->Bind (local);
  recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

  Ptr<Socket> source = Socket::CreateSocket (c.Get (1), tid);
  InetSocketAddress remote = InetSocketAddress (Ipv4Address ("255.255.255.255"), 80);
  source->SetAllowBroadcast (true);
  source->Connect (remote);

  // Tracing
  wifiPhy.EnablePcap ("wifi-simple-adhoc", devices);

  // Output what we are doing
  NS_LOG_UNCOND ("Testing " << numPackets  << " packets sent with receiver rss " << rss
);

  Simulator::ScheduleWithContext (source->GetNode ()->GetId (),
                                  Seconds (1.0), &GenerateTraffic,
                                  source, packetSize, numPackets, interPacketInterval);

  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

4. WIFI TO WIRED BRIDGING

Default network topology includes some number of AP nodes specified by the variable nWifis (defaults to two). Off of each AP node, there are some number of STA nodes specified by the variable nStas (defaults to two). Each AP talks to its associated STA nodes. There are bridge net devices on each AP node that bridge the whole thing into one network.

```
//
//      +-----+      +-----+            +-----+      +-----+
//      | STA |      | STA |            | STA |      | STA |
//      +-----+      +-----+            +-----+      +-----+
//    192.168.0.2  192.168.0.3        192.168.0.5  192.168.0.6
//      --------     --------           --------     --------
//      WIFI STA     WIFI STA           WIFI STA     WIFI STA
//      --------     --------           --------     --------
//        ((*))        ((*))       |      ((*))         ((*))
//                                 |
//              ((*))              |             ((*))
//             -------                          -------
//             WIFI AP   CSMA ========= CSMA    WIFI AP
//             -------   ----           ----    -------
//             ##############           ##############
//                 BRIDGE                   BRIDGE
//             ##############           ##############
//               192.168.0.1              192.168.0.4
//               +---------+              +---------+
//               | AP Node |              | AP Node |
//               +---------+              +---------+
//
```

```
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/bridge-helper.h"
#include <vector>
#include <stdint.h>
#include <sstream>
#include <fstream>

using namespace ns3;

int main (int argc, char *argv[])
{
  uint32_t nWifis = 2;
```

```
uint32_t nStas = 2;
bool sendIp = true;
bool writeMobility = false;

CommandLine cmd;
cmd.AddValue ("nWifis", "Number of wifi networks", nWifis);
cmd.AddValue ("nStas", "Number of stations per wifi network", nStas);
cmd.AddValue ("SendIp", "Send Ipv4 or raw packets", sendIp);
cmd.AddValue ("writeMobility", "Write mobility trace", writeMobility);
cmd.Parse (argc, argv);

NodeContainer backboneNodes;
NetDeviceContainer backboneDevices;
Ipv4InterfaceContainer backboneInterfaces;
std::vector<NodeContainer> staNodes;
std::vector<NetDeviceContainer> staDevices;
std::vector<NetDeviceContainer> apDevices;
std::vector<Ipv4InterfaceContainer> staInterfaces;
std::vector<Ipv4InterfaceContainer> apInterfaces;

InternetStackHelper stack;
CsmaHelper csma;
Ipv4AddressHelper ip;
ip.SetBase ("192.168.0.0", "255.255.255.0");

backboneNodes.Create (nWifis);
stack.Install (backboneNodes);

backboneDevices = csma.Install (backboneNodes);

double wifiX = 0.0;

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

for (uint32_t i = 0; i < nWifis; ++i)
  {
    // calculate ssid for wifi subnetwork
    std::ostringstream oss;
    oss << "wifi-default-" << i;
    Ssid ssid = Ssid (oss.str ());

    NodeContainer sta;
    NetDeviceContainer staDev;
    NetDeviceContainer apDev;
    Ipv4InterfaceContainer staInterface;
```

```
    Ipv4InterfaceContainer apInterface;
    MobilityHelper mobility;
    BridgeHelper bridge;
    WifiHelper wifi;
    WifiMacHelper wifiMac;
    YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
    wifiPhy.SetChannel (wifiChannel.Create ());

    sta.Create (nStas);
    mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                   "MinX", DoubleValue (wifiX),
                                   "MinY", DoubleValue (0.0),
                                   "DeltaX", DoubleValue (5.0),
                                   "DeltaY", DoubleValue (5.0),
                                   "GridWidth", UintegerValue (1),
                                   "LayoutType", StringValue ("RowFirst"));


    // setup the AP.
    mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
    mobility.Install (backboneNodes.Get (i));
    wifiMac.SetType ("ns3::ApWifiMac",
                     "Ssid", SsidValue (ssid));
    apDev = wifi.Install (wifiPhy, wifiMac, backboneNodes.Get (i));

    NetDeviceContainer bridgeDev;
    bridgeDev = bridge.Install (backboneNodes.Get (i), NetDeviceContainer (apDev,
backboneDevices.Get (i)));

    // assign AP IP address to bridge, not wifi
    apInterface = ip.Assign (bridgeDev);

    // setup the STAs
    stack.Install (sta);
    mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                               "Mode", StringValue ("Time"),
                               "Time", StringValue ("2s"),
                               "Speed", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"),
                               "Bounds", RectangleValue (Rectangle (wifiX, wifiX+5.0,0.0,
(nStas+1)*5.0)));
    mobility.Install (sta);
    wifiMac.SetType ("ns3::StaWifiMac",
                     "Ssid", SsidValue (ssid),
                     "ActiveProbing", BooleanValue (false));
    staDev = wifi.Install (wifiPhy, wifiMac, sta);
    staInterface = ip.Assign (staDev);
```

```
      // save everything in containers.
      staNodes.push_back (sta);
      apDevices.push_back (apDev);
      apInterfaces.push_back (apInterface);
      staDevices.push_back (staDev);
      staInterfaces.push_back (staInterface);

      wifiX += 20.0;
    }

  Address dest;
  std::string protocol;
  if (sendIp)
    {
      dest = InetSocketAddress (staInterfaces[1].GetAddress (1), 1025);
      protocol = "ns3::UdpSocketFactory";
    }
  else
    {
      PacketSocketAddress tmp;
      tmp.SetSingleDevice (staDevices[0].Get (0)->GetIfIndex ());
      tmp.SetPhysicalAddress (staDevices[1].Get (0)->GetAddress ());
      tmp.SetProtocol (0x807);
      dest = tmp;
      protocol = "ns3::PacketSocketFactory";
    }

  OnOffHelper onoff = OnOffHelper (protocol, dest);
  onoff.SetConstantRate (DataRate ("500kb/s"));
  ApplicationContainer apps = onoff.Install (staNodes[0].Get (0));
  apps.Start (Seconds (0.5));
  apps.Stop (Seconds (3.0));

  wifiPhy.EnablePcap ("wifi-wired-bridging", apDevices[0]);
  wifiPhy.EnablePcap ("wifi-wired-bridging", apDevices[1]);

  if (writeMobility)
    {
      AsciiTraceHelper ascii;
      MobilityHelper::EnableAsciiAll (ascii.CreateFileStream ("wifi-wired-bridging.mob"));
    }

  Simulator::Stop (Seconds (5.0));
  Simulator::Run ();
  Simulator::Destroy ();
}
```

5. WIFI TO LTE(4G) CONNECTION

```
#include "ns3/lte-helper.h"
#include "ns3/epc-helper.h"

#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/csma-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/wimax-module.h"
#include "ns3/internet-module.h"
#include "ns3/global-route-manager.h"
#include "ns3/ipcs-classifier-record.h"
#include "ns3/service-flow.h"
#include <iostream>
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/point-to-point-helper.h"
#include <iomanip>
#include <string>
#include <fstream>
#include <vector>

NS_LOG_COMPONENT_DEFINE ("WimaxSimpleExample");

using namespace ns3;


int main (int argc, char *argv[])
{
Config::SetDefault ("ns3::LteAmc::AmcModel", EnumValue (LteAmc::PiroEW2010));
  bool verbose = false;

  int duration = 500, schedType = 0;


  uint16_t numberOfUEs=2;              //Default number of ues attached to each eNodeB

  Ptr<LteHelper> lteHelper;     //Define LTE
  Ptr<EpcHelper>  epcHelper;    //Define EPC
```

```
  NodeContainer remoteHostContainer;           //Define the Remote Host
  NetDeviceContainer internetDevices;   //Define the Network Devices in the Connection
between EPC and the remote host

  Ptr<Node> pgw;                          //Define the Packet Data Network Gateway(P-GW)
  Ptr<Node> remoteHost;         //Define the node of remote Host

  InternetStackHelper internet;               //Define the internet stack
  PointToPointHelper p2ph;                        //Define Connection between EPC and the
Remote Host
  Ipv4AddressHelper ipv4h;                        //Ipv4 address helper
  Ipv4StaticRoutingHelper ipv4RoutingHelper;    //Ipv4 static routing helper
  Ptr<Ipv4StaticRouting> remoteHostStaticRouting;

  Ipv4InterfaceContainer internetIpIfaces;      //Ipv4 interfaces

  CommandLine cmd;
  cmd.AddValue ("scheduler", "type of scheduler to use with the network devices", schedType);
  cmd.AddValue ("duration", "duration of the simulation in seconds", duration);
  cmd.AddValue ("verbose", "turn on all WimaxNetDevice log components", verbose);
  cmd.Parse (argc, argv);
  LogComponentEnable ("UdpClient", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpServer", LOG_LEVEL_INFO);
  //LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  //LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer ssNodes;
  NodeContainer bsNodes;

  ssNodes.Create (2);
  bsNodes.Create (1);


uint32_t nCsma = 3;

  NodeContainer p2pNodes;
  p2pNodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer p2pDevices;
  p2pDevices = pointToPoint.Install (p2pNodes);

  NodeContainer csmaNodes;
```

```
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
        "Ssid", SsidValue (ssid),
        "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, ssNodes);

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

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);

MobilityHelper mobility1;

mobility1.SetPositionAllocator ("ns3::GridPositionAllocator",
                        "MinX", DoubleValue (0.0),
                        "MinY", DoubleValue (0.0),
                        "DeltaX", DoubleValue (5.0),
                        "DeltaY", DoubleValue (10.0),
                        "GridWidth", UintegerValue (3),
                        "LayoutType", StringValue ("RowFirst"));
```

```
  mobility1.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility1.Install (wifiApNode);
(wifiApNode.Get(0) -> GetObject<ConstantPositionMobilityModel>()) ->
SetPosition(Vector(100.0, 501.0, 0.0));
  InternetStackHelper stack1;
  stack1.Install (csmaNodes);
  stack1.Install (wifiApNode);
  stack1.Install (ssNodes);

  Ipv4AddressHelper address1;

  address1.SetBase ("11.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer p2pInterfaces;
  p2pInterfaces = address1.Assign (p2pDevices);

  address1.SetBase ("11.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer csmaInterfaces;
  csmaInterfaces = address1.Assign (csmaDevices);

  address1.SetBase ("11.1.3.0", "255.255.255.0");
  address1.Assign (staDevices);
  address1.Assign (apDevices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps1 = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps1.Start (Seconds (1.0));
  serverApps1.Stop (Seconds (duration+0.1));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1000));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps1 =
    echoClient.Install (ssNodes.Get (0));
  clientApps1.Start (Seconds (2.0));
  clientApps1.Stop (Seconds (duration+0.1));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();


  //pointToPoint.EnablePcapAll ("third");
  phy.EnablePcap ("third", apDevices.Get (0));
  //csma.EnablePcap ("third", csmaDevices.Get (0), true);

    lteHelper = CreateObject<LteHelper> ();
```

```
   epcHelper = CreateObject<EpcHelper> ();

   lteHelper->SetEpcHelper (epcHelper);
   lteHelper->SetSchedulerType("ns3::RrFfMacScheduler");
   lteHelper->SetAttribute ("PathlossModel",
                    StringValue ("ns3::FriisPropagationLossModel"));
    pgw = epcHelper->GetPgwNode ();

   remoteHostContainer.Create (1);
   remoteHost = remoteHostContainer.Get (0);
   internet.Install (remoteHostContainer);

    p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));
    p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
    p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));
    internetDevices = p2ph.Install (pgw, remoteHost);

    ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
    internetIpIfaces = ipv4h.Assign (internetDevices);

    remoteHostStaticRouting = ipv4RoutingHelper.GetStaticRouting (remoteHost->GetObject<Ipv4> ());
    remoteHostStaticRouting->AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask ("255.0.0.0"),
1);

    std::cout << "2. Installing LTE+EPC+remotehost. Done!" << std::endl;


MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc;
  positionAlloc = CreateObject<ListPositionAllocator> ();

  positionAlloc->Add (Vector (0.0, 500.0, 0.0)); //STA

  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
  mobility.Install(ssNodes.Get(0));

  Ptr<ConstantVelocityMobilityModel> cvm = ssNodes.Get(0)-
>GetObject<ConstantVelocityMobilityModel>();
  cvm->SetVelocity(Vector (5, 0, 0)); //move to left to right 10.0m/s

  positionAlloc = CreateObject<ListPositionAllocator> ();

  positionAlloc->Add (Vector (0.0, 500.0, 10.0)); //MAG1AP
  positionAlloc->Add (Vector (0.0, 510.0, 0.0));  //MAG2AP

  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
```

```
  mobility.Install (NodeContainer(bsNodes.Get(0),ssNodes.Get(1)));

NetDeviceContainer ssDevs, bsDevs;

        bsDevs = lteHelper->InstallEnbDevice (bsNodes);
        ssDevs=lteHelper->InstallUeDevice (ssNodes);

        for (uint16_t j=0; j < numberOfUEs; j++)
        {
              lteHelper->Attach (ssDevs.Get(j), bsDevs.Get(0));
        }


        Ipv4InterfaceContainer iueIpIface;
        iueIpIface = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ssDevs));

  lteHelper->ActivateEpsBearer (ssDevs, EpsBearer (EpsBearer::NGBR_VIDEO_TCP_DEFAULT),
EpcTft::Default ());

UdpServerHelper udpServer;
  ApplicationContainer serverApps;
  UdpClientHelper udpClient;
  ApplicationContainer clientApps;
  udpServer = UdpServerHelper (100);

  serverApps = udpServer.Install (ssNodes.Get (0));
  serverApps.Start (Seconds (6));
  serverApps.Stop (Seconds (duration));

  udpClient = UdpClientHelper (iueIpIface.GetAddress (0), 100);
  udpClient.SetAttribute ("MaxPackets", UintegerValue (200000));
  udpClient.SetAttribute ("Interval", TimeValue (Seconds (0.004)));
  udpClient.SetAttribute ("PacketSize", UintegerValue (1024));

  clientApps = udpClient.Install (remoteHost);
  clientApps.Start (Seconds (6));
  clientApps.Stop (Seconds (duration));
    lteHelper->EnableTraces ();

  NS_LOG_INFO ("Starting simulation.....");
Simulator::Stop(Seconds(duration));

  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
  return 0;
}
```

6. CREATING A SIMPLE WIFI ADHOC GRID

n20 n21 n22 n23 n24
n15 n16 n17 n18 n19
n10 n11 n12 n13 n14
n5 n6 n7 n8 n9
n0 n1 n2 n3 n4
The layout is affected by the parameters given to GridPositionAllocator;
By default, GridWidth is 5 and numNodes is 25..

Flow 1: 0->24
Flow 2: 20->4
Flow 3: 10->4

STEPS:

1. Setup a 5x5 wireless adhoc network with a grid. You may use examples/wireless/wifi-simple-adhoc-grid.cc as a base.

2. Install the OLSR routing protocol.

3. Setup three UDP traffic flows, one along each diagonal and one along the middle (at high rates of transmission).

4. Setup the ns-3 flow monitor for each of these flows.

5. Now schedule each of the flows at times 1s, 1.5s, and 2s.

6. Now using the flow monitor, observe the throughput of each of the UDP flows. Furthermore, use the tracing mechanism to monitor the number of packet collisions/drops at intermediary nodes. Around which nodes are most of the collisions/drops happening?

7. Now repeat the experiment with RTS/CTS enabled on the wifi devices.

8. Show the difference in throughput and packet drops if any.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/olsr-helper.h"
#include "ns3/flow-monitor-module.h"
```

```
#include "myapp.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

NS_LOG_COMPONENT_DEFINE ("Lab5");

using namespace ns3;

uint32_t MacTxDropCount, PhyTxDropCount, PhyRxDropCount;

void
MacTxDrop(Ptr<const Packet> p)
{
  NS_LOG_INFO("Packet Drop");
  MacTxDropCount++;
}

void
PrintDrop()
{
  std::cout << Simulator::Now().GetSeconds() << "\t" << MacTxDropCount << "\t"<<
PhyTxDropCount << "\t" << PhyRxDropCount << "\n";
  Simulator::Schedule(Seconds(5.0), &PrintDrop);
}

void
PhyTxDrop(Ptr<const Packet> p)
{
  NS_LOG_INFO("Packet Drop");
  PhyTxDropCount++;
}
void
PhyRxDrop(Ptr<const Packet> p)
{
  NS_LOG_INFO("Packet Drop");
  PhyRxDropCount++;
}
int main (int argc, char *argv[])
{
  std::string phyMode ("DsssRate1Mbps");
  double distance = 500;  // m
  uint32_t numNodes = 25;  // by default, 5x5
  double interval = 0.001; // seconds
  uint32_t packetSize = 600; // bytes
```

```
  uint32_t numPackets = 10000000;
  std::string rtslimit = "1500";
  CommandLine cmd;

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
  cmd.AddValue ("distance", "distance (m)", distance);
  cmd.AddValue ("packetSize", "distance (m)", packetSize);
  cmd.AddValue ("rtslimit", "RTS/CTS Threshold (bytes)", rtslimit);
  cmd.Parse (argc, argv);
  // Convert to time object
  Time interPacketInterval = Seconds (interval);

  // turn off RTS/CTS for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue
(rtslimit));
  // Fix non-unicast data rate to be the same as that of unicast
  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue
(phyMode));

  NodeContainer c;
  c.Create (numNodes);

  // The below set of helpers will help us to put together the wifi NICs we want
  WifiHelper wifi;

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
  // set it to zero; otherwise, gain will be added
  wifiPhy.Set ("RxGain", DoubleValue (-10) );
  // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
  wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a non-QoS upper mac, and disable rate control
  NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));
  // Set it to adhoc mode
  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);
```

```
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (distance),
                                 "DeltaY", DoubleValue (distance),
                                 "GridWidth", UintegerValue (5),
                                 "LayoutType", StringValue ("RowFirst"));
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (c);

  // Enable OLSR
  OlsrHelper olsr;

  Ipv4ListRoutingHelper list;
  list.Add (olsr, 10);

  InternetStackHelper internet;
  internet.SetRoutingHelper (list); // has effect on the next Install ()
  internet.Install (c);

  Ipv4AddressHelper ipv4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer ifcont = ipv4.Assign (devices);

  // Create Apps

  uint16_t sinkPort = 6; // use the same for all apps

  // UDP connection from N0 to N24

   Address sinkAddress1 (InetSocketAddress (ifcont.GetAddress (24), sinkPort)); // interface of
n24
   PacketSinkHelper packetSinkHelper1 ("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
   ApplicationContainer sinkApps1 = packetSinkHelper1.Install (c.Get (24)); //n2 as sink
   sinkApps1.Start (Seconds (0.));
   sinkApps1.Stop (Seconds (100.));

   Ptr<Socket> ns3UdpSocket1 = Socket::CreateSocket (c.Get (0),
UdpSocketFactory::GetTypeId ()); //source at n0

   // Create UDP application at n0
   Ptr<MyApp> app1 = CreateObject<MyApp> ();
   app1->Setup (ns3UdpSocket1, sinkAddress1, packetSize, numPackets, DataRate ("1Mbps"));
```

```
  c.Get (0)->AddApplication (app1);
  app1->SetStartTime (Seconds (31.));
  app1->SetStopTime (Seconds (100.));

  // UDP connection from N10 to N14

   Address sinkAddress2 (InetSocketAddress (ifcont.GetAddress (14), sinkPort)); // interface of
n14
   PacketSinkHelper packetSinkHelper2 ("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
   ApplicationContainer sinkApps2 = packetSinkHelper2.Install (c.Get (14)); //n14 as sink
   sinkApps2.Start (Seconds (0.));
   sinkApps2.Stop (Seconds (100.));

   Ptr<Socket> ns3UdpSocket2 = Socket::CreateSocket (c.Get (10),
UdpSocketFactory::GetTypeId ()); //source at n10

   // Create UDP application at n10
   Ptr<MyApp> app2 = CreateObject<MyApp> ();
   app2->Setup (ns3UdpSocket2, sinkAddress2, packetSize, numPackets, DataRate ("1Mbps"));
   c.Get (10)->AddApplication (app2);
   app2->SetStartTime (Seconds (31.5));
   app2->SetStopTime (Seconds (100.));

   // UDP connection from N20 to N4

    Address sinkAddress3 (InetSocketAddress (ifcont.GetAddress (4), sinkPort)); // interface of
n4
    PacketSinkHelper packetSinkHelper3 ("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
    ApplicationContainer sinkApps3 = packetSinkHelper3.Install (c.Get (4)); //n2 as sink
    sinkApps3.Start (Seconds (0.));
    sinkApps3.Stop (Seconds (100.));

    Ptr<Socket> ns3UdpSocket3 = Socket::CreateSocket (c.Get (20),
UdpSocketFactory::GetTypeId ()); //source at n20

    // Create UDP application at n20
    Ptr<MyApp> app3 = CreateObject<MyApp> ();
    app3->Setup (ns3UdpSocket3, sinkAddress3, packetSize, numPackets, DataRate
("1Mbps"));
    c.Get (20)->AddApplication (app3);
    app3->SetStartTime (Seconds (32.));
    app3->SetStopTime (Seconds (100.));

 // Install FlowMonitor on all nodes
```

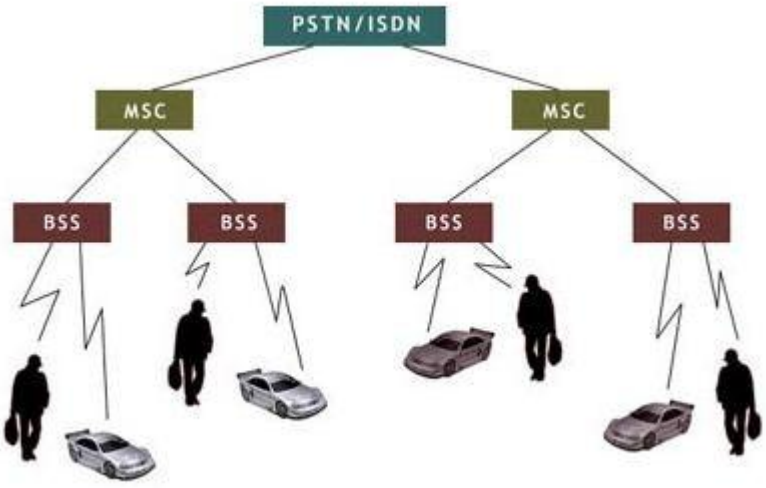
```
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll();


  // Trace Collisions

Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacTx
Drop", MakeCallback(&MacTxDrop));

Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/PhyRxD
rop", MakeCallback(&PhyRxDrop));

Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/PhyTxD
rop", MakeCallback(&PhyTxDrop));

  Simulator::Schedule(Seconds(5.0), &PrintDrop);

  Simulator::Stop (Seconds (100.0));
  Simulator::Run ();

  PrintDrop();

  // Print per flow statistics
  monitor->CheckForLostPackets ();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmon.GetClassifier ());
  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter !=
stats.end (); ++iter)
    {
              Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);

     if ((t.sourceAddress == Ipv4Address("10.1.1.1") && t.destinationAddress ==
Ipv4Address("10.1.1.25"))
            || (t.sourceAddress == Ipv4Address("10.1.1.11") && t.destinationAddress ==
Ipv4Address("10.1.1.15"))
            || (t.sourceAddress == Ipv4Address("10.1.1.21") && t.destinationAddress ==
Ipv4Address("10.1.1.5")))
       {
              NS_LOG_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress
<< " Dst Addr " << t.destinationAddress);
              NS_LOG_UNCOND("Tx Packets = " << iter->second.txPackets);
              NS_LOG_UNCOND("Rx Packets = " << iter->second.rxPackets);
```

```
                NS_LOG_UNCOND("Throughput: " << iter->second.rxBytes * 8.0 / (iter->second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1024 << " Kbps");
        }
    }
  monitor->SerializeToXmlFile("lab-5.flowmon", true, true);

  Simulator::Destroy ();

  return 0;
}
```

7. To Study Architecture of GSM.

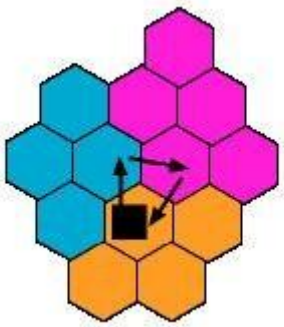
System Architecture

- A GSM network consists of several functional entities, whose functions and interfaces are defined. The GSM network can be divided into following broad parts.
    - The Mobile Station (MS)
    - The Base Station Subsystem (BSS)
    - The Network Switching Subsystem (NSS)
    - The Operation Support Subsystem OSS)
- A GSM Public Land Mobile Network (PLMN) consists of at least one Service Area controlled by a Mobile Switching Center (MSC) connected to the Public Switched Telephone Network (PSTN)
- The architecture of a GSM Public Land Mobile Network (PLMN)

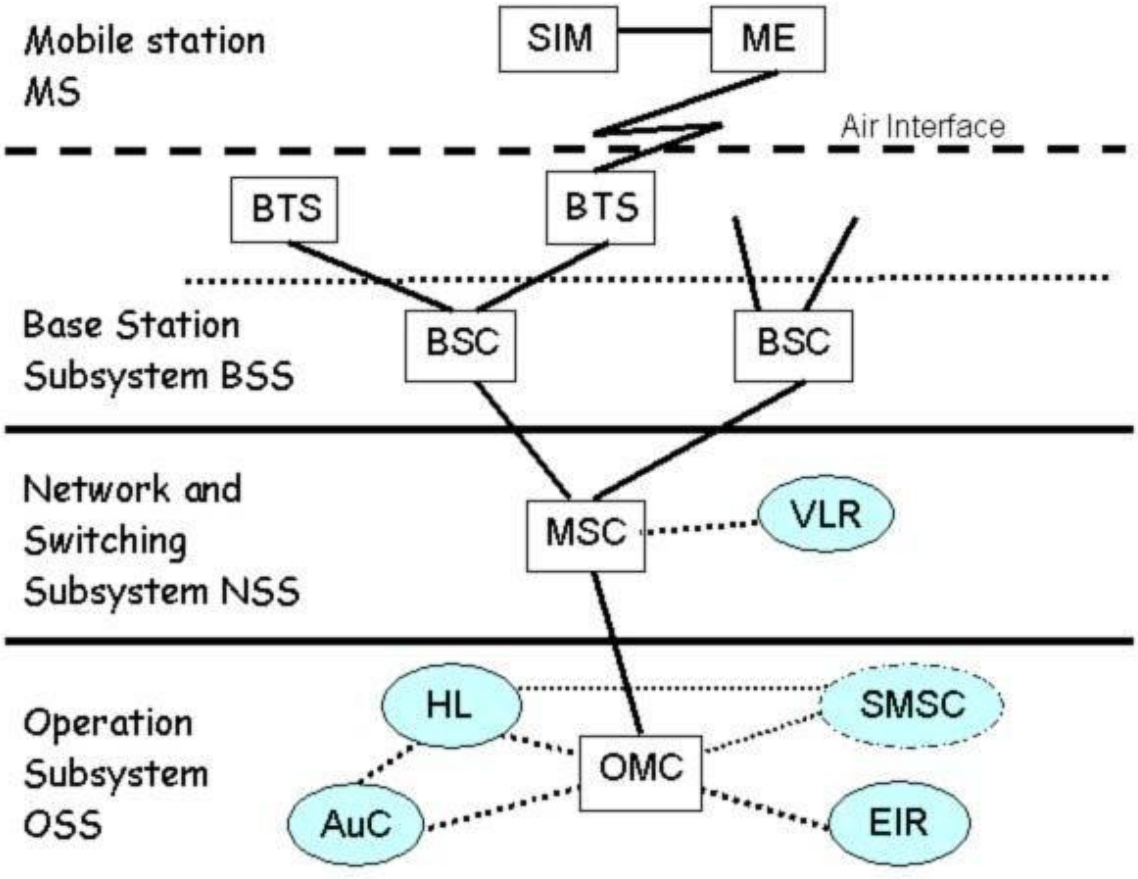


- A Base Station Subsystem (BSS) consists of o A Base Station Controller (BSC)
    - At least one radio access point or Base Transceiver Station (BTS) for Mobile Stations (MS), which are mobile phones or other handheld devices (for example PDA computers) with phone interface.
- A BTS, with its aerial and associated radio frequency components, is the actual transmission and reception component. A Network Cell is the area of radio coverage by one BTS. One or more BTSs are in turn managed by a BSC. A network cell cluster covered by one or several BSSs can be managed as a Location Area (LA). All these BSSs must however be controlled by a single MSC.

- Figure shows three LAs of 3, 4 and 4 cells respectively with a MS moving across cell and LA boundaries where a MS moving across cell and LA boundaries. 3 LAs consisting of 4 and 5 cells respectively are shown.

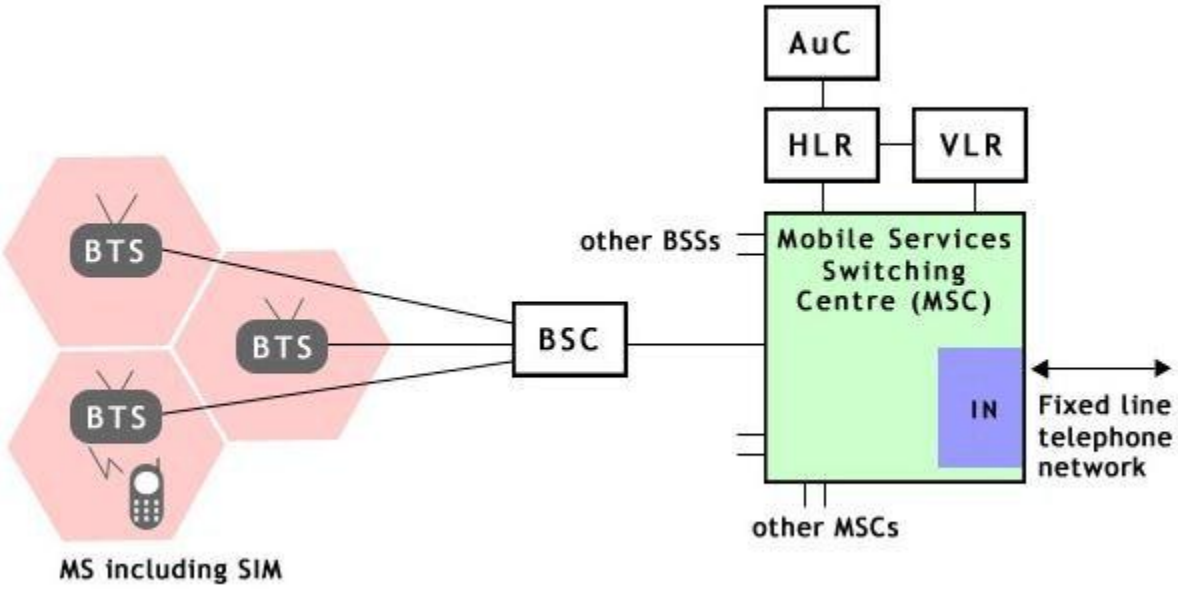


- A more detailed architecture of a single MSC controlled Service Area is outlined in figure below.



- components of the tree GSM network subsystems
- Radio Subsystem (RSS) consisting of the BSSs and all BSS connected MS devices.
  - Network and Switching Subsystem (NSS)
  - Operation Subsystem (OSS)
- Specified in GSM 01.02 (‘General description of a GSM Public Land Mobile Network(PLMN)’) and the GSM components are,

  ME = Mobile Equipment
  BTS = Base Receiving Station
  BSC = Base Station Controller
  MSC = Mobile Switching Center
  VLR = Visitor Location Register
  OMC = Operation and Maintenance Center
  AuC = Authentication Center
  HLR = Home Location Register
  EIR = Equipment Identity Register
  SMSC = Short Message Service Centre
- A MSC is also through a Gateway MSC (GMSC) connected to other MSCs and to the Public Switched Telephone Network (PSTN) with the Integrated Services Digital Network (ISDN) option. The Inter-Working Function (IWF) of GMSC connects the circuit switched data paths of a GSM network with the PSTN/ISDN. A GMSC is usually integrated in an MSC.
- Basic GSM network components,



Mobile Station = MS
Subscriber Identity Module = SIM
Base Transcriver Station = BTS
Base Station Controller = BSC

HLR = Home Location Register
VLR = Visited Location Register
AuC = Authentication Centre
IN = Interrogating Node