Northeastern University

Department of Electrical and Computer Engineering

# EECE 4638: **Wireless Design and Simulation**

Lecturer: ***Rahman Doost-Mohammady***

TAs:

**Subhramoy Mohanti**

Lab # 4 : NS-3 LTE Handover

Group:

*Taylor Skilling (001171439)*

*Chris Kenyon (001112428)*

Semester: Summer II 2015

Date: August 10th, 2015

Lab Session: *Monday-Thursday, 3:20 PM*

##### Lab Location: 429 Dana Research Lab, Northeastern University, Boston, MA 02115

##### Content Page

##### Introduction / Objective 3

##### Design Approach 4

##### Results and Analysis

##### Appendix A

1. **Introduction / Objectives**

The intent of this experiment is to observe what happens when a UE moves between different eNBs in an LTE Network environment. We want to see how the connection handover affects the properties of the network between the base stations and the unit switching between them. Using UDP applications, as the UE moves away from the range of one eNB and approaches another eNB, the SNR associated with the serving eNB goes lower while the SNR of the new eNB goes higher. When the signal to noise ratio from the serving eNB goes under a threshold value, the handover process will be initiated and the existing network connection of that UE will be transferred to the next eNB with the best SNR.

1. **Design Approach**

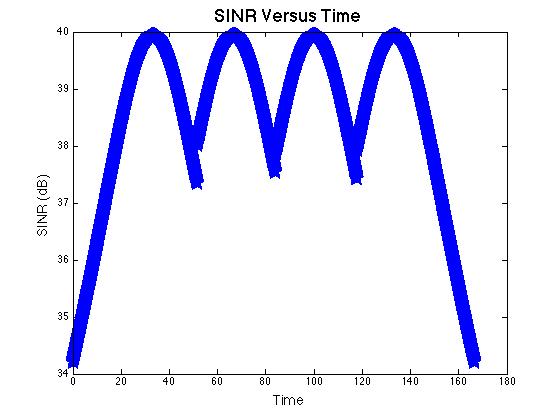
We will design this simulation using NS3 along with the help of some provided libraries. For setting up the LTE simulation, the LteHelper class from *ns3/lte-module.h*, *ns3/core-module.h, ns3/network-module.h, ns3/internet-module.h, ns3/applications-module.h, ns3/point-to-point-module.h, ns3/config-store-module.h*.

To set up the simulation, first we set up the UDP client with by setting up the interval and max packets. We continue to setup the LteHelper object, the NodeContainer, the Internet and installing it, the PointToPointHelper, NetDeviceContainer, the Ipv4AddressHelper, and the Ipv4StaticRoutingHelper. We then set up the network like from the previous lab, setting up the NodeContainers for the UE nodes and the eNB nodes, setting up mobility and position allocators for each of them, and then attaching them. We will use four ENB nodes to represent the stations each at a distance of 500 from each other, and just one UE node to represent the end users. The ue and its mobility model then had its velocity set with a vector of speed 15. The Udp connection with the eps barrier and handover protocol.

For the simulation, the PhyTraces, MacTraces, RlcTraces, PdcpTraces, UE nodes must then be attached to one of the ENB nodes, the data radio bearer needs to be activated, the data traces are enabled, and finally the simulation is run for roughly 3 minutes. See Lab\_4.cc in the Appendix for the code with more details.

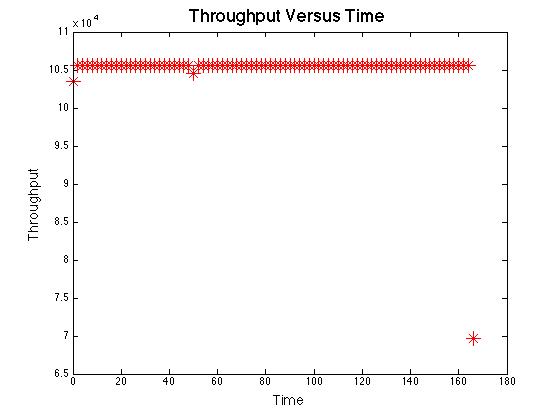
Running waf to build the program will simulate the network and produce text files with the statistics. MATLAB was used to parse the relevant information from simulation outputs, compute the average of relevant measurements, and plot the results. See Measurements.m in the Appendix.

1. **Results and Analysis**



**Figure 1**

From analyzing the data in Figure 1, we immediately see that there are very clear relationships between the SINR and the power as the UE moves past the eNB nodes. The signal to noise ratio steadily increases until it reaches its peak value at the positions of the eNB nodes. Immediately after, the SINR decreases at a similar rate it previously was increasing. It then decreases until it reaches the range of the next eNB node when it begins to increase toward the same peak. This pattern repeats as the UE travels by each eNBs, maintaining a relatively strong signal the whole time even during the handover.



**Figure 2**

Alongside these conclusions, we see from Figure 2 that along with maintaining a pretty steady signal even in the handover process, it also maintains a consistent throughput in the system. There was a very minor drop in throughput at 50 seconds, which aligns with the first handover. This blip was insignificant and the overall trend was that the throughput maintains a strong and consistent rate around 10.5 x 104 [units?]. Thus the even with multiple handovers, the LTE network is reliable enough to maintain a high and consistent throughput.

Overall, we see that this handover LTE network is effective in an ideal setting without the random interference over the actual distance that this network would span in a real world application.

1. **Appendix A**

**Lab\_4.cc**

/\*

\* Lte lab 2

\*/

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/mobility-module.h"

#include "ns3/lte-module.h"

#include "ns3/applications-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/config-store-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("Lte\_Handover");

void

NotifyConnectionEstablishedUe (std::string context,

uint64\_t imsi,

uint16\_t cellid,

uint16\_t rnti)

{

std::cout << context

<< " UE IMSI " << imsi

<< ": connected to CellId " << cellid

<< " with RNTI " << rnti

<< std::endl;

}

void

NotifyHandoverStartUe (std::string context,

uint64\_t imsi,

uint16\_t cellid,

uint16\_t rnti,

uint16\_t targetCellId)

{

std::cout << context

<< " UE IMSI " << imsi

<< ": previously connected to CellId " << cellid

<< " with RNTI " << rnti

<< ", doing handover to CellId " << targetCellId

<< std::endl;

}

void

NotifyHandoverEndOkUe (std::string context,

uint64\_t imsi,

uint16\_t cellid,

uint16\_t rnti)

{

std::cout << context

<< " UE IMSI " << imsi

<< ": successful handover to CellId " << cellid

<< " with RNTI " << rnti

<< std::endl;

}

void

NotifyConnectionEstablishedEnb (std::string context,

uint64\_t imsi,

uint16\_t cellid,

uint16\_t rnti)

{

std::cout << context

<< " eNB CellId " << cellid

<< ": successful connection of UE with IMSI " << imsi

<< " RNTI " << rnti

<< std::endl;

}

void

NotifyHandoverStartEnb (std::string context,

uint64\_t imsi,

uint16\_t cellid,

uint16\_t rnti,

uint16\_t targetCellId)

{

std::cout << context

<< " eNB CellId " << cellid

<< ": start handover of UE with IMSI " << imsi

<< " RNTI " << rnti

<< " to CellId " << targetCellId

<< std::endl;

}

void

NotifyHandoverEndOkEnb (std::string context,

uint64\_t imsi,

uint16\_t cellid,

uint16\_t rnti)

{

std::cout << context

<< " eNB CellId " << cellid

<< ": completed handover of UE with IMSI " << imsi

<< " RNTI " << rnti

<< std::endl;

}

/\*\*

\* Sample simulation script for an automatic X2-based handover based on the RSRQ measures.

\* It instantiates two eNodeB, attaches one UE to the 'source' eNB.

\* The UE moves between both eNBs, it reports measures to the serving eNB and

\* the 'source' (serving) eNB triggers the handover of the UE towards

\* the 'target' eNB when it considers it is a better eNB.

\*/

int

main (int argc, char \*argv[])

{

uint16\_t numberOfUes = 1;

uint16\_t numberOfEnbs = 4; //based on slides

uint16\_t numBearersPerUe = 1;

double distance = 500.0; // m

double yForUe = 200.0; // m

double speed = 15; // m/s

double simTime = (double)(numberOfEnbs + 1) \* distance / speed; // 2500 m / 15 m/s = 166.66 secs

double enbTxPowerDbm = 30.0;

// change some default attributes so that they are reasonable for

// this scenario, but do this before processing command line

// arguments, so that the user is allowed to override these settings

Config::SetDefault ("ns3::UdpClient::Interval", TimeValue (MilliSeconds (10)));

Config::SetDefault ("ns3::UdpClient::MaxPackets", UintegerValue (1000000));

Config::SetDefault ("ns3::LteHelper::UseIdealRrc", BooleanValue (true));

// Command line arguments

//CommandLine cmd;

//cmd.AddValue ("simTime", "Total duration of the simulation (in seconds)", simTime);

//cmd.AddValue ("speed", "Speed of the UE (default = 20 m/s)", speed);

//cmd.AddValue ("enbTxPowerDbm", "TX power [dBm] used by HeNBs (defalut = 46.0)", enbTxPowerDbm);

//cmd.Parse (argc, argv);

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

Ptr<PointToPointEpcHelper> epcHelper = CreateObject<PointToPointEpcHelper> ();

lteHelper->SetEpcHelper (epcHelper);

lteHelper->SetSchedulerType ("ns3::RrFfMacScheduler");

lteHelper->SetHandoverAlgorithmType ("ns3::A2A4RsrqHandoverAlgorithm");

lteHelper->SetHandoverAlgorithmAttribute ("ServingCellThreshold",

UintegerValue (30));

lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset",

UintegerValue (1));

// lteHelper->SetHandoverAlgorithmType ("ns3::A3RsrpHandoverAlgorithm");

// lteHelper->SetHandoverAlgorithmAttribute ("Hysteresis",

// DoubleValue (3.0));

// lteHelper->SetHandoverAlgorithmAttribute ("TimeToTrigger",

// TimeValue (MilliSeconds (256)));

Ptr<Node> pgw = epcHelper->GetPgwNode ();

// Create a single RemoteHost

NodeContainer remoteHostContainer;

remoteHostContainer.Create (1);

Ptr<Node> remoteHost = remoteHostContainer.Get (0);

InternetStackHelper internet;

internet.Install (remoteHostContainer);

// Create the Internet

PointToPointHelper p2ph;

p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));

p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));

p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));

NetDeviceContainer internetDevices = p2ph.Install (pgw, remoteHost);

Ipv4AddressHelper ipv4h;

ipv4h.SetBase ("1.0.0.0", "255.0.0.0");

Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (internetDevices);

Ipv4Address remoteHostAddr = internetIpIfaces.GetAddress (1);

// Routing of the Internet Host (towards the LTE network)

Ipv4StaticRoutingHelper ipv4RoutingHelper;

Ptr<Ipv4StaticRouting> remoteHostStaticRouting = ipv4RoutingHelper.GetStaticRouting (remoteHost->GetObject<Ipv4> ());

// interface 0 is localhost, 1 is the p2p device

remoteHostStaticRouting->AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask ("255.0.0.0"), 1);

/\*

\* Network topology:

\*

\* | + --------------------------------------------------------->

\* | UE

\* |

\* | d d d

\* y | |-------------------x-------------------x-------------------

\* | | eNodeB eNodeB

\* | d |

\* | |

\* | | d = distance

\* o (0, 0, 0) y = yForUe

\*/

NodeContainer ueNodes;

NodeContainer enbNodes;

enbNodes.Create (numberOfEnbs);

ueNodes.Create (numberOfUes);

// Install Mobility Model in eNB

Ptr<ListPositionAllocator> enbPositionAlloc = CreateObject<ListPositionAllocator> ();

for (uint16\_t i = 0; i < numberOfEnbs; i++)

{

Vector enbPosition (distance \* (i + 1), distance, 0);

enbPositionAlloc->Add (enbPosition);

}

MobilityHelper enbMobility;

enbMobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

enbMobility.SetPositionAllocator (enbPositionAlloc);

enbMobility.Install (enbNodes);

// Install Mobility Model in UE

MobilityHelper ueMobility;

ueMobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");

ueMobility.Install (ueNodes);

ueNodes.Get (0)->GetObject<MobilityModel> ()->SetPosition (Vector (0, yForUe, 0));

ueNodes.Get (0)->GetObject<ConstantVelocityMobilityModel> ()->SetVelocity (Vector (speed, 0, 0));

// Install LTE Devices in eNB and UEs

Config::SetDefault ("ns3::LteEnbPhy::TxPower", DoubleValue (enbTxPowerDbm));

NetDeviceContainer enbLteDevs = lteHelper->InstallEnbDevice (enbNodes);

NetDeviceContainer ueLteDevs = lteHelper->InstallUeDevice (ueNodes);

// Install the IP stack on the UEs

internet.Install (ueNodes);

Ipv4InterfaceContainer ueIpIfaces;

ueIpIfaces = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ueLteDevs));

// Assign IP address to UEs, and install applications

for (uint32\_t u = 0; u < ueNodes.GetN (); ++u)

{

Ptr<Node> ueNode = ueNodes.Get (u);

// Set the default gateway for the UE

Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ueNode->GetObject<Ipv4> ());

ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);

}

// Attach all UEs to the first eNodeB

//for (uint16\_t i = 0; i < numberOfUes; i++)

// {

//attach manually

lteHelper->Attach (ueLteDevs.Get (0), enbLteDevs.Get (0));

// }

NS\_LOG\_LOGIC ("setting up applications");

// Install and start applications on UEs and remote host

uint16\_t dlPort = 10000;

uint16\_t ulPort = 20000;

// randomize a bit start times to avoid simulation artifacts

// (e.g., buffer overflows due to packet transmissions happening

// exactly at the same time)

Ptr<UniformRandomVariable> startTimeSeconds = CreateObject<UniformRandomVariable> ();

startTimeSeconds->SetAttribute ("Min", DoubleValue (0));

startTimeSeconds->SetAttribute ("Max", DoubleValue (0.010));

for (uint32\_t u = 0; u < numberOfUes; ++u)

{

Ptr<Node> ue = ueNodes.Get (u);

// Set the default gateway for the UE

Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ue->GetObject<Ipv4> ());

ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);

for (uint32\_t b = 0; b < numBearersPerUe; ++b)

{

++dlPort;

++ulPort;

ApplicationContainer clientApps;

ApplicationContainer serverApps;

NS\_LOG\_LOGIC ("installing UDP DL app for UE " << u);

UdpClientHelper dlClientHelper (ueIpIfaces.GetAddress (u), dlPort);

clientApps.Add (dlClientHelper.Install (remoteHost));

PacketSinkHelper dlPacketSinkHelper ("ns3::UdpSocketFactory",

InetSocketAddress (Ipv4Address::GetAny (), dlPort));

serverApps.Add (dlPacketSinkHelper.Install (ue));

NS\_LOG\_LOGIC ("installing UDP UL app for UE " << u);

UdpClientHelper ulClientHelper (remoteHostAddr, ulPort);

clientApps.Add (ulClientHelper.Install (ue));

PacketSinkHelper ulPacketSinkHelper ("ns3::UdpSocketFactory",

InetSocketAddress (Ipv4Address::GetAny (), ulPort));

serverApps.Add (ulPacketSinkHelper.Install (remoteHost));

Ptr<EpcTft> tft = Create<EpcTft> ();

EpcTft::PacketFilter dlpf;

dlpf.localPortStart = dlPort;

dlpf.localPortEnd = dlPort;

tft->Add (dlpf);

EpcTft::PacketFilter ulpf;

ulpf.remotePortStart = ulPort;

ulpf.remotePortEnd = ulPort;

tft->Add (ulpf);

EpsBearer bearer (EpsBearer::NGBR\_VIDEO\_TCP\_DEFAULT);

lteHelper->ActivateDedicatedEpsBearer (ueLteDevs.Get (u), bearer, tft);

Time startTime = Seconds (startTimeSeconds->GetValue ());

serverApps.Start (startTime);

clientApps.Start (startTime);

} // end for b

}

// Add X2 inteface

lteHelper->AddX2Interface (enbNodes);

// X2-based Handover

//lteHelper->HandoverRequest (Seconds (0.100), ueLteDevs.Get (0), enbLteDevs.Get (0), enbLteDevs.Get (1));

// Uncomment to enable PCAP tracing

// p2ph.EnablePcapAll("lena-x2-handover-measures");

lteHelper->EnablePhyTraces ();

lteHelper->EnableMacTraces ();

lteHelper->EnableRlcTraces ();

lteHelper->EnablePdcpTraces ();

Ptr<RadioBearerStatsCalculator> rlcStats = lteHelper->GetRlcStats ();

rlcStats->SetAttribute ("EpochDuration", TimeValue (Seconds (1.0)));

Ptr<RadioBearerStatsCalculator> pdcpStats = lteHelper->GetPdcpStats ();

pdcpStats->SetAttribute ("EpochDuration", TimeValue (Seconds (1.0)));

// connect custom trace sinks for RRC connection establishment and handover notification

Config::Connect ("/NodeList/\*/DeviceList/\*/LteEnbRrc/ConnectionEstablished",

MakeCallback (&NotifyConnectionEstablishedEnb));

Config::Connect ("/NodeList/\*/DeviceList/\*/LteUeRrc/ConnectionEstablished",

MakeCallback (&NotifyConnectionEstablishedUe));

Config::Connect ("/NodeList/\*/DeviceList/\*/LteEnbRrc/HandoverStart",

MakeCallback (&NotifyHandoverStartEnb));

Config::Connect ("/NodeList/\*/DeviceList/\*/LteUeRrc/HandoverStart",

MakeCallback (&NotifyHandoverStartUe));

Config::Connect ("/NodeList/\*/DeviceList/\*/LteEnbRrc/HandoverEndOk",

MakeCallback (&NotifyHandoverEndOkEnb));

Config::Connect ("/NodeList/\*/DeviceList/\*/LteUeRrc/HandoverEndOk",

MakeCallback (&NotifyHandoverEndOkUe));

Simulator::Stop (Seconds (simTime));

Simulator::Run ();

// GtkConfigStore config;

// config.ConfigureAttributes ();

Simulator::Destroy ();

return 0;

}

**Measurements.m**

%% Lab 4 Measurements

% Declare file names

sinrFile = 'DlRsrpSinrStats.txt';

rlcFile = 'DlRlcStats.txt';

% Load SINR from Phy Traces

fid = fopen(sinrFile);

C = textscan(fid,'%f32%f32%f32%f32%f32%f32','HeaderLines',1);

fclose(fid);

% Chop data appropriately

SinrTimes = C{1};

SinrTimes = SinrTimes(1:2:length(SinrTimes));

SinrData = C{6};

SinrData = SinrData(1:2:length(SinrData));

% Convert to dB

SinrData = 10\*log10(SinrData);

% Load RLC stats

fid = fopen(rlcFile);

D = textscan(fid,'%f%f%f%f%f%f%f%f%f%f%[^\n]','HeaderLines',1, 'Delimiter',' ',...

'MultipleDelimsAsOne',1);

fclose(fid);

% Chop data appropriately

rlcTimes = D{1};

rlcTimes = rlcTimes(1:2:length(rlcTimes));

rlcRxBytes = D{10};

rlcRxBytes = rlcRxBytes(1:2:length(rlcRxBytes));

fprintf('Completed data parsing\n');

% Plot Throughput versus Time

figure(1)

plot(rlcTimes,rlcRxBytes,'r\*','markersize',14)

xlabel('Time','FontSize',14)

ylabel('Throughput','FontSize',14)

%xlim([20,50])

title('Throughput Versus Time','FontSize',18)

% Plot SINR versus Time

figure(2)

plot(SinrTimes,SinrData,'bp','markersize',14)

xlabel('Time','FontSize',14)

ylabel('SINR (dB)', 'FontSize', 14)

%ylabel('Average SINR','FontSize',14)

%ylim([0,80])

title('Average SINR Versus Power','FontSize',18)