EXPT NO : 8

NAME :Krishnaditya Kancharla

BATCH : D

ROLL NO : 2015120017

TITLE:OPEN SOURCE LTE/EPC NETWORK SIMULATOR USING NS-3

AIM:

1. Create 2 E-Node B and 4 UEs.
2. Attach 2 Ues to each E-NodeB.
3. Add mobility module to UEs.
4. Demonstrate Handover among E-NodeB.

SOFTWARE: ns-3

THEORY:

What is ns-3?

ns-3 is a discrete-event network simulator, targeted primarily for research and educational use. ns-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use.

The goal of the ns-3 project is to develop a preferred, open simulation environment for networking research: it should be aligned with the simulation needs of modern networking research and should encourage community contribution, peer review, and validation of the software.

CODE:

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2013 Centre Tecnologic de Telecomunicacions de Catalunya (CTTC)
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
```

```
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Manuel Requena <manuel.requena@cttc.es>
 */

#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"
#include "ns3/netanim-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("LenaX2HandoverMeasures");

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[])
{
  // LogLevel logLevel = (LogLevel)(LOG_PREFIX_ALL | LOG_LEVEL_ALL);

  // LogComponentEnable ("LteHelper", logLevel);
  // LogComponentEnable ("EpcHelper", logLevel);
  // LogComponentEnable ("EpcEnbApplication", logLevel);
  // LogComponentEnable ("EpcX2", logLevel);
  // LogComponentEnable ("EpcSgwPgwApplication", logLevel);

  // LogComponentEnable ("LteEnbRrc", logLevel);
  // LogComponentEnable ("LteEnbNetDevice", logLevel);
  // LogComponentEnable ("LteUeRrc", logLevel);
  // LogComponentEnable ("LteUeNetDevice", logLevel);
  // LogComponentEnable ("A2A4RsrqHandoverAlgorithm", logLevel);
  // LogComponentEnable ("A3RsrpHandoverAlgorithm", logLevel);

  uint16_t numberOfUes = 2;
  uint16_t numberOfEnbs = 2;
  uint16_t numBearersPerUe = 0;
  double distance = 500.0; // m
  double yForUe = 500.0;   // m
  double speed = 20;       // m/s
  double simTime = (double)(numberOfEnbs + 1) * distance / speed; // 1500 m / 20 m/s = 75 secs
  double enbTxPowerDbm = 46.0;

  // 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 (false));

  // 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));
```

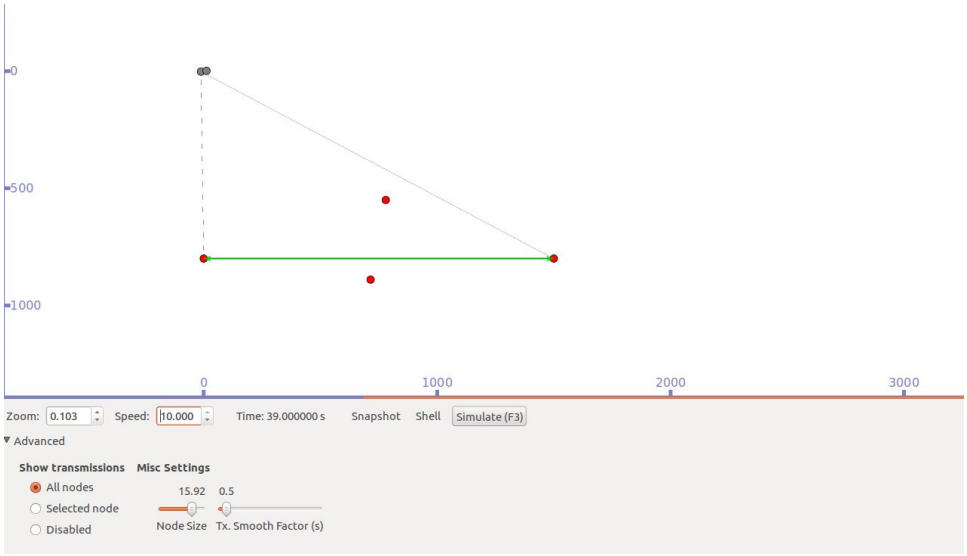
```
  lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset",
                                    UintegerValue (1));

  //  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 * (3*i), distance+300, 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+50, 0));
ueNodes.Get (0)->GetObject<ConstantVelocityMobilityModel> ()->SetVelocity (Vector (speed, 0, 0));
ueNodes.Get (1)->GetObject<MobilityModel> ()->SetPosition (Vector (1495, 500, 500));
ueNodes.Get (1)->GetObject<ConstantVelocityMobilityModel> ()->SetVelocity (Vector (-1*speed,0.5*speed,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));

// Attach all UEs to the first eNodeB
for (uint16_t i = 0; i < numberOfUes; i++)
  {
    lteHelper->Attach (ueLteDevs.Get (i), enbLteDevs.Get (i));
  }


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));
AnimationInterface anim ("prathameshhandover.xml");
  //anim.SetConstantPosition(enbLteDevs.Get (2),0,50);
  //anim.SetConstantPosition(enbLteDevs.Get (3),300,50);
  anim.UpdateNodeSize(2,10,10);
  anim.UpdateNodeSize(3,10,10);
  anim.UpdateNodeSize(4,10,10);

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

  // GtkConfigStore config;
  // config.ConfigureAttributes ();

  Simulator::Destroy ();
  return 0;

}
```

OUTPUT:



CONCLUSION: Thus handover in LTE was studied using NS3.