MWC Experiment 9

Shivani Bhat

BE EXTC A
2022200013

Aim:

To implement and simulate LTE Handoff on ns3 and NetAnim

Theory:

1. LTE Handoff (Handover)

Definition:

- Handoff (or handover) in LTE is the process of **transferring an active UE (User Equipment) connection from one eNodeB (base station) to another** without dropping the ongoing call or data session.
- LTE supports **hard handover (break-before-make)** due to its all-IP architecture. Soft handover (make-before-break) is not standard in LTE.

Types of Handover in LTE:

1. **Intra-LTE Handover** (same RAT – Radio Access Technology)
    - *Intra-eNodeB*: UE moves within the same eNodeB coverage (cell reselection).
    - *Inter-eNodeB*: UE moves between different eNodeBs (most common).
2. **Inter-RAT Handover**
    - LTE ↔ 3G (UMTS) or LTE ↔ 2G (GSM)
    - Required for legacy network support.

Key Metrics & Decisions:

- **Received Signal Strength (RSS)** / Reference Signal Received Power (RSRP)
- Signal-to-Interference-plus-Noise Ratio (SINR)
- Handover is triggered when the target cell's signal exceeds the source cell's signal by a **hysteresis margin** for a minimum **time-to-trigger (TTT)**.

Handover Procedure in LTE:

1. UE measures neighbor cell signals and reports to source eNodeB.
2. Source eNodeB decides to handover based on thresholds, hysteresis, and TTT.
3. Handover request sent to target eNodeB via X2 interface.

4. Target eNodeB prepares resources and responds.
5. UE is commanded to switch to the target eNodeB.
6. UE synchronizes and uplink begins at target; old connection is released.

Objectives:

- Maintain seamless connectivity.
- Minimize packet loss, latency, and dropped calls.
- Balance load between cells.

---

2. NS-3 (Network Simulator 3)

Definition:

- NS-3 is a **discrete-event network simulator** primarily used for research and educational purposes.
- Written in C++ with Python bindings.

Key Features:

- Supports **wired and wireless networks**, including LTE, WiFi, MANET, and more.
- Models realistic **protocol stacks**: PHY, MAC, TCP/IP layers.
- Allows simulation of **mobility, propagation models, fading, traffic patterns**, and QoS metrics.
- Modular design: users can write **custom applications, nodes, and protocols**.

Advantages of NS-3:

- Open-source and widely supported in academic research.
- Provides **trace-based outputs** for detailed analysis.
- Integrates with tools like NetAnim and PyViz for visualization.

Applications in LTE Research:

- LTE handoff performance evaluation.
- Resource allocation and scheduling simulations.
- Mobility and QoS testing under varying network conditions.

---

3. NetAnim (Network Animator)

Definition:

- NetAnim is a **visual animation tool for NS-3**, allowing dynamic visualization of nodes, packet flow, and mobility.
- Useful for **LTE handoff demonstrations**, network topology validation, and animation of wireless scenarios.

Features:

- Displays **node positions and mobility trajectories** in real-time.
- Animates **packet transmissions** (color-coded, directional).
- Shows **handover events**, drops, and successful connections visually.
- Supports **exporting animation sequences** for reports or presentations.

Workflow with NS-3:

1. Create NS-3 simulation script with required topology and mobility.
2. Enable NetAnim output by using AnimationInterface.
3. Run NS-3 simulation to generate .xml animation trace.
4. Open .xml in NetAnim to visualize events, nodes, and traffic.

Usefulness in LTE Handoff:

- Observe **UE movement across cells**.
- Verify **handover trigger conditions and timing**.
- Present **realistic handover animation** for reports or defense.

Code and Results:

```
#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)
```

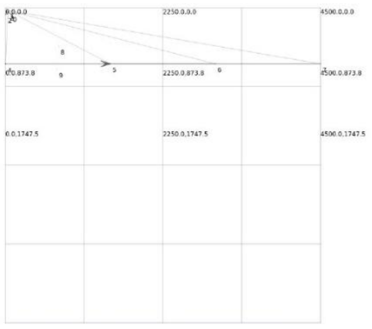
```
{
 Ptr<Node> ue = ueNodes.Get (u);
 // Set the default gateway for the UE
 Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ue->GetObject<Ipv4> ());
 ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);

 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)));
```

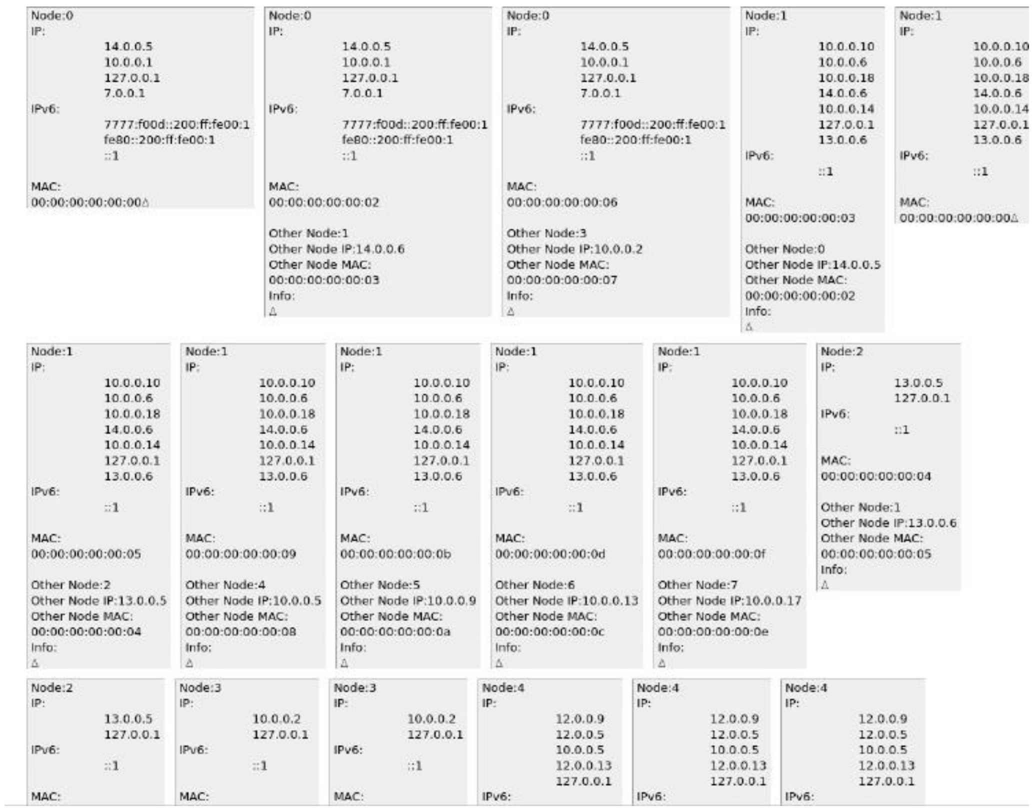
```
  Ptr<RadioBearerStatsCalculator> pdcpStats = lteHelper->GetPdcpStats ();
  pdcpStats->SetAttribute ("EpochDuration", TimeValue (Seconds (1.0)));

  // 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 ("LTEhandover.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;
```

}

When the number of enodeb and ues is changed from 2 to 4:



When the IP address is changed from 1.0.0.0 to 10.0.0.0:



```
Node:0
IP:
            14.0.0.5
            10.0.0.1
            127.0.0.1
            7.0.0.1
IPv6:
            7777:f00d::200:ff:fe00:1
            fe80::200:ff:fe00:1
            ::1

MAC:
00:00:00:00:00:00Δ
```

```
Node:0
IP:
            14.0.0.5
            10.0.0.1
            127.0.0.1
            7.0.0.1
IPv6:
            7777:f00d::200:ff:fe00:1
            fe80::200:ff:fe00:1
            ::1

MAC:
00:00:00:00:00:02

Other Node:1
Other Node IP:14.0.0.6
Other Node MAC:
00:00:00:00:00:03
Info:
Δ
```

```
Node:0
IP:
            14.0.0.5
            10.0.0.1
            127.0.0.1
            7.0.0.1
IPv6:
            7777:f00d::200:ff:fe00:1
            fe80::200:ff:fe00:1
            ::1

MAC:
00:00:00:00:00:06

Other Node:3
Other Node IP:10.0.0.2
Other Node MAC:
00:00:00:00:00:07
Info:
Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:03

Other Node:0
Other Node IP:14.0.0.5
Other Node MAC:
00:00:00:00:00:02
Info:
Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:00Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:05

Other Node:2
Other Node IP:13.0.0.5
Other Node MAC:
00:00:00:00:00:04
Info:
Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:09

Other Node:4
Other Node IP:10.0.0.5
Other Node MAC:
00:00:00:00:00:08
Info:
Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:0b

Other Node:5
Other Node IP:10.0.0.9
Other Node MAC:
00:00:00:00:00:0a
Info:
Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:0d

Other Node:6
Other Node IP:10.0.0.13
Other Node MAC:
00:00:00:00:00:0c
Info:
Δ
```

```
Node:1
IP:
            10.0.0.10
            10.0.0.6
            10.0.0.18
            14.0.0.6
            10.0.0.14
            127.0.0.1
            13.0.0.6
IPv6:
            ::1

MAC:
00:00:00:00:00:0f

Other Node:7
Other Node IP:10.0.0.17
Other Node MAC:
00:00:00:00:00:0e
Info:
Δ
```

```
Node:2
IP:
            13.0.0.5
            127.0.0.1
IPv6:
            ::1

MAC:
00:00:00:00:00:04

Other Node:1
Other Node IP:13.0.0.6
Other Node MAC:
00:00:00:00:00:05
Info:
Δ
```

```
Node:2
IP:
            13.0.0.5
            127.0.0.1
IPv6:
            ::1

MAC:
```

```
Node:3
IP:
            10.0.0.2
            127.0.0.1
IPv6:
            ::1

MAC:
```

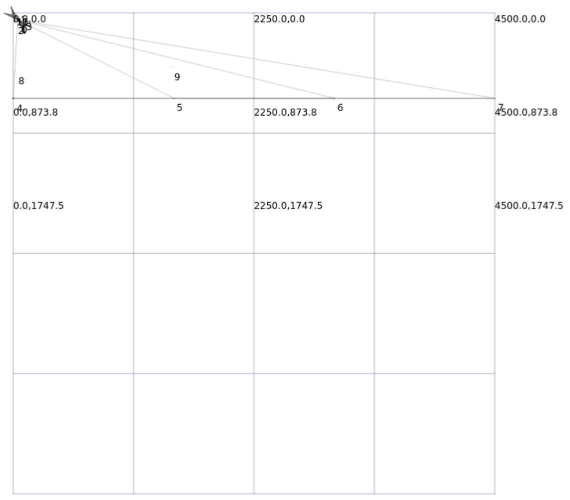
```
Node:3
IP:
            10.0.0.2
            127.0.0.1
IPv6:
            ::1

MAC:
```

```
Node:4
IP:
            12.0.0.9
            12.0.0.5
            10.0.0.5
            12.0.0.13
            127.0.0.1
IPv6:
```

```
Node:4
IP:
            12.0.0.9
            12.0.0.5
            10.0.0.5
            12.0.0.13
            127.0.0.1
IPv6:
```

```
Node:4
IP:
            12.0.0.9
            12.0.0.5
            10.0.0.5
            12.0.0.13
            127.0.0.1
IPv6:
```

When power is increased from 46.6dBm to 100dBm:



Conclusion:

1. **Topology:** 2 eNBs, 2 UEs moving linearly, 1 remote host.
2. **Mobility:** UEs move at **constant speed**; eNBs are fixed.
3. **Handover Trigger:** Signal quality (RSRQ) → automatic X2-based handover.
4. **Traffic:** UDP DL/UL traffic; optional QoS via dedicated bearers.
5. **Monitoring:** PHY, MAC, RLC, PDCP traces; console logs for handover events.
6. X2 interface enables smooth handovers.
7. Handover success depends on UE position, speed, and signal thresholds.
8. Hands-on insight into LTE mobility management & protocol behavior.