SIMON FRASER UNIVERSITY

ENSC 427

COMMUNICATIONS NETWORKS

Performance of VoLTE

Submitted by:

Himanshu GARG	301215259	hgarg@sfu.ca
Evgeny KUZNETSOV	301215280	ekuznets@sfu.ca
Bob JIU	301215502	cjiu@sfu.ca

www.hgarg.com/ensc427

Submitted to:
Dr. Ljiljana TRAJKOVIC
April 16, 2017

Abstract

Long Term Evolution (LTE) is the latest high speed mobile broadband technology that is gaining widespread attention due to its high data rates and improved Quality of Service (QoS). LTE is able to support high speed data services which uses packet switched network, but to support voice services these devices switch from LTE connection to 2G/3G mobile network which uses Circuit Switched Fallback Network (CSFB).The purpose of this project is to analyse performance of Voice over LTE networks. The goal is to implement LTE using NS-3, simulating the scenario and measure throughput, delay and handover performance

Acronyms

CS	Circuit Switch
LTE	Long Term Evolution
VoLTE	Voice over LTE
IMS	Internet Protocol Multimedia Subsystem
E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
SRVCC	Single Radio Voice Call Continuity
EPC	Evolved Packet Core
MME	Mobility Management Entity
SGW	Serving Gateway
VOIP	Voice Over IP
PDN	Packet Data Network
QoS	Quality of Service
PCRF	Policy Charging and Rules Function
SIM	Subscriber Identity Module
ISIM	IP Multimedia Services Identity Module
TE	Terminal Equipment
CSFB	Circuit Switched Fallback
MT	Mobile Termination
UDP	User Datagram Protocol
GBR	Guaranteed Bit Rate
RSRQ	Reference Signal Received Quality
eNB	eNodeB
UE	User Equipment (Any LTE capable device)

Contents

List of Figures

1 Introduction

In the past, the native voice call service was initiated by wired network through only Circuit Switched (CS) domain. It is a good network in terms of Quality of Service (QoS) but it wastes lots of resources for the during of a call. Packets Switched (PS) system divides data into packets and therefore save those resources. The second (2G) and third (3G) generation mobile networks provided voice communication service through Circuit Switched (CS) domain and data service through Packet Switched (PS) domain. Nowadays, Long Term Evolution (LTE) technology, which provides powerful capabilities to deal with growing data services over mobile application, spreads rapidly as an evolution of fourth generation network through PS domain. Compared with 2G/3G technologies, LTE supports a peak data rate up to 100 Mbps (downlink) and 50 Mbps (uplink).[1]

However, the nature of LTE, or the lack of CS domain, raises the issue on how to provide quality voice call and promise its continuity with 2G/3G legacy networks. The solution to this problem requires not only providing quality voice call service over a large number users but also a space for further improvement to adapt to the developing LTE network. [2] To solve these problems, the most two well-known methods have been applied to the industry, Circuit Switched Fallback (CSFB) and Voice over LTE (VoLTE).

Circuit Switched Fallback (CSFB) system is basically a handover method from LTE to 2G/3G as long as the LTE coverage is available in 2G/3G area. When the user equipment (UE) makes or receives a voice call, the eNodeB will redirect to 2G/3G networks. On the other hand, Internet Protocol Multimedia Subsystem (IMS) was invented to guarantee QoS, several IMS-based VoLTE technologies have been introduced over these years. The most satisfactory one of them is Single Radio Voice Call Continuity (SRVCC) as a 3GPP solution. The IMS must be available for the UE to initiate a voice call and for the implementation of the Application Server.[2]

This paper will firstly introduce the architecture of the networks and the procedure to establish the networks. Then it will present the simulation results of VoLTE using NS-3. Finally, this paper will use the results to analyze capability and efficiency of VoLTE network.

2 Architecture of LTE Network

Typical voice over LTE network consists of three main components: The User Equipment (UE), The Evolved UMTS Terrestrial Radio Access Network (E-UTRAN) and The Evolved Packet Core (EPC).[3] The endpoint can be connected to any IP based network in the diagram below represented as cloud. The Packet Data Network (PDN) acts as gateway connector to the rest of the world and thus makes it possible for two EU to communicate with each other.

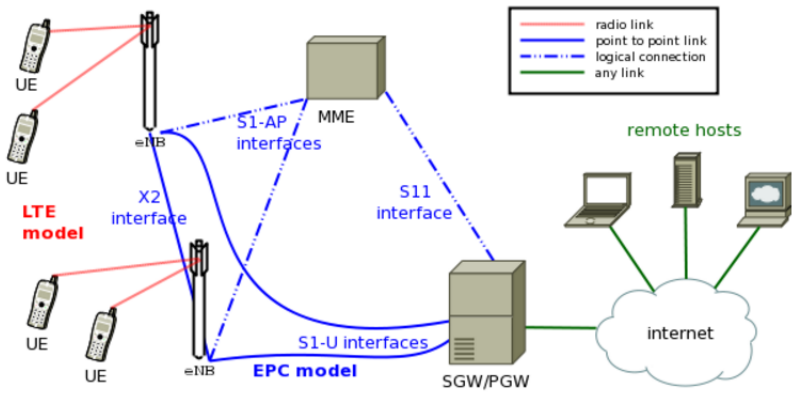


Figure 1: Architecture of an LTE network

2.1 User Equipment (UE)

User Equipment refers to any mobile terminal as communication device which is authorized to be used in the network. User Equipment can be smartphone, tablet, a laptop equipped with a mobile broadband adaptor, or other devices. The architecture of UE for LTE consists of three main components: Universal Integrated Circuit Card (UICC), Mobile Termination (MT) and Terminal Equipment (TE). The Mobile Termination (MT) deals with all the communication functions and the Terminal Equipment (TE) terminates the data streams.[3] Each UE must contain one UICC, known as SIM card for each LTE communication device. UICC stores user-specific information about the user's phone number, home network identity and security keys etc. UICC consists of four modules: Subscriber Identity Module (SIM), UMTS Subscriber Identity Module (USIM), CDMA Subscriber Identity Module (CSIM) or Re-Useable Identification Module (R-UIM) and IP Multimedia Services Identity Module (ISIM).

- Subscriber Identity Module (SIM): SIM identity information used by a GSM network.
- UMTS Subscriber Identity Module (USIM): USIM information used by a UMTS or LTE network.
- CDMA Subscriber Identity Module (CSIM) or Re-Useable Identification Module (R-UIM): identity information used by a CDMA network.
- IP Multimedia Services Identity Module (ISIM): ISIM identity information used by the IMS subsystem.[4]

2.2 Evolved UMTS Terrestrial Radio Access Network (E-UTRAN)

E-UTRAN consists of multiple of ENodeB or eNB stations. Each eNB can be seen as a hardware that allows a wireless LTE network to be accessible by UE, mobile phone for example. eNB serves the role of establishing connection from EU to Evolved Packet Core (EPC) when a user wants to transmit data or make a call. It is also responsible for routing of the calls, signaling, user data encryption and IP header compression. As EU moves across the space it may disconnect from one eNB and establish connection with another. The whole network that covers particular area on which LTE signaling and communication is possible is called E-UTRAN.

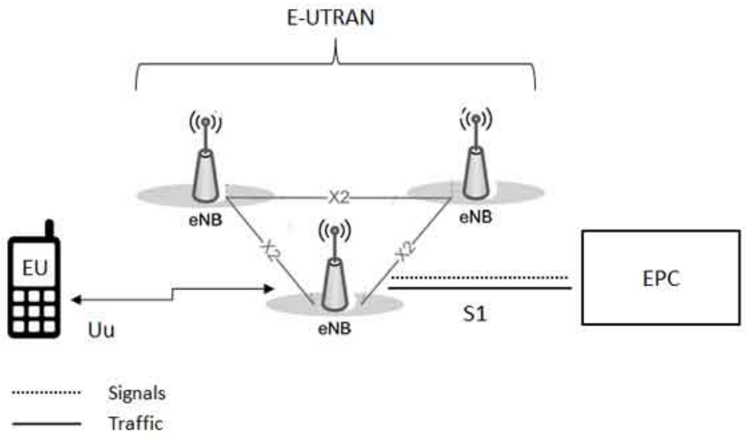


Figure 2: Evolved UMTS Terrestrial Radio Access Network

2.3 Evolved Packet Core (EPC)

EPC consist of four main components Mobility Management Entity (MME) , Serving Gateway (SGW), Packet Data Network Gateway (PGW) and Home Subscriber Server (HSS). EPC is responsible for communication and management of user traffic with external world networks. MME acts as control server for LTE, 2G, 3G networks by providing control functionality between user and communication channel. It initiates the signaling processes for instance, when a client wants to send data. HSS obtains the user information and register the data communication in the database. Then UE passes data packets to SGW which in its turn sends it to PGW and then can be transmitted to the rest of the world. The are also Policy Charging and Rules Function (PCRF) set in places for correct maintenance of network rules and client charging policies based on user‘s profile.[5]

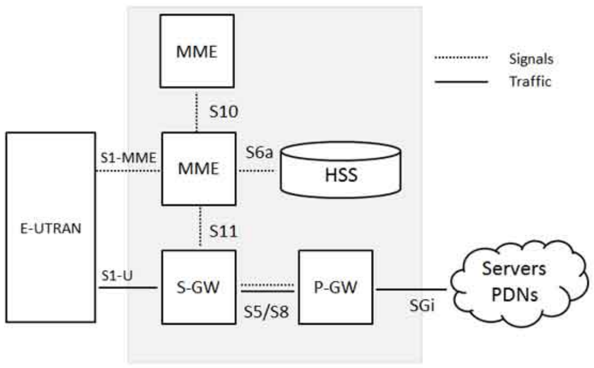


Figure 3: EPC Core

2.4 X2 Handover

X2 interface is a point to point interface between 2 eNodeBs. In the X2 model implemented in the simulator, the X2 interface is a point-to-point link between the two eNBs. A point-to-point device is created in both eNBs and the two point-to-point devices are attached to the point-to-point link.
For this simulation we use A2-A4-RSRQ handover algorithm as shown in the figure below. This algoritm utilizes the Reference Signal Received Quality (RSRQ) measurements acquired from events A2 and A4. These events are described below:

- Event A2 (serving cells RSRQ becomes worse than threshold) is leveraged to indicate that the UE is experiencing poor signal quality and may benefit from a handover.[6]
- Event A4 (neighbour cells RSRQ becomes better than threshold) is used to detect neighbouring cells and acquire their corresponding RSRQ from every attached UE, which are then stored internally by the algorithm. By default, the algorithm configures Event A4 with a very low threshold, so that the trigger criteria are always true. [6]

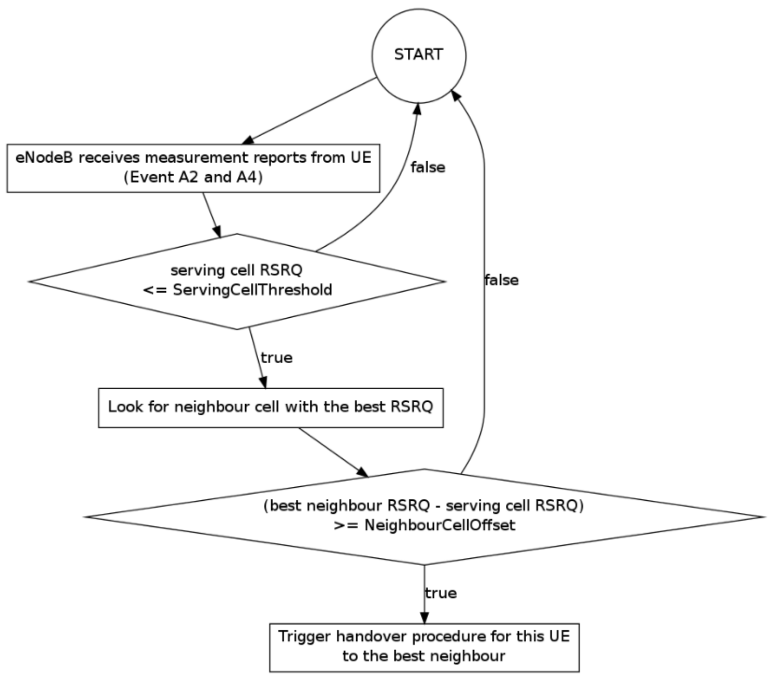

START
eNodeB receives measurement reports from UE (Event A2 and A4)
false
serving cell RSRQ <= ServingCellThreshold
false
true
Look for neighbour cell with the best RSRQ
(best neighbour RSRQ - serving cell RSRQ) >= NeighbourCellOffset
true
Trigger handover procedure for this UE to the best neighbour

Figure 4: A2-A4-RSRQ Handover Algorithm

3 Quality of Service (QoS)

To perform VoLTE application, network have to comply with two standardized LTE characteristics as defined in the 3GPP TS 23.203 standard "Policy and Charging Control Architecture".[7]

Name of service: IMS Signalling

- Class: 5 Resource: non-GBR Priority: 1 Packet delay budget: 100ms Packet Error Loss:10^{-6}

IMS signaling is responsible for providing an interface connection for the EU by controlling LTE radio access medium of the signal. It is important that this connection is established before any user data communication occurs thus it cannot tolerate long delays and packet losses.

Name of service: Conversational Voice

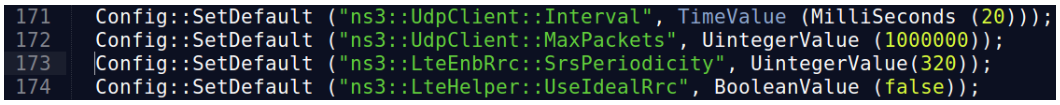
- Class: 1 Resource: GBR Priority: 2 budget: 100ms Packet Error Loss:10^{-2}

Conversational Voice is an actual application that is running on user phone and provide service of Voice Over IP (VOIP) communication. It is more tolerable to losses and small delay is acceptable why the bit rate should be guaranteed so provide stable end to end service for users.

4 Simulation

4.1 Simulation Setup

We start by setting up a simple scenario where 2 users are connected to a single eNodeB and measure the throughput of the network. We use a UDP echo module to simulate our traffic. There are 3 attributes which we can configure; maximum packets transmitted, the time interval between packets, and the packet size. The attributes used in our simulation are shown below



```
Config::SetDefault ("ns3::UdpClient::Interval", TimeValue (MilliSeconds (20)));
Config::SetDefault ("ns3::UdpClient::MaxPackets", UintegerValue (1000000));
Config::SetDefault ("ns3::LteEnbRrc::SrsPeriodicity", UintegerValue(320));
Config::SetDefault ("ns3::LteHelper::UseIdealRrc", BooleanValue (false));
```

Figure 5: Configuration for UDP traffic

In order to simulate traffic from the internet to eNodeB and to UEs remote host is set up. Each remote host is assigned an IP address and route. We also configure an X2 interface between each pair of eNodeBs in order to achieve a successful handover request. Flow Monitor module is used to monitor the simulation. The flow monitor is installed on all UEs and remote host. The throughput and delay metrics are calculated using the flow monitor. The simulation is run using Waf, a python based framework for compiling and running applications.

4.2 Multiple Users

We start by setting up a simple scenario where 2 users are connected to a single eNodeB and measure the throughput of the network. Then we increase the no. of users to 4,6,10,20,24,80 and 160 and see the effect of users on throughput and delay.

SRSPeriodicity sets the maximum number of allowable UEs to be attached to eNodeB. Also, to enable more than 30 users to be connected to a single eNodeB, SRSPeriodicity is set to its maximum value i.e 320. The simulation is run for 1s before it stops and is destroyed. A snapshot of the simulation with 20 users is shown below

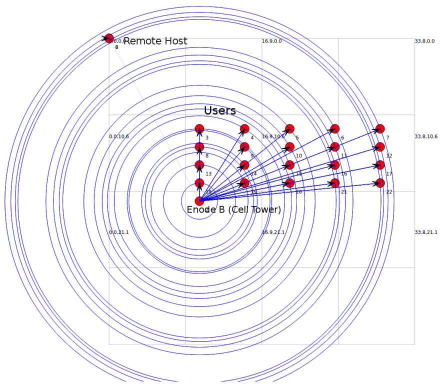


Figure 6: 1 eNodeB and 20 Users

It can be observed from the graphs below that throughput increases first with an increase in number of UE's and then starts to level out when we have too many UE's connecting to the same eNodeB. For the delay, we see that the downlink delay increases with an increase in number of UE's.

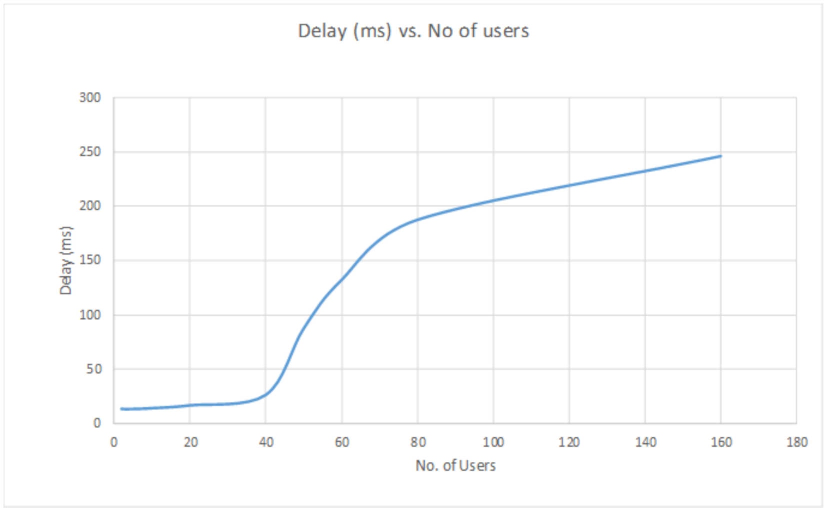


Figure 7: Delay vs. No of Users

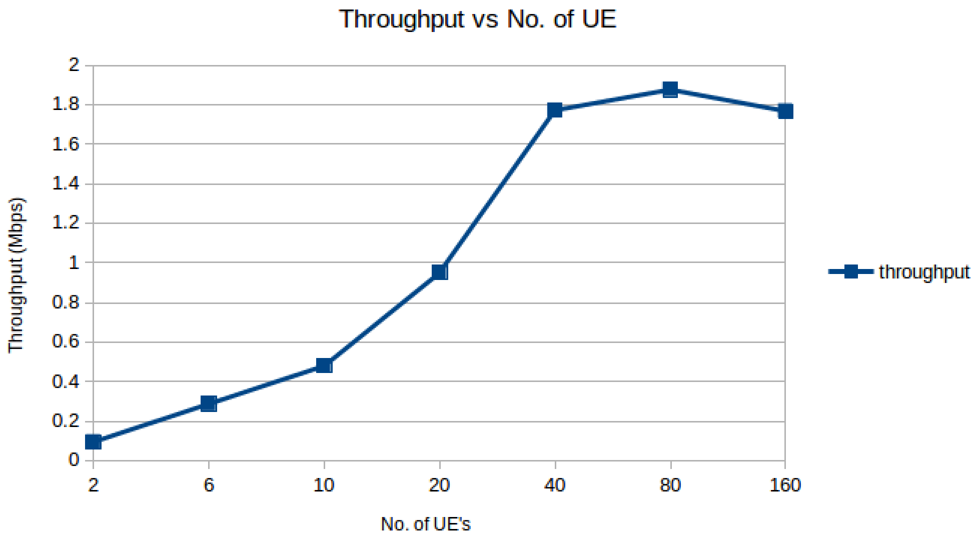


Figure 8: Throughput vs. No. of Users

4.3 X2 Handover scenario

In this scenario start by having 2 eNodeBs and 1 user moving through these eNodes at different speeds and we observe how X2 Handover affects the data transmitted between UE's and the Internet. We also compare a scenario where the UE moves away from a eNodeB and doesn't initiate a handover request. The graphs below show our results for both these scenarios. UE first moves towards the eNodeB and then away from eNodeB at 30m/s or 72 km/h.

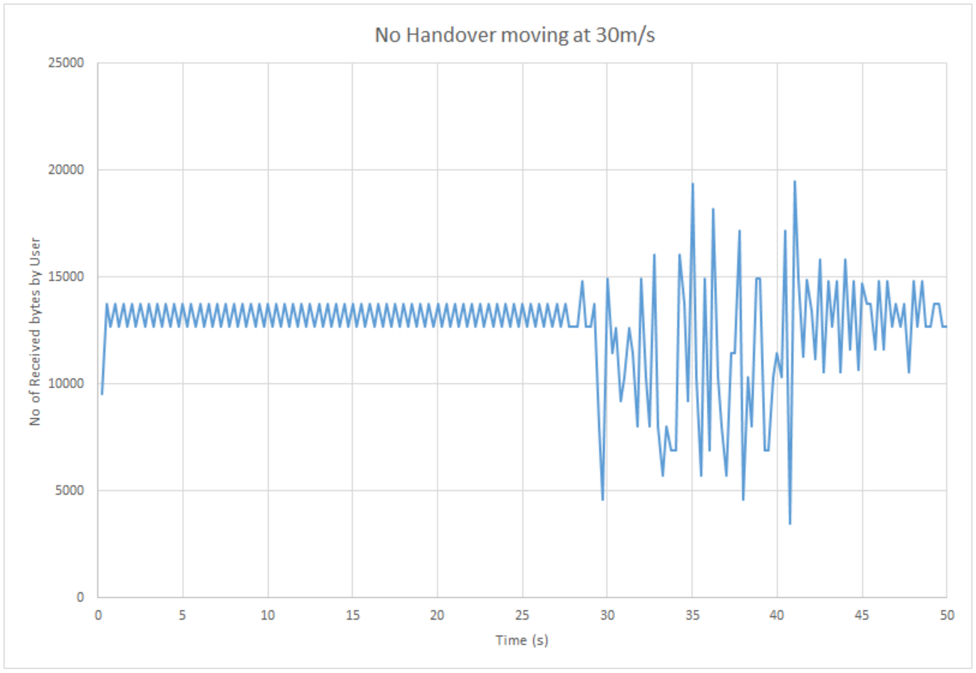


Figure 9: No Handover UE moving at 30m/s

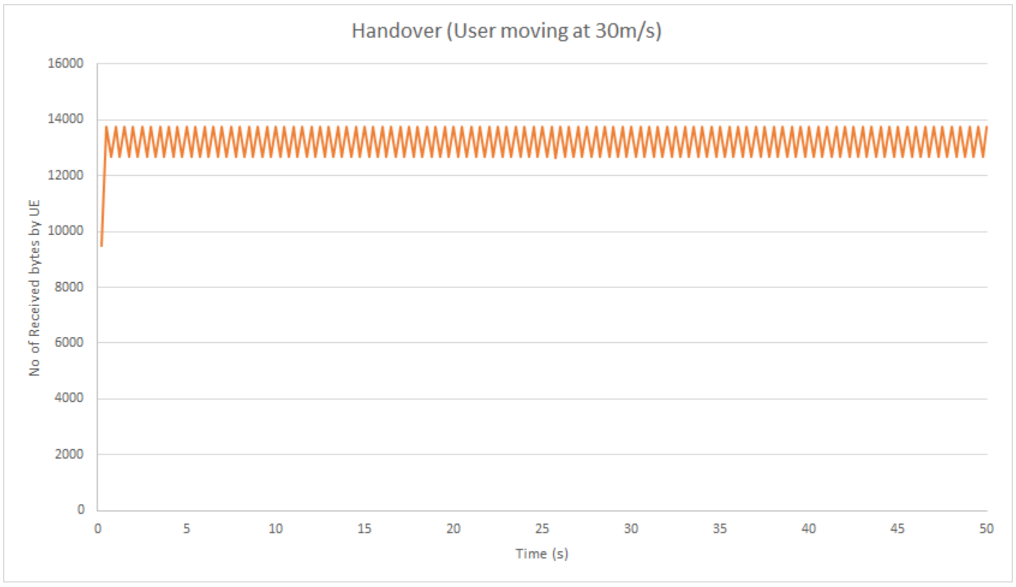


Figure 10: Handover UE moving at 30m/s

As we can see from the graphs, in case of handover we have a stable connection throughout 50 seconds, because as the connection starts to worsen, the A2-A4 RSRQ handover decides to switch to the nearest eNodeB to maintain a stable connection, but in case of no handover, the connection starts to worsen as we move away from the eNode. The scenario we used for handover is shown in the figure below

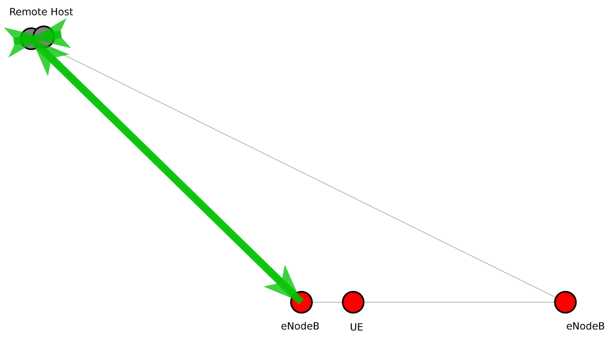


Figure 11: Connected to eNodeB 1

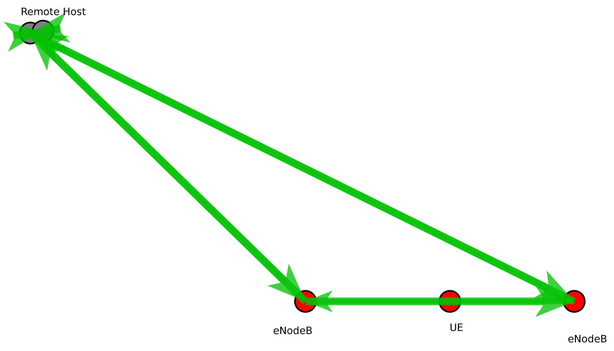


Figure 12: Handover in process

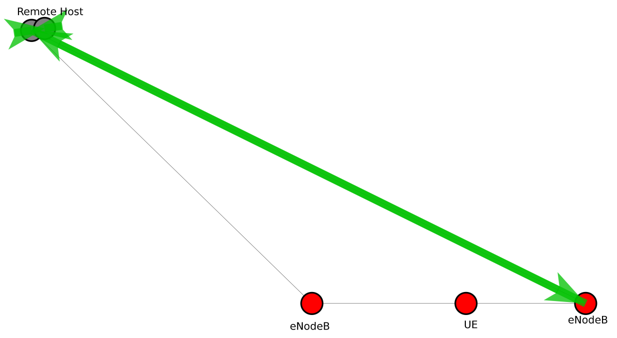


Figure 13: Successful Handover to eNodeB 2

5 Discussion and Conclusion

In this project we simulated 3 different scenarios and discussed the performance of VoLTE. The factors we changed were the No of UE's to measure the performance of eNodeB's and impact of X2 handover. From these scenarios we observe that total throughput increases with an increase in the number of UE's and then starts to level out which is what we expect. Also, the delay has a direct relation with the number of UE's. Also, we see that handover helps to maintain a stable connection with minimum packet loss while in case of no handover the connection worsens as the user moves away from the eNodeB. Conclusively, VoLTE is a promising solution to current CSFB technology. With the companies starting to adopt VoLTE in their networks it will soon become widely adopted.

Some difficulties we had regarding this project was learning to work with NS-3. However, with a lot of well structured documentation it took us a little time to get used to NS-3 but it allowed us to gain much more granular data and perform simulations with better control over the parameters. A lot of unanswered google discussions on NS3 LTE topic was also one of the challenges we had to overcome.

For our future work we would like to study the effects of different scheduling algorithms on throughput and delay. We would also like to set up more realisitic simulations for instance, like walls, apartment buildings and compare them with traditional GSM networks.

References

[1] A. A. El Arby and O. Thiare, “Handling voice in lte,” in *Computer, Communications, and Control Technology (I4CT), 2015 International Conference on.* IEEE, 2015, pp. 7–10.

[2] V. Paisal, “Seamless voice over lte,” in *Internet Multimedia Services Architecture and Application (IMSAA), 2010 IEEE 4th International Conference on.* IEEE, 2010, pp. 1–5.

[3] T. Point. (2017) Lte network architecture. [Online]. Available: https://www.tutorialspoint.com/lte/lte_network_architecture.htm

[4] I. V. Architecture. (2017) 3glteinfo. [Online]. Available: http://www.3glteinfo.com/ims-volte-architecture/#comment-47813

[5] G. Association. (2014) Volte service description and implementation guidelines. [Online]. Available: http://www.gsma.com/network2020/wp-content/uploads/2014/05/FCM.01-v1.1.pdf

[6] O. Source. (2017) ns-3 a discrete-event network simulator. [Online]. Available: https://www.nsnam.org/docs/models/html/lte-design.html

[7] M. Tabany and C. Guy, “An end-to-end qos performance evaluation of volte in 4g e-utran-based wireless networks,” in *the 10th International Conference on Wireless and Mobile Communications*, 2014.

6 Appendix

```
#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/flow-monitor-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"
#include "ns3/gnuplot.h"
#include <math.h>
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("lte_handover_scenario_automatic_handover
   ");

void
NotifyConnectionEstablishedUe (std::string context,
                               uint64_t imsi,
                               uint16_t cellid,
                               uint16_t rnti)
{
  std::cout << Simulator::Now ().GetSeconds () << " " << 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 << Simulator::Now ().GetSeconds () << " " << 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 << Simulator::Now ().GetSeconds () << "␣" << 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 << Simulator::Now ().GetSeconds () << "␣" << 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 << Simulator::Now ().GetSeconds () << "␣" << 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 << Simulator::Now ().GetSeconds () << " " << context
            << " eNB CellId " << cellid
            << ": completed handover of UE with IMSI " << imsi
            << " RNTI " << rnti
            << std::endl;
}

void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<
   FlowMonitor> flowMon,Gnuplot2dDataset DataSet)
  {
    double localThrou=0;
    std::map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon->
       GetFlowStats();
    Ptr<Ipv4FlowClassifier> classing = DynamicCast<
       Ipv4FlowClassifier> (fmhelper->GetClassifier());
    for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
       stats = flowStats.begin (); stats != flowStats.end (); ++
       stats)
    {
      Ipv4FlowClassifier::FiveTuple fiveTuple = classing->FindFlow
          (stats->first);
       //if (fiveTuple.sourceAddress == Ipv4Address("7.0.0.2") ||
           fiveTuple.sourceAddress == Ipv4Address("7.0.0.1") /*||
          fiveTuple.sourceAddress == Ipv4Address("7.0.0.2") ||
          fiveTuple.sourceAddress == Ipv4Address("7.0.0.3")*/)
      {

      std::cout<<"Flow ID      : " << stats->first <<" ; "<<
         fiveTuple.sourceAddress <<" -----> "<<fiveTuple.
         destinationAddress<<std::endl;
      std::cout<<"Tx Packets : " << stats->second.txPackets<<std::
         endl;
      std::cout<<"Rx Packets : " << stats->second.rxPackets<<std::
         endl;
      std::cout<<"Duration     : "<<(stats->second.timeLastRxPacket
         .GetSeconds()-stats->second.timeFirstTxPacket.GetSeconds
         ())<<std::endl;
      std::cout<<"Last Received Packet  : "<< stats->second.
```

```
            timeLastRxPacket.GetSeconds()<<" Seconds"<<std::endl;
        std::cout<<"Throughput: " << stats->second.rxBytes * 8.0 / (
            stats->second.timeLastRxPacket.GetSeconds()-stats->second
            .timeFirstTxPacket.GetSeconds())/1024  << " Kbps"<<std::
            endl;
        std::cout<< "Mean{Delay}: " << (stats->second.delaySum.
            GetSeconds()/stats->second.rxPackets) << "\n";
        std::cout<< "Mean{Jitter}: " << (stats->second.jitterSum.
            GetSeconds()/(stats->second.rxPackets)) << "\n";
        std::cout<< "Total{Delay}: " << (stats->second.delaySum.
            GetSeconds()) << "\n";
        std::cout<< "Total{Jitter}: " << (stats->second.jitterSum.
            GetSeconds()) << "\n";
        std::cout<< "Lost Packets: " << (stats->second.lostPackets)
           << "\n";
        std::cout<< "Dropped Packets: " << (stats->second.
            packetsDropped.size()) << "\n";
        localThrou=(stats->second.rxBytes * 8.0 / (stats->second.
            timeLastRxPacket.GetSeconds()-stats->second.
            timeFirstTxPacket.GetSeconds())/1024);
        // updata gnuplot data
                DataSet.Add((double)Simulator::Now().GetSeconds(),(
                    double) localThrou);
        std::cout<<"
           ------------------------------------------------------"<<std
            ::endl;
        }
     }
        Simulator::Schedule(Seconds(0.02),&ThroughputMonitor,
            fmhelper, flowMon,DataSet);
        flowMon->SerializeToXmlFile ("ThroughputMonitor.xml", true,
            true);

  }

int
main (int argc, char *argv[])
{
  LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
  CommandLine cmd;
  uint16_t numberOfUes = 1;
  uint16_t numberOfEnbs = 1;
  double interPacketInterval = 20;
  //uint16_t numBearersPerUe = 0;
```

```
double distance = 500.0; // m
double yForUe = 500.0;    // m
double speed = 40;        // m/s
double simTime = 1.0;//(double)(numberOfEnbs + 1) * distance /
   speed; // 1500 m / 20 m/s = 75 secs

Config::SetDefault ("ns3::UdpClient::Interval", TimeValue (
   MilliSeconds (20)));
Config::SetDefault ("ns3::UdpClient::MaxPackets", UintegerValue
   (1000000));
Config::SetDefault ("ns3::LteHelper::UseIdealRrc", BooleanValue
   (false));
Config::SetDefault ("ns3::LteEnbRrc::SrsPeriodicity",
   UintegerValue(320));


// Command line arguments
cmd.AddValue ("numberOfUes", "Number␣of␣UEs", numberOfUes);
cmd.AddValue ("numberOfEnbs", "Number␣of␣eNodeBs", numberOfEnbs)
   ;
cmd.AddValue ("simTime", "Total␣duration␣of␣the␣simulation␣(in␣
   seconds)", simTime);
cmd.AddValue("interPacketInterval", "Inter␣packet␣interval␣[ms])
   ", interPacketInterval);
cmd.Parse (argc, argv);

Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
Ptr<PointToPointEpcHelper> epcHelper = CreateObject<
   PointToPointEpcHelper> ();
lteHelper->SetEpcHelper (epcHelper);
lteHelper->SetSchedulerType ("ns3::RrFfMacScheduler");

lteHelper->SetHandoverAlgorithmType ("ns3::NoOpHandoverAlgorithm
   "); // disable automatic handover

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);

lteHelper->SetHandoverAlgorithmType ("ns3::
   A2A4RsrqHandoverAlgorithm");
lteHelper->SetHandoverAlgorithmAttribute ("ServingCellThreshold"
   ,UintegerValue (30));
lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset",
   UintegerValue (1));

NodeContainer ueNodes;
NodeContainer enbNodes;
enbNodes.Create (numberOfEnbs);
ueNodes.Create (numberOfUes);

// Create position allocators and mobility models
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
NetDeviceContainer enbLteDevs = lteHelper->InstallEnbDevice (
   enbNodes);
NetDeviceContainer ueLteDevs = lteHelper->InstallUeDevice (
   ueNodes);

// Install the IP stack on the UEs
internet.Install (ueNodes);
Ipv4InterfaceContainer ueIpIface;
ueIpIface = 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 (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 (0.10));

ApplicationContainer appContainer;
```

```
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);

        ++dlPort;
        ++ulPort;

        std::cout<<ipv4RoutingHelper.GetStaticRouting (ue->
           GetObject<Ipv4> ());
        ApplicationContainer clientApps;
        ApplicationContainer serverApps;



        NS_LOG_LOGIC ("installing UDP DL app for UE " << u);
        UdpClientHelper dlClientHelper (ueIpIface.GetAddress (u)
           , dlPort);

        clientApps.Add (dlClientHelper.Install (remoteHost));
        dlClientHelper.SetAttribute("Interval",TimeValue (
           MilliSeconds(interPacketInterval)));
        dlClientHelper.SetAttribute("MaxPackets", UintegerValue
           (1000000));
        //dlClientHelper.SetAttribute("PacketSize",
           UintegerValue (160));
        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))
           ;
        ulClientHelper.SetAttribute("Interval",TimeValue (
           MilliSeconds(interPacketInterval)));
        ulClientHelper.SetAttribute("MaxPackets", UintegerValue
           (1000000));
        //ulClientHelper.SetAttribute("PacketSize",
           UintegerValue (160));


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

        Time startTime = Seconds (startTimeSeconds->GetValue ())
           ;
        serverApps.Start (startTime);
        clientApps.Start (startTime);
  }


// Add X2 inteface
lteHelper->AddX2Interface (enbNodes);
lteHelper->EnablePhyTraces ();
lteHelper->EnableMacTraces ();
lteHelper->EnableRlcTraces ();
lteHelper->EnablePdcpTraces ();


// 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));
AnimationInterface anim ("lte_animation_test.xml");
anim.SetMaxPktsPerTraceFile (100000000000);
anim.SetMobilityPollInterval(Seconds(1));
//Gnuplot parameters

  std::string fileNameWithNoExtension = "FlowVSThroughput_";
  std::string graphicsFileName        = fileNameWithNoExtension
     + ".png";
  std::string plotFileName            = fileNameWithNoExtension
     + ".plt";
  std::string plotTitle               = "Flow vs Throughput";
  std::string dataTitle               = "Throughput";

  // Instantiate the plot and set its title.
  Gnuplot gnuplot (graphicsFileName);
  gnuplot.SetTitle (plotTitle);

  // Make the graphics file, which the plot file will be when it
  // is used with Gnuplot, be a PNG file.
  gnuplot.SetTerminal ("png");

  // Set the labels for each axis.
  gnuplot.SetLegend ("Flow", "Throughput");
  Gnuplot2dDataset dataset;
  dataset.SetTitle (dataTitle);
  dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);

//flowMonitor declaration
FlowMonitorHelper fmHelper;
Ptr<FlowMonitor> allMon = fmHelper.InstallAll();
allMon->CheckForLostPackets ();
// call the flow monitor function
ThroughputMonitor(&fmHelper, allMon, dataset);

Simulator::Run ();

gnuplot.AddDataset (dataset);
std::ofstream plotFile (plotFileName.c_str());
gnuplot.GenerateOutput (plotFile);
plotFile.close ();

Simulator::Destroy ();
return 0;

}
```