LTE Network Simulation in NS-3

Márcio Tamide Dias dos Anjos, Matěj Hykš ,
Jorge Carrascoso Agudo, Cheikh Cisse

The objective of the project is to simulate an LTE network using the Network Simulator 3 (NS-3) framework.

LTE Network Simulation in NS-3 – Assignment 08

1 Objective

The objective of the project is to gain hands-on experience in simulating an LTE network using the Network Simulator 3 (NS-3) framework. Students will create a comprehensive network scenario, configure LTE parameters, and analyze network performance.

2 Task

2.1 Scenario Definition

Define a network scenario including, but not limited to, the following elements:

- At least two eNodeBs.
- A minimum of ten User Equipment (UE) devices.
- At least one remote host (server) on the Internet.
- Appropriate IP addressing and routing configuration.

2.2 Mobility

Study the position allocators and mobility models in NS-3. Assign appropriate mobility models for the eNodeBs and the UEs according to the following requirements:

- Geographically distribute eNodeBs and UEs to create a realistic network topology. Use 2 different position allocators for the eNodeBs and the UEs.
- Five UEs are stationary, five UEs mimic the movement of pedestrians.

2.3 Path loss model

Study the available propagation loss models in NS-3. Choose a suitable propagation loss model to simulate the scenario where the UEs are in a building.

2.4 Traffic Generation

Study the available applications in NS-3. Choose the suitable applications to simulate the scenario where the UEs are browsing the web.

2.5 Simulation Run

- Execute the simulation with different sets of parameters and collect data on network performance.
- Run the simulation for a sufficient duration to observe network behavior.
- Capture and analyze key metrics, such as throughput, latency, and packet loss.

2.6 Analysis and Report

Prepare a detailed report that includes the following:

- Description of your network scenario.
- Configuration settings, including LTE and application parameters.
- Results and analysis of network performance.
- Any issues encountered during the simulation and how they were resolved.
- Recommendations for improving network performance (if applicable).

3 Submission

- Submit your report in a digital format, along with any necessary simulation scripts and configuration files.
- Prepare a presentation (max. 10 min) highlighting the key ideas and results of your project.

1 Scenario Definition

First thing first, let's talk about the topology of the assignment. Topology will consist of eNodeBs, User Equipment's, Packet Data Network Gateway and Remote Host. Let's begin with creation of eNodeBs.

1.1 Creation of eNodeBs

The code creates a specified number of eNodeBs (2) within the simulation, establishing the base stations that facilitate wireless communication with UEs.

Listing 1.1: Creation of eNodeBs

```
uint16_t numNodePairs = 2;
NodeContainer enbNodes;
enbNodes.Create(numNodePairs);
```

Default number of *eNodeBs* is **2**, but we can change that with command line argument for example:

Listing 1.2: Command line argument for number of eNodeBs

```
./ns3 run "project␣--numNodePairs=3"
```

In this simulation, the eNodeBs are configured with a constant position mobility model, implying that they are stationary throughout the simulation (Static Mobility Model). But their position is selected randomly in range from 0 to 500 for both **X** and **Y** axis. Part of the code which takes care of it is:

Listing 1.3: Creation of list of random position allocator

```
for (int i = 0; i < 20; i++) {
    positionAlloc->Add(Vector(getRandomNumber(0, 500),
                              getRandomNumber(0, 500), 0.0));
}
```

where *getRandomNumber(0, 500)* is simple function to create number in said range.

1.2 Creation of UEs

Two sets of UEs are instantiated: stationary and walking. Stationary UEs are placed at fixed positions, while walking UEs employ a random walk 2D mobility model, simulating mobile devices that move within defined bounds.

Listing 1.4: Creation of static UEs

```
NodeContainer stationaryUeNodes;
stationaryUeNodes.Add(ueNodes.Get(0));
stationaryUeNodes.Add(ueNodes.Get(1));
stationaryUeNodes.Add(ueNodes.Get(2));
stationaryUeNodes.Add(ueNodes.Get(3));
stationaryUeNodes.Add(ueNodes.Get(4));
```

Listing 1.5: Creation of UEs with Random Walk mobility model

```
NodeContainer walkingUeNodes;
walkingUeNodes.Add(ueNodes.Get(5));
walkingUeNodes.Add(ueNodes.Get(6));
walkingUeNodes.Add(ueNodes.Get(7));
walkingUeNodes.Add(ueNodes.Get(8));
walkingUeNodes.Add(ueNodes.Get(9));
```

UEs interact with eNodeBs to access LTE services, transmitting and receiving data as required thus we must assignt UEs to eNodeBs, but first let's create helper which will allow us to do so:

Listing 1.6: Assigment of UEs to eNodeBs

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper>();
Ptr<PointToPointEpcHelper> epcHelper =
    CreateObject<PointToPointEpcHelper>();
lteHelper->SetEpcHelper(epcHelper);

NetDeviceContainer enbLteDevs =
    lteHelper->InstallEnbDevice(enbNodes);
NetDeviceContainer ueLteDevs =
    lteHelper->InstallUeDevice(ueNodes);
```

1.3 PGW (Packet Data Network Gateway)

The Packet Data Network Gateway acts as a crucial interface between the LTE network and external packet data networks, like the internet:

It handles data routing, serves as a point of connection to external networks, and manages IP addresses for UEs Similar to eNodeBs, in the simulation code, the PGW is configured with a constant position mobility model. Let's create one with the code:

Listing 1.7: Creation of Packet Gateway

```
MobilityHelper pgwMobility;
pgwMobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
pgwMobility.SetPositionAllocator(positionAlloc);
pgwMobility.Install(pgw);
```

It allows UEs within the simulation to communicate with external hosts (simulated by the Remote Host node) and access resources beyond the LTE network boundaries. The PGW serves as a key entity responsible for managing IP addresses assigned to UEs.

1.4 Remote Host

The Remote Host is a simulated node that serves a specific purpose within the LTE simulation:

External Interaction Representation: It acts as a representative node from an external network, such as the internet or another separate network entity that communicates with the LTE simulation.

Simulating External Communications: The Remote Host likely simulates interactions between the LTE network and entities external to it.

Similar to other stationary components in the simulation, such as eNodeBs and the PGW, the Remote Host is configured to remain static using a constant position mobility model.

Listing 1.8: Creation of Remote host

```
NodeContainer remoteHostContainer;
remoteHostContainer.Create(1);
Ptr<Node> remoteHost = remoteHostContainer.Get(0);
InternetStackHelper internet;
internet.Install(remoteHostContainer);

MobilityHelper remoteHostMobility;
remoteHostMobility.SetMobilityModel(
        "ns3::ConstantPositionMobilityModel");
remoteHostMobility.SetPositionAllocator(positionAlloc);
remoteHostMobility.Install(remoteHost);
```

2 Mobility

2.1 Constant Position Mobility Model

The constant position mobility model is employed for eNodeBs, the PGW, the Remote Host, and certain UEs within the simulation, Nodes using this model remain fixed at specific coordinates throughout the simulation (Static Nature).

Listing 2.1: Stationary Nodes

```
MobilityHelper stationaryMobility;
stationaryMobility.SetMobilityModel(
    "ns3::ConstantPositionMobilityModel");
stationaryMobility.SetPositionAllocator(positionAlloc);
stationaryMobility.Install(stationaryUeNodes);
```

2.2 Random Walk 2D Mobility Model

This model is specifically utilized for some UEs in the simulation.

Listing 2.2: Walking Mobility

```
MobilityHelper walkingMobility;
walkingMobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
    "Bounds",
    RectangleValue(Rectangle(-500, 500, -500, 500)));
walkingMobility.SetPositionAllocator(positionAlloc);
walkingMobility.Install(walkingUeNodes);
```

This scenario helps evaluate the LTE network's performance in handling mobile devices' movements.

3 Path loss model

While trying multiple path loss model, we had no success. Main one which we tried was **BuildingsPropagationLossModel**, but even simple line of the code:

```
CreateObject<BuildingsPropagationLossModel>();
```

caused problems and we weren't able to run the code at all.

4 Traffic generation

The server generates main and embedded objects of varying sizes, triggered internally or by client requests. This is indicated by the functions **MainObjectGenerated** and **EmbeddedObjectGenerated**.

The server traces the generation of main and embedded objects, connecting them to respective callback functions (**MainObjectGenerated** and **EmbeddedObject-Generated**) through **httpServer->TraceConnectWithoutContext()** calls.

In our project we created HTTP server using example **three-gpp-http-example**. Let's start with creation of the **ThreeGppHttpServerHelper** class which will act as web server on our remote host.

Listing 4.1: Creation of web server on remote host

```
ThreeGppHttpServerHelper serverHelper(remoteHostAddr);
ApplicationContainer serverApps = serverHelper.Install(
    remoteHostContainer.Get(0));
Ptr<ThreeGppHttpServer> httpServer = serverApps.Get(0)
    ->GetObject<ThreeGppHttpServer>();

httpServer->TraceConnectWithoutContext("ConnectionEstablished",
    MakeCallback(&ServerConnectionEstablished));
httpServer->TraceConnectWithoutContext("MainObject",
    MakeCallback(&MainObjectGenerated));
httpServer->TraceConnectWithoutContext("EmbeddedObject",
    MakeCallback(&EmbeddedObjectGenerated));
httpServer->TraceConnectWithoutContext("Tx",
    MakeCallback(&ServerTx));
```

and continue with creation of **ThreeGppHttpClientHelper** class which will be installed on each UEs in **for** cycle and act as web client.

The HTTP server-client communication is orchestrated using ThreeGppHttpServer-Helper and ThreeGppHttpClientHelper classes.

Tracing established connections and objects enables detailed analysis of communication events, object sizes, successful receptions, and potential failures, providing insights into network behavior and performance.

This setup mimics real-world HTTP communication, allowing the generation, transmission, and reception of objects between the server and multiple clients within the LTE network.

Listing 4.2: Creation of web clients on UEs

```
ThreeGppHttpClientHelper clientHelper(remoteHostAddr);
```

```
ApplicationContainer clientApps;
for (uint32_t u = 0; u < ueNodes.GetN(); ++u)
{
    clientApps = clientHelper.Install(ueNodes.Get(u));
    Ptr<ThreeGppHttpClient> httpClient = clientApps.Get(0)
        ->GetObject<ThreeGppHttpClient>();

    httpClient->TraceConnectWithoutContext("RxMainObject",
        MakeCallback(&ClientMainObjectReceived));
    httpClient->TraceConnectWithoutContext("RxEmbeddedObject",
        MakeCallback(&ClientEmbeddedObjectReceived));
    httpClient->TraceConnectWithoutContext("Rx",
        MakeCallback(&ClientRx));
    }
```

5 Configuration

The LTE configuration in the simulation involves creating LTE devices and establishing connections between User Equipments (UEs) and Evolved Node Base Stations (eNodeBs), including the activation of default Evolved Packet System (EPS) bearers.
The code involves the creation of the eNodeBs, then LTE devices are installed for eNodeBs and UEs. Next step includes attaching UEs to specific eNodeBs and activating default EPS bearers.
Default EPS bearers facilitate initial communication and data transfer between UEs and the LTE network immediately upon attachment, activating default EPS bearers ensures that UEs have an established path for immediate communication within the LTE network.

5.1 LTE Configuration Settings

5.1.1 Parameters set

- Number of eNodeBs and UEs: Defined using numNodePairs and numberOfUes variables, respectively.
- Simulation Time: Configured via the simTime variable, determining the total duration of the simulation.
- Web Socket Ports: Defined for server and client via webSocketServer and webSocketClient variables, respectively.
- Inter-Packet Interval: Set as interPacketInterval to control the timing between packets.

5.1.2 LTE Device Installation

Installation of eNodeBs and UEs using InstallEnbDevice / InstallUeDevice functions, respectively.
The UE Attachment is executed through iteration, attaching UEs to specific eNodeBs based on conditional logic.
The activation of EPS Bearer is done automatically when we're attaching the UEs.

5.2 Application Parameters

Here we have the settings of the HttpServer and HttpClient for the application.

5.2.1 HTTP Server and Client Setup

The script utilizes ThreeGppHttpServerHelper and ThreeGppHttpClientHelper to create instances of HTTP server and client applications. These simulate HTTP traffic, allowing communication between the server (remoteHost) and multiple clients (UEs).

5.2.2 HTTP Object Sizes

These parameters set the size of main and embedded objects within the HTTP responses. They determine the payload size and content structure in simulated HTTP transactions.

Listing 5.1: Object size

```
httpVariables->SetMainObjectSizeMean(102400);
httpVariables->SetMainObjectSizeStdDev(40960);
```

Mean size of the main object (100kB), standard deviation for the main object size (40kB).

5.2.3 Trace Connections

HTTP Server Trace Connections

The code establishes trace connections to capture and monitor various events within the simulation, especially related to HTTP communication.

HTTP Client Trace Connections

Similar trace connections for the HTTP clients monitor events like the reception of main objects, embedded objects, and packets.

These traces are instrumental in capturing and analyzing events that occur during HTTP communication, providing insights into the behavior and performance of the simulated network and applications.

6 Simulation Execution

Here, the captures of the output of the code, first the average delay of the communication, and then the data rate of the communication.

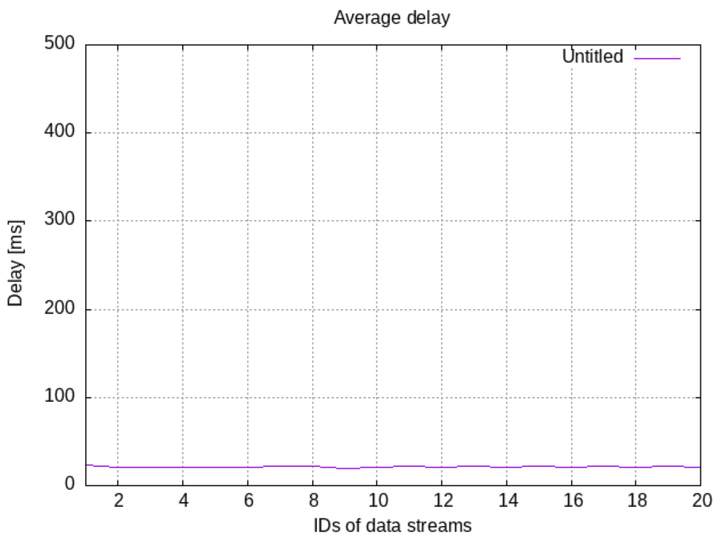

Fig. 6.1: Average delay

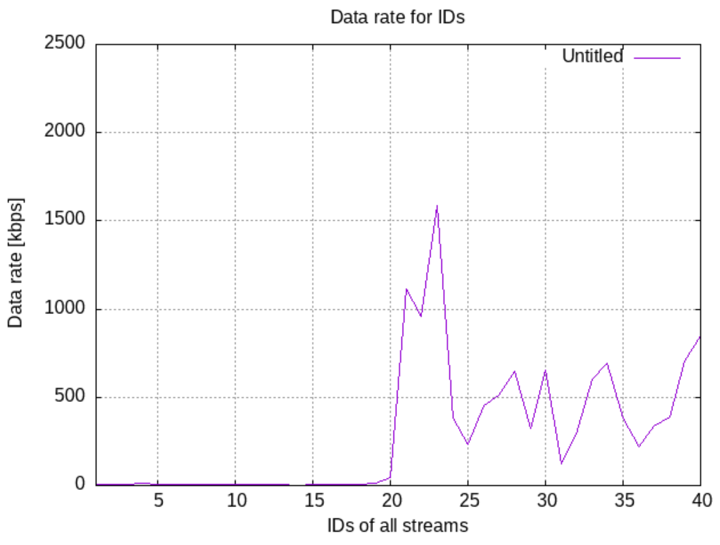

Fig. 6.2: Data rate