Trabalho Laboratorial 5

Comunicações Móveis

Network Simulation - LTE

Diogo Remião & Miguel Pinheiro

Maio 2021

Faculdade de Engenharia da Universidade do Porto
TEC

1 O Projeto

O principal objetivo deste trabalho é estudar o desempenho duma rede LTE em vários cenários usando técnicas de simulação de eventos discretos. Tal como no trabalho anterior, foi utilizado o simulador de rede ns-3 e realizamos três estudos de forma a caracterizar melhor o funcionamento da rede LTE em diferentes cenários de utilização.

As simulações apresentadas foram largamente inspiradas nas realizadas no trabalho anterior sobre Wi-Fi e nos *scripts* de exemplo de LTE (lena-simple.cc e lena-simple-epc.cpp) que foram fornecidos juntamento com o simulador de rede ns-3.

1.1 First Study - Throughput vs Distance

Para este primeiro estudo , inspiramo-nos fortemente no homologo do trabalho anterior, e assim analisamos a influência que a distância entre o equipamento do utilizador (ou "UE") e o nó de acesso possui no *throughput*. Em teoria , seria de esperar que o aumento da distância entre um receptor e um transmissor resulte na queda gradual do *throughput* do equipamento do utilizador , juntamente com a diminuição do "Signal to Noise Ratio" (SNR) ou , neste caso, do "Signal to Interference plus Noise Ratio" (SINR) que é o que nos é fornecido na execução do script.

Para obter os resultados apresentados, criámos um eNodeB na posição de origem (0,0,0) e um equipamento de utilizador (UE) com distância variável conforme a iteração ("distância",0,0). Esta simulação foi baseada no scrip "lena-simples.cc" com a adição do *override* da distância. Os dados sobre o número de bit/s presentes no meio foram obtidos do ficheiro "DlRlcStats.txt" e os dados sobre o SINR obtidos do ficheiro "DlRsrpSinrStats.txt".

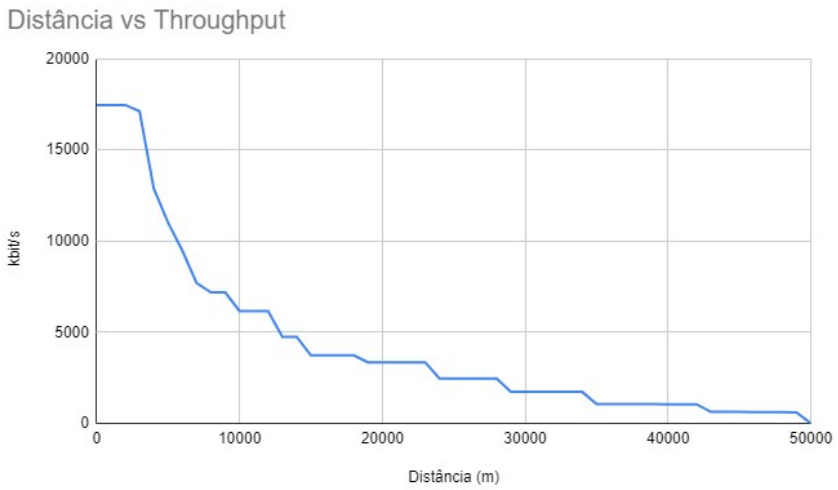

Figure 1.1: *throughput* vs Distância

Obtivemos um intervalo de confiança de +-1317m para um limite de confiança de 95%.

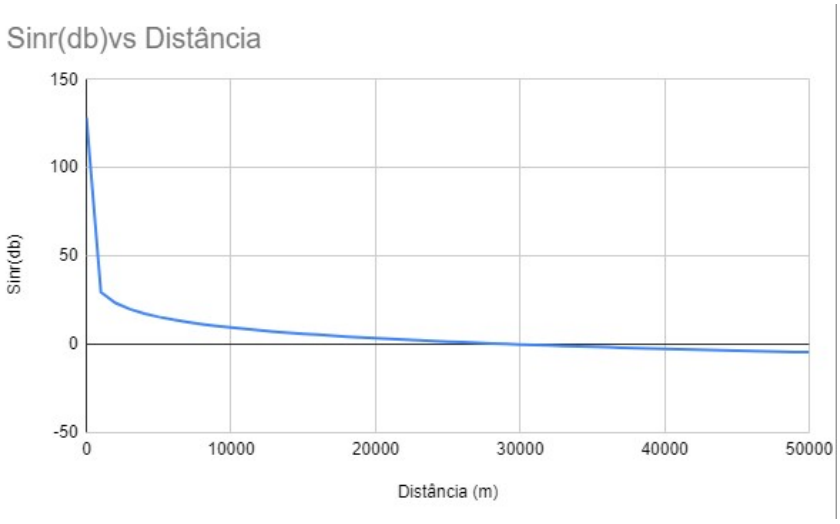

Figure 1.2: Relação entre SINR(Sinal to Interference+Noise Ratio) e Distância

Obtivemos um intervalo de confiança de +-5,2 dB para um limite de confiança de 95%.

Analisando os gráficos , podemos concluir que as nossas análises prévias se confirmam. O *throughput* entra declinio apartir dos 3000m, ponto apartir do qual desce gradualmente. Entre os 49000m e os 50000m verifica-se uma acentuada descida para valores proximos de 0.

1.2 Second Study-Throughput vs Number of UE's

O segundo estudo deste trabalho é mais um vez baseado no seu equivalente do trabalho anterior. Como tal , iremos avaliar os efeitos provocados no *throughput* provocados pela presença de vários equipamentos de utilizador na rede. É de esperar que com o aumento de utilizadores da rede , o *throughput* vá diminuindo gradualmente, uma vez que a capacidade da rede terá de ser distribuida de igual forma entre os vários equipamentos. De forma a realizar esta experiência , colocámos um eNodeB na origem (0,0,0) e vários UE's distribuidos numa circunferência com raio de 100m em torno do eNodeB, seguindo o mesmo procedimento que o segundo estudo da simulação de Wi-Fi.

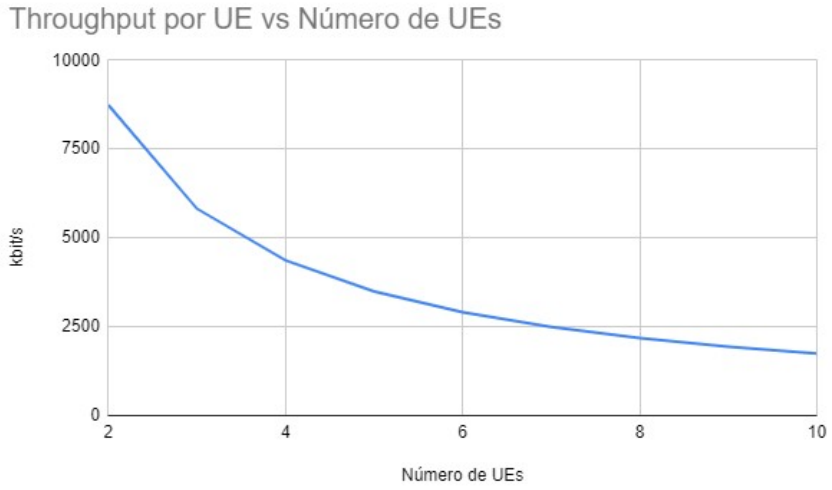

Figure 1.3: *throughput* vs Número de UE's

Obtivemos um intervalo de confiança de +-1490 kbit/s para um limite de confiança de 95%.

Facilmente confirmamos que os dados obtidos se alinham com o que era expectável. À medida que aumentámos o número de equipamentos na rede , o *throughput* de cada um deles vai diminuindo. Para um maior numero de UE's , o acréscimo de cada equipamento representa uma diferença menor dado que a rede já se encontra bastante dividida.

1.3 Third Study - Throughput with Carrier Aggregation

Para o terceiro estudo, queriamos observar a influência que a *Carrier Aggregation (CA)* ou agregação de portadoras, tinha sobre o *throughput* do sistema. Os testes foram realizados com a mesma metodologia do primeiro estudo , em que íamos aumentando progressivamente a distância entre o eNodeB e o equipamento do utilizador. A utilização de agregação de portadoras deverá permitir um aumento do *throughput* de cada utilizador , permitindo a utilização de vários blocos de frequência a um mesmo utilizador. Da mesma forma, quantos mais blocos estiverem a ser usados pelo utilizador , maior deverá ser o seu *throughput*.

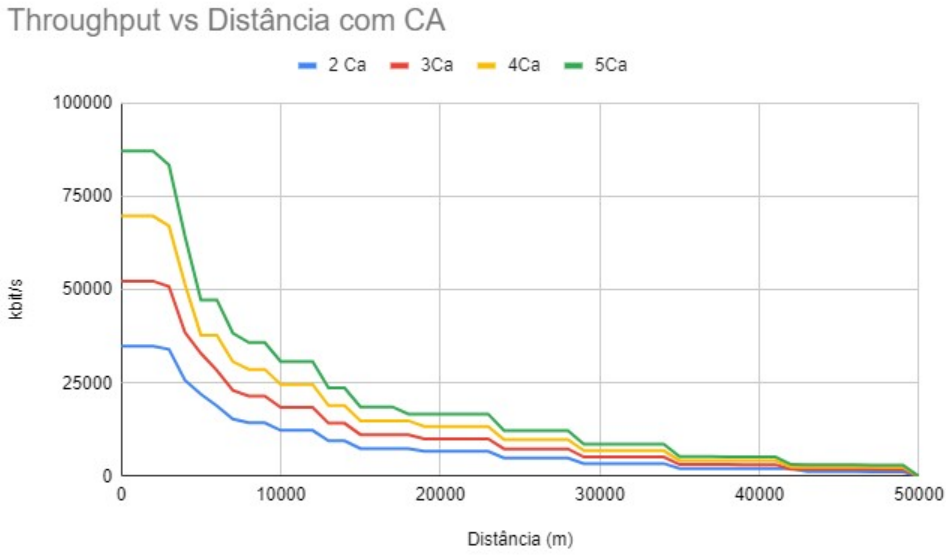

Figure 1.4: Influência do uso de CA no *throughput* vs Distância

CA	Intervalo de Confiança a 95%
2	+-2636m
3	+-3957m
4	+-5223m
5	+-6525m

A figura 1.4 demonstra o que afirmamos anteriormente. Verificamos que o aumento do numero de *carriers* traduz-se num linear aumento do *throughput* total, do que podemos concluir que a utilização de *carrier aggregation* faz com que a informação seja dividida de igual forma pelos vários canais.

A Apendice

A.1 Script para o 1º e 3º estudo

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/config-store.h"
#include <ns3/buildings-helper.h>
using namespace ns3;

int main (int argc, char *argv[])
{
  Time simTime = MilliSeconds (1050);
  bool useCa = false;
  int CaNumber=2;
  int distance=1000;
  CommandLine cmd (__FILE__);
  cmd.AddValue ("simTime", "Total duration of the ↩
     simulation", simTime);
  cmd.AddValue ("useCa", "Whether to use carrier ↩
     aggregation.", useCa);
  cmd.AddValue ("CaNumber", "Whether to use carrier ↩
     aggregation.", CaNumber);
  cmd.AddValue ("distance", "Total distance of the ↩
     simulation", distance);
  cmd.Parse (argc, argv);

  ConfigStore inputConfig;
  inputConfig.ConfigureDefaults ();
  // Parse again so you can override default values from ↩
     the command line
  cmd.Parse (argc, argv);

  if (useCa)
   {
     Config::SetDefault ("ns3::LteHelper::UseCa", ↩
        BooleanValue (useCa));
     Config::SetDefault ("ns3::LteHelper::↩
        NumberOfComponentCarriers", UintegerValue (↩
        CaNumber));
     Config::SetDefault ("ns3::LteHelper::↩
        EnbComponentCarrierManager", StringValue ("ns3::↩
        RrComponentCarrierManager"));
```

```
      }

    Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
    // Create Nodes: eNodeB and UE
    NodeContainer enbNodes;
    NodeContainer ueNodes;
    enbNodes.Create (1);
    ueNodes.Create (1);
    // Install Mobility Model
    MobilityHelper mobility;
    Ptr<ListPositionAllocator> positionAlloc = CreateObject↩
       <ListPositionAllocator> ();
    positionAlloc->Add (Vector (0.0, 0.0, 0.0));
    mobility.SetPositionAllocator (positionAlloc);

    mobility.SetMobilityModel ("ns3::↩
       ConstantPositionMobilityModel");
    mobility.Install (enbNodes);
    //BuildingsHelper::Install (enbNodes);

    Ptr<ListPositionAllocator> positionAlloc2 = ↩
       CreateObject<ListPositionAllocator> ();
    positionAlloc2->Add (Vector (0.0, distance , 0.0));
    mobility.SetPositionAllocator (positionAlloc2);
    mobility.SetMobilityModel ("ns3::↩
       ConstantPositionMobilityModel");
    mobility.Install (ueNodes);
    //BuildingsHelper::Install (ueNodes);

    // Create Devices and install them in the Nodes (eNB ↩
       and UE)
    NetDeviceContainer enbDevs;
    NetDeviceContainer ueDevs;
    // Default scheduler is PF, uncomment to use RR
    //lteHelper->SetSchedulerType ("ns3::RrFfMacScheduler")↩
       ;

    enbDevs = lteHelper->InstallEnbDevice (enbNodes);
    ueDevs = lteHelper->InstallUeDevice (ueNodes);

    // Attach a UE to a eNB
    lteHelper->Attach (ueDevs, enbDevs.Get (0));

    // Activate a data radio bearer
    enum EpsBearer::Qci q = EpsBearer::GBR_CONV_VOICE;
    EpsBearer bearer (q);
    lteHelper->ActivateDataRadioBearer (ueDevs, bearer);
    lteHelper->EnableTraces ();
    Simulator::Stop (simTime);
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
  }
```

A.2 Script para o 2º estudo

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/config-store.h"
#include <ns3/buildings-helper.h>
#include <cstring>
#include <string>
#include <math.h>
#define PI 3.14159265
using namespace ns3;
int main (int argc, char *argv[])
{
  Time simTime = MilliSeconds (5000);
  bool useCa = false;
  int distance=100;
  int nr_of_ue = 2;

  CommandLine cmd (__FILE__);
  cmd.AddValue ("simTime", "Total duration of the ↩
     simulation", simTime);
  cmd.AddValue ("useCa", "Whether to use carrier ↩
     aggregation.", useCa);
  cmd.AddValue ("distance", "Total distance of the ↩
     simulation", distance);
  cmd.AddValue ("nr_of_ue", "Total distance of the ↩
     simulation", nr_of_ue);
  cmd.Parse (argc, argv);
  ConfigStore inputConfig;
  inputConfig.ConfigureDefaults ();
  // Parse again so you can override default values from ↩
     the command line
  cmd.Parse (argc, argv);

  if (useCa)
   {
     Config::SetDefault ("ns3::LteHelper::UseCa", ↩
        BooleanValue (useCa));
     Config::SetDefault ("ns3::LteHelper::↩
        NumberOfComponentCarriers", UintegerValue (2));
     Config::SetDefault ("ns3::LteHelper::↩
        EnbComponentCarrierManager", StringValue ("ns3::↩
        RrComponentCarrierManager"));
   }

  Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
  // Create Nodes: eNodeB and UE
  NodeContainer enbNodes;
  NodeContainer ueNodes;
```

```
    enbNodes.Create (1);
    ueNodes.Create (nr_of_ue);
    // Install Mobility Model
    MobilityHelper mobility;
   Ptr<ListPositionAllocator> positionAlloc = CreateObject<↩
      ListPositionAllocator> ();
    positionAlloc->Add (Vector (0.0, 0.0, 0.0)); //node 0 ↩
       Sink node
    mobility.SetPositionAllocator (positionAlloc);
    mobility.SetMobilityModel ("ns3::↩
       ConstantPositionMobilityModel");
    mobility.Install (enbNodes);
    //BuildingsHelper::Install (enbNodes);
    double current_angle = 0.0;
    double angle_step = 360.0 / nr_of_ue;
    Ptr<ListPositionAllocator> positionAlloc2 = ↩
       CreateObject<ListPositionAllocator> ();
    while (current_angle < 360.0 )
      {
          //std::cout << "Current angle: " << current_angle↩
              << "\n";
          positionAlloc2->Add (Vector (distance*cos(↩
             current_angle * PI / 180.0), distance*sin(↩
             current_angle * PI / 180.0), 0.0)); //Sender ↩
             node x
          current_angle += angle_step;
      }
    mobility.SetPositionAllocator (positionAlloc2);
    mobility.SetMobilityModel ("ns3::↩
       ConstantPositionMobilityModel");
    mobility.Install (ueNodes);
    //BuildingsHelper::Install (ueNodes);
    // Create Devices and install them in the Nodes (eNB ↩
       and UE)
    NetDeviceContainer enbDevs;
    NetDeviceContainer ueDevs;
    // Default scheduler is PF, uncomment to use RR
    //lteHelper->SetSchedulerType ("ns3::RrFfMacScheduler")↩
       ;
    enbDevs = lteHelper->InstallEnbDevice (enbNodes);
    ueDevs = lteHelper->InstallUeDevice (ueNodes);
    // Attach a UE to a eNB
    lteHelper->Attach (ueDevs, enbDevs.Get (0));
   // Activate a data radio bearer
    enum EpsBearer::Qci q = EpsBearer::GBR_CONV_VOICE;
    EpsBearer bearer (q);
    lteHelper->ActivateDataRadioBearer (ueDevs, bearer);
    lteHelper->EnableTraces ();
    Simulator::Stop (simTime);
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
  }
```