Simulating LTE Cellular Systems: an Open Source Framework

Giuseppe Piro, Student Member, IEEE, Luigi Alfredo Grieco, Member, IEEE, Gennaro Boggia, Senior Member, IEEE, Francesco Capozzi, Student Member, IEEE, and Pietro Camarda

Survey

Published by M.K. Han
LIST
There are four main components:

- the **Simulator**,
- the **NetworkManager**,
- the **FlowsManager**,
- the **FrameManager**.

Events scheduling is handled by the *Simulator* class.
In details, at the beginning of each simulation, the *Calendar* is populated by only three events:
(i) the start of the simulation, using the `Simulator::Run()` method;
(ii) the start of the **FrameManager**, using the `FrameManager::StartFrame()` method;
(iii) the end of the simulation, using the `Simulator::Stop()` method.

Three kinds of LTE network nodes have been implemented:
UE, eNB, and MME/GW. They are created, destroyed, and handled by the **NetworkManager**.
Each LTE network node can be a source or a destination of data flows, defined by the classical five-tuple:
source and destination IP addresses, sender and receiver ports, and the transport protocol type.

- the **Simulator**,
- the **NetworkManager**,  
- the **FlowsManager**,  
- the **FrameManager**.

**LTE-Sim** provides a support for radio resource allocation in a time-frequency domain.
In the time domain, radio resources are distributed every Transmission Time Interval (TTI), each one lasting 1 ms.
Furthermore each TTI is composed by two time slot of 0.5 ms, corresponding to 14 OFDM symbols in the default configuration with short cyclic prefix; 10 consecutive TTIs form the LTE Frame.

In the frequency domain, instead, the whole bandwidth is divided into 180 kHz sub-channels, corresponding to 12 consecutive and equally spaced sub-carriers. As the sub-channel dimension is fixed, for different system bandwidth configurations the number of sub-channels varies accordingly.

A time/frequency radio resource spanning over one 0.5 ms time slot in the time domain and over one sub-channel in the frequency domain is called Resource Block (RB) and corresponds to the smallest radio resource that can be assigned to a UE for data transmission.
**SOFTWARE DESIGN : Frame Structure(1/3)**

**LTE-Sim** supports two frame structure types proposed in [14] for the E-UTRAN.
The first one is defined for FDD mode and it is called frame structure type 1. The second one is called frame structure type 2 and is defined for TDD mode.


For the frame structure type 1, the bandwidth is divided into two parts, allowing downlink and uplink data transmissions, simultaneously in the time. **For the frame structure type 2**, the LTE Frame is divided into two consecutive half-frames, each one lasting 5 ms.

According to [14], Tab. II reports seven implemented uplink-downlink configurations of type 2 (TDD) frame. We note that sub-frames 0 and 5 are always reserved for downlink transmission.

The frame structure and the TDD frame configuration have been defined in the FrameManager.

**TABLE II**

<table>
<thead>
<tr>
<th>configuration number</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; half frame</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; half frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>D</td>
<td>S</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>S</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>S</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>S</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>S</td>
</tr>
<tr>
<td>6</td>
<td>D</td>
<td>S</td>
</tr>
</tbody>
</table>

D = downlink sub-frame; U = uplink sub-frame; S = Special Sub-frame.
SOFTWARE DESIGN : Frame Structure (2/3)

- **FDD**
  - Uplink and downlink separated in frequency domain
  
  ![FDD Diagram](image)

  One radio frame, $T_{\text{frame}} = 10$ ms

  - One subframe, $T_{\text{subframe}} = 1$ ms

<table>
<thead>
<tr>
<th>Subframe</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **TDD**
  - Uplink and downlink separated in time domain ➔ "special subframe"
  - Same numerology etc as FDD ➔ economy of scale

![TDD Diagram](image)
SOFTWARE DESIGN: Frame Structure (3/3)

PHYSICAL RESOURCES: Frame & RB mapping

SOFTWARE DESIGN: UE Channel Status & Resource Scheduling

Time-frequency fading, user #1

Time-frequency fading, user #2

User #1 scheduled

User #2 scheduled

Time

Frequency

1 ms
180 kHz
LTE-Sim Power Allocation (ENodeB)

/src/scenarios/simple.h (con't)

```c
// CREATE COMPONENT MANAGERS
Simulator simulator = Simulator::GetInstance();
FrameManager frameManager = FrameManager::GetInstance();
NetworkManager networkManager = NetworkManager::GetInstance();
FlowManager flowManager = FlowManager::GetInstance();

// Create CHANNELS
LteChannel dlCh = new LteChannel();
LteChannel ulCh = new LteChannel();

// CREATE SPECTRUM
BandwidthManager::spectrum = new BandwidthManager(5, 5, 0, 0);

// CREATE CELL
int idCell = 0;  // km
int radius = 1;  // km
int minDistance = 0.0035;  // km
int posX = 0;
int posY = 0;
Cell cell = networkManager->CreateCell(idCell, radius, minDistance, posX, posY);

// Create ENodeB
int idEnb = 1;
ENodeB* enb = networkManager->CreateEnodeb(idEnb, cell, posX, posY, dlCh, ulCh, spectrum);
```

Network-manager.cpp

```c
ENodeB::CreateEnodeb(int id, Cell* cell, double posX, double posY, LteChannel* dlCh, LteChannel* ulCh, BandwidthManager* spectrum)
{
    ENodeB* enb = new ENodeB(id, cell);  // ENodeB (i, cell);
    enb->GetPhy()->SetDlChannel(dlCh);
    enb->GetPhy()->SetUlChannel(ulCh);
    enb->GetPhy()->GetSchedulerManager()->SetSchedulerManager(enb);
    GetNodeBContainer()->push_back(enb);
    return enb;
}
```

/src/device/EnodeB.cpp

```c
void ENodeB::SetDLScheduler(ENodeB::DLSchedulerType type)
{
    ENodeB::DLScheduler* scheduler = GetMacEntity()->GetSchedulerManager()->GetSchedulerManager();
    switch (type)
    {
        case ENodeB::DLScheduler_TYPE_PROPORTIONAL_FAIR:
        scheduler = new DL_PF_PacketsScheduler();
        break;
```

bandwidth-manager.cpp

```c
BandwidthManager::BandwidthManager(double ulBW, double dlBW, int ulOffset, int dOffset)
{
    m_ulBandwidth = ulBW;
    m_dlBandwidth = dlBW;
    m_ulOffset = ulOffset;
    m_dlOffset = dOffset;
    m_operativeSubBand = 3;
    m_dlSubChannels.clear();
    m_ulSubChannels.clear();
    m_ulBandwidth = ulBW;
    m_dlBandwidth = dlBW;
    m_ulOffset = ulOffset;
    m_dlOffset = dOffset;
    m_operativeSubBand = 3;
    m_dlSubChannels.clear();
    m_ulSubChannels.clear();

    if (dlBW == 3)
    {
        for (int i = 0; i < dOffset + RBs For 3 MHz; i++)
        {
            m_dlSubChannels.push_back(DL_LOW_FREQUENCY_BAND_1 + (1 * 0.18));
        }
    }
    if (dlBW == 5)
    {
        for (int i = 0; i < dOffset + RBs For 5 MHz; i++)
        {
            m_dlSubChannels.push_back(DL_LOW_FREQUENCY_BAND_1 + (1 * 0.18));
        }
    }
    else if (dlBW == 10)
    {
        // Define UL_LOW_FREQUENCY_BAND_1 1920 MHz
        // Define UL_HIGH_FREQUENCY_BAND_1 1980 MHz
        // Define DL_LOW_FREQUENCY_BAND_1 2110 MHz
        // Define DL_HIGH_FREQUENCY_BAND_1 2170 MHz
```

E-UTRA Bands, Channel Bandwidths, and Frequency Allocations

**E-UTRA Operating Band** indicates the carrier frequency. Not all LTE frequency bands support all bandwidths. Chart includes FDD and TDD channels.

LTE-Sim Power Allocation (ENodeB)

```
ENodeB* NetworkManager::CreateENodeb (int id,  
    Cell* cell,  
    double pos_X, double pos_Y,  
    LteChannel *dlCh, LteChannel *ulCh,  
    BandwidthManager *bm)  
{  
    ENodeB* enb = new ENodeB (id, cell, pos_X, pos_Y); //ENodeB (1, cell);  
enb->GetPhy ()->SetDLChannel (dlCh);  
enb->GetPhy ()->SetULChannel (ulCh);  
enb->GetPhy ()->SetBandwidthManager (bm);  
ulCh->AddDevice (emb);  
GetENodeBContainer ()->push_back (emb);  
return enb;  }
```

```
void EnbLtePhy::DoSetBandwidthManager (void)  
{  
    BandwidthManager* s = GetBandwidthManager ();  
    std::vector<double> channels = s->GetDLSubChannels ();  
    TransmittedSignal txSignal = new TransmittedSignal ();  
    std::vector<double> values;  
    std::vector<double>::iterator it;  
    double powerTx = pow (10., (GetTxPower () - 30) / 10); // in natural unit  
    double txPower = 10 * log10 (powerTx / channels.size ()); // in dB  
    for (it = channels.begin (); it != channels.end (); it++)  
    {  
        values.push_back (txPower);  
    }  
    txSignal->SetValues (values);  
    txSignal->SetBandwidthManager (s->Copy());  
    SetTxSignal (txSignal);  }
```

`channels.size()` is 25 (`RBs_for_5MHz`)  
(subchannel is 25)  
ref: previous page
LTE-Sim RB Allocation (eNodeB)

/src/scenarios/simple.h

```c
//Create GM
Gateway *gw = networkManager->CreateGateway();

//Create UE
int idUE = 1;
int posX_ue = 40; //m
int posY_ue = 80; //m
int speed = 3; //m/s
double speedDirection = 45;

UserEquipment* ue = networkManager->CreateUserEquipment(idUE, posX_ue, posY_ue, speed, speedDirection, cell, enb);

//Create an Application
QoSParameters *qos = new QoSParameters();
int applicationID = 9;
int srcPort = 8;
int dstPort = 1093;
double startTime = 0.1; //s
double stepTime = 0.12;
Application* app = new FlowManager->CreateApplication(appId, gw, ue, srcPort, dstPort, appType:TRANSPORT_PROTOCOL_TYPE_UDP, applicationType:APPLICATION_TYPE_INFINITE_BUFFER, qos, startTime, stepTime);
simulator->setStop(0.13);
simulator->run();
```

/src/componentManagers/NetworkManager.cpp

```c
UserEquipment* ue = new UserEquipment(id,
posX, posY, speed, speedDirection, cell, enb, 0, Mobility::RANDOM_DIRECTION);

ue->GetPhy() -> SetDlChannel (enb->GetPhy() -> GetDlChannel ());
ue->GetPhy () -> SetUlChannel (enb->GetPhy() -> GetUlChannel ());

FullbandCqiManager *cqiManager = new FullbandCqiManager ();
cqiManager->GetCqiReportingMode (cqiManager::PERIODIC);
cqiManager->GetCqiReportingInterval ();
cqiManager->GetCqiDevice (ue);
ue->SetCqiManager (cqiManager);
enb->RegisterUserEquipment (ue);

MacroCellUrbanAreaChannelRealization *c_d1 = new MacroCellUrbanAreaChannelRealization (emb, ue);
ue->GetPhy () -> SetPropagationLossModel (c_d1);
ue->GetPhy () -> GetPropagationLossModel (c_d1);

UEContextManager->GetUserEquipmentContainer () -> push_back (ue);

return ue;
```

Eneb : Send PacketBurst

/src/phy/enb-lte-phy.cpp : eNodeB

```c
void
EnbLtePhy::StartTx (PacketBurst* p)
{
    std::cout "Node " GetDevice () -> GetID () NetworkNode () StartPhyTx () GetDevice ()
GetDlChannel () -> StartTx (p, GetDlSignal (), GetDevice ());
```

*GetTxSignal() : 각 subchannel에 할당된 txpower가 저장되어 있다.
ref: 이전 page참조: DoSetBandwidthManager() StartTx()는 UE의 UEItePhy::StartRx(...) 를 Call함.
UE: Send CQI

#include <vector>

class AMCModule
{
public:
    vector<double> CreateCQIFeedbacks(double snr)
    {
        // Code
    }

private:
    // Code
};

void FullbandCqiManager::CreateCqiFeedbacks(vector<double> snr)
{
    #ifdef TEST_CQI_FEEDBACKS
        std::cout << "FullbandCqiManager::CreateCqiFeedbacks" << std::endl;
    #endif

    #ifdef AMC_MAPPING
        std::cout << snr << std::endl;
        for (int i = 0; i < snr.size(); i++)
        {
            std::cout << snr.at(i) << " ";
        }
        std::cout << std::endl;
    #endif

    UserEquipment* thisNode = (UserEquipment*)GetDevice();
    NetworkNode* targetNode = thisNode->GetTargetNode();

    AMCModule* amc = GetDevice()->GetProtocolStack()->GetMacEntity()->GetAmcModule();
    std::vector<int> cqi = amc->CreateCqiFeedbacks(snr);

    CqiIdealControlMessage* msg = new CqiIdealControlMessage();
    msg->SetSourceDevice(thisNode);
    int nSubChannels = cqi.size();
    std::vector<double> dSubChannels = targetNode->GetPhy()->GetBandwidthManager()->GetDSubChannels();
    for (int i = 0; i < nSubChannels; i++)
    {
        msg->AddNewRecord(dSubChannels.at(i), cqi.at(i));
    }
    SetLastSent();

    thisNode->GetPhy()->SendIdealControlMessage(msg);

    #ifdef TEST_CQI_FEEDBACKS
        std::cout << "\t snr" << snr_ << " cqi" << cqi_ << std::endl;
    #endif

    return cqi;
};
eNodeB

FrameManager::StartFrame()
FrameManager::ResourceAllocation()
{
    ...

    //record->DownlinkResourceBlockAllocation();
    Simulator::Init()->Schedule(0, 0, &ENodeB::DownlinkResourceBlockAllocation, record);
}

void ENodeB::DownlinkResourceBlockAllocation (void)
{
    if (GetDLScheduler () != NULL && GetNumberOfUserEquipmentRecords () > 0)
    {
        GetDLScheduler ()->Schedule();
    }
    else
    {
        //send only reference symbols
        //PacketBurst *pb = new PacketBurst ();
        //SendPacketBurst (pb);
    }
}

void DownlinkPacketScheduler::DoSchedule (void)
{
    #ifdef SCHEDULER_DEBUG
        std::cout << "Start DL packet scheduler for node " << GetMacEntity ()->GetDevice ()->GetIDNetworkNode () << std::endl;
    #endif

    UpdateAverageTransmissionRate ();
    SelectFlowsToSchedule ();

    if (GetFlowsToSchedule ()->size () == 0)
    {}
    else
    {
        RBsAllocation ();
    }
    StopSchedule ();
}

DL_PF_PacketScheduler 가 셋팅되어 있는 상태 (PPT 8page)
ifdef SCHEDULER_DEBUG
std::cout << "t = RB " << g << " assigned to the "
<< flow_name << " getApplication () ->getApplicationID () 
<< std::endl;
#endif

double sinr = amc->getSinrFromCQI (scheduledFlow(). getSINR () / (G));
SINRForCQIIndex[15] = {
-4.63, -2.6, -0.12, 2.26, 4.73, 7.53, 8.67, 11.32, 
14.24, 15.21, 18.53, 21.32, 23.47, 28.49, 34.6
};

double effectiveSinr = GetEssEffectiveSinr (L_bFlowScheduledSNR (L_scheduledFlowIndex));
int mcs = amc->GetMCSFromCQI (effectiveSinr);  
if (mcs == amc->GetMCSFromCQI (effectiveSinr)) return mcs; 
int transportBlockSize = amc->GetTBSFromMCS (mcs, scheduledFlow());

for (int rb = 0; rb < transportBlockSize; rb++)
{  
double sinr = amc->getSinrFromCQI (flow->GetCQIFeedbacks (r). GetSinr () / transportBlockSize (r) (GetEstSinrValues (flow);  
int mcs = amc->GetMCSFromCQI (effectiveSinr);  
if (mcs == amc->GetMCSFromCQI (effectiveSinr)) return mcs; 
int transportBlockSize = amc->GetTBSFromMCS (mcs, flow->GetListOfAllocatedRBs (r) (GetEstSinrValues (flow);  

// Table 10.3.1.1 of 3GPP TS 36.213 v8.8.0
int McsToTbs[28] = {
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 
19, 20, 21, 22, 23, 24, 25, 26
};

for (int rb = 0; rb < transportBlockSize; rb++)
{  
double sinr = amc->getSinrFromCQI (flow->GetCQIFeedbacks (r). GetSinr () / transportBlockSize (r) (GetEstSinrValues (flow);  
int mcs = amc->GetMCSFromCQI (effectiveSinr);  
if (mcs == amc->GetMCSFromCQI (effectiveSinr)) return mcs; 
int transportBlockSize = amc->GetTBSFromMCS (mcs, flow->GetListOfAllocatedRBs (r) (GetEstSinrValues (flow);  

Throughput Calculation ref. site
http://4g-lte-world.blogspot.kr/2012/12/transport-block-size-code-rate-protocol.html