System Level Simulator
User Manual

Title	System Level Simulator – User Manual
Document Number	
Author	Jaewon Lee
Creation Date	September 28, 2016
Last Modified	January 17, 2019
Version	1.0
Status	Working
Path To File	

Modified By	Date	Version	Reason For Change
Jaewon Lee	1/23/17	0.01	Created (HLD)
Jaewon Lee	2/17/18	0.1	Initial version (HLD)
Minsig Han	3/14/18	0.2	List of features
Minsig Han	5/19/18	0.3	System Level Simulator Structure
Jaewon Lee	8/22/18	0.4	How to use
Jaewon Lee	11/7/18	0.5	Expansion to 5G eMBB Simulator
Minsig Han	12/29/18	0.6	Expansion to 5G URLLC Simulator
Jaewon Lee	1/17/19	1.0	Minor modification

TABLE OF CONTENTS

List of Figures

List of Tables

1. Introduction

A link level simulator (LLS), a system level simulator (SLS), and a network level simulator (NLS) are examples of simulators for evaluating the performance of a mobile communication system. The link level simulator has the main purpose of measuring the error probability according to the signal to interference ratio using the accurate physical layer model including channel coding and modulation and demodulation. In the system level simulator, the main purpose is to measure system level performance based on the channel model according to the location of the MS and the wireless interworking model of the BS and the MS where a large number of BSs and MSs are arranged according to the experimental environment.

1.1 Purpose

This document is intended for use by software engineers working on the system level simulator. In this document, firstly, we describe the system level simulator in a simple form to improve the understanding of the system level simulator and understand the structure of the simulator. The goal is to develop the ability to develop simulators of more complex structures, handling extensions to 5G simulators.

1.2 Contact points

Chung G. Kang: ccgkang@korea.ac.kr

Minjoong Rim: minjoong@dongguk.edu

Jaewon Lee: lijrew@korea.ac.kr

Minsig Han: als4585@korea.ac.kr

1.3 Definitions

The following definitions are used for the simulator and this document.

Table 1-1 Definitions

Word	Description
BS	Related to cell Refers to sector (NOT base station site), TRP(transmission reception point), RSU(road side unit), RRH(remote radio head), RU(RF unit), or RRU(remote radio unit) We will not use the terms TRP, RSU, RRH, RU, or RRU in this simulator.
MS	Refers to mobile station, user, device, UE(user equipment), UT(user terminal) We will not use the terms UE or UT in this simulator.
Site	Related to base station position Base station, site, eNode-B, eNB We will not use the terms eNode-B or eNB in this simulator.
BBU	Baseband unit, DU(digital unit)

	We will not use the term DU in this simulator.

1.4 Acronyms

The following acronyms are used for the simulator and this document.

Table 1-2 Acronyms

Acronym	Description
MS	Mobile Station
BS	Base Station
RS	Relay Station
BBU	Baseband Unit
RX	Receiver
TX	Transmitter
DL	Downlink
UL	Uplink
FDD	Frequency Division Duplexing
TDD	Time Division Duplexing
TTI	Transmission Time Interval
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access@
RB	Resource Block
ARQ	Automatic Repeat Request
HARQ	Hybrid ARQ
ACK	Acknowledgment
NACK	Negative Acknowledgment
LOS	Line of Sight
NLOS	Non Line of Sight
SNR	Signal to Noise Ratio
SINR	Signal to Interference plus Noise Ratio
ESM	Effective SINR Mapping
FER	Frame Error Rate
CDF	Cumulative Distribution Function
V2X	Vehicular to Anything
D2D	Device to Device Communication
IoT	Internet of Things
CoMP	Coordinate Multi Point
MIMO	Multiple Input Multiple Output
FD-MIMO	Full Dimensional MIMO
MU-MIMO	Multi-User MIMO
ULA	Uniform Linear Arrays
CDL	Clustered Delay Line
TDL	Tapped Delay Line
AoD	Azimuth angle of departure
AoA	Azimuth angle of arrival
ZoD	Zenith angle of departure
ZoA	Zenith angle of arrival
MAC	Media Access Control

RRM	Radio Resource Management
GUI	Graphical User Interface
SLS	System Level Simulation, System Level Simulator
LLS	Link Level Simulation, Link Level Simulator
NS	Network Simulation, Network Simulator
sim	Simulation, Simulator
config	Configuration
param	Parameter
pos	Position
p	Pointer

1.5 Development environment

- OS: Windows 10
- Language: C++
- Compiler: Visual studio 2015

1.6 Terminology

Various terms and definitions related to BSs, MSs, etc. are used in the papers and standards. Although one simulator can be used for various purposes, it is necessary to unify terms constituently to avoid confusion when using simulator. The following terms are used in this simulator.

Table 1-3 Term used in the simulator

Terminology	Definitions
BS	Related to cell Refers to sector (NOT base station site), TRP(transmission reception point), RSU(road side unit), RRH(remote radio head), RU(RF unit), or RRU(remote radio unit) We will not use the terms TRP, TRxP, RSU, RRH, RU, or RRU in this simulator.
MS	Refers to mobile station, user, device, UE(user equipment), UT(user terminal) We will not use the terms UE or UT in this simulator.
Site	Related to base station position Base station, site, eNode-B, eNB, gNB We will not use the terms eNode-B or eNB in this simulator.

2. List of features

2.1 Performance metrics

2.1.1 User Equipment (UE) Specific

- Beamforming direction
- User packet outage rate (POR)
- User perceived throughput (UPT)
- Signal to Interference and Noise Ratio (SINR)
- etc

2.1.2 Cell Specific

- Average throughput (peak, mean, edge)
- Average spectral efficiency
- Wideband Signal to Interference and Noise Ratio (SINR)
- Average user perceived throughput (UPT)
- Beamforming direction
- Packet outage rate (POR)
- etc

2.2 Simulation parameters for configuration

2.2.1 General parameters

- Frequency
- Bandwidth
- Number of transmit and receive antennas
- Planar antenna array structure
  - ✓ Number of antenna ports
  - ✓ Number of antenna elements per port (vertical direction)
  - ✓ Antenna element spacing in horizontal and vertical direction
  - ✓ Polarization type and slant angle

2.2.2 Network layout

- Regular hexagonal grid with fixed inter-eNodeB distance for UMi, UMa and Rural
- Regular hexagonal grid with small cell for Dense Urban
- Rectangular grid for Indoor Hotspot

2.2.3 Pathloss

- Spatial channel model based
- Exponential model based

2.2.4 Shadowing

- 2-D correlated lognormal-distribution
- Uncorrelated

2.2.5 Small scale fading

- 3GPP 3D spatial channel model

2.2.6 MS Setting

- Spatial distribution of MSs
- Speed of MS movement
- Receiver noise figure and thermal noise density

2.2.7 BS Setting

- Maximum antenna gain
- Antenna gain pattern
  - ✓ TR 36.873 antenna element gain pattern

2.2.8 Scheduler

- Round Robin
- Proportional fair
- Proportional fair for Multiuser MIMO (MU-PF)

2.3 Preconfigured Setups

- Simple version for Education
- 5G eMBB version
- 5G URLLC version

3. System Level Simulator Structure e

3.1 Overview

The simulator is composed of Simulation Main which manages the whole simulation as shown in the following figure, Network Configuration which manages the arrangement of BSs and MSs, Radio Resource Management which manages radio resources, Link Performance which measures simulation performance, and Channel Model. Each module performs the following functions.

- Network Configuration: Network, BS placement, MS placement, mobility, association, admission control
- Radio Resource Management: Scheduling, MS feedback, power control, rate control, hybrid ARQ
- Link Performance: Performance, FER calculation, transmission and reception, effective SINR mapping
- Channel Model: Channel, antenna, large scale channel, small scale channel, shot term channel

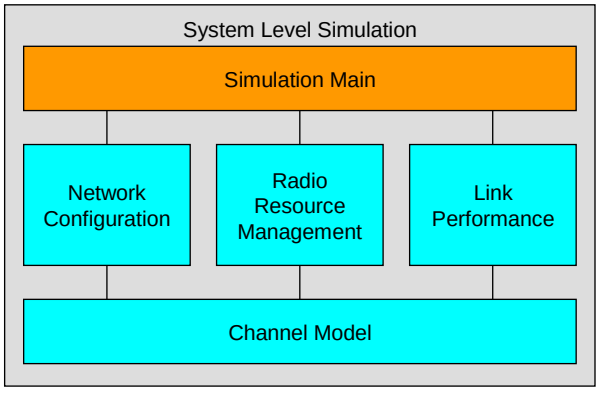

Figure 3-1. System Level Simulator Structure

3.2 Class Structure

3.2.1 Class Overview

The simulator has the following class structure. There are "SystemSim" which includes variables related to the overall simulation, "SystemBS" which represents each BS (more precisely, cell or sector), and "SystemMS" class which represents each MS. The reason for dividing a module horizontally for each class is to make it easier to replace with the module with another module.

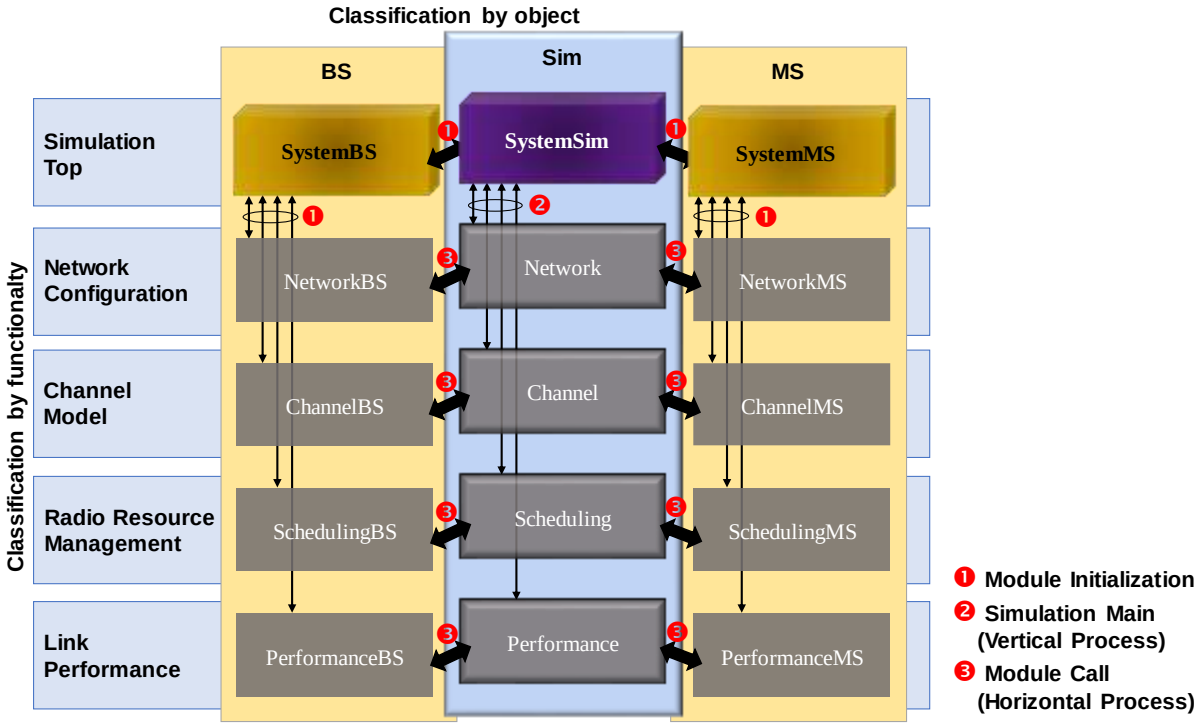

Figure 3-2. System Level Simulator Class Structure

Table 3-1 Term used in the simulator

Directory	Name	Simulation	Base Station	Mobile Station
Simulation Top	Class	SystemSim	SystemBS	SystemMS
	Variable	Sim	BS	MS
Network Configuration	Class	Network	NetworkBS	NetworkMS
	Variable	network	network	network
Radio Resource Management	Class	Scheduling	SchedulingBS	SchedulingMS
	Variable	scheduling	scheduling	scheduling
Link Performance	Class	Performance	PerformanceBS	PerformanceMS
	Variable	performance	performance	performance
Channel Model	Class	Channel	ChannelBS	ChannelMS
	Variable	channel	channel	channel

The variables associated with these classes are defined as global variables. There are one global variable for each class (i.e., Sim related to the simulation, BS related to the base station and MS related to the mobile station).

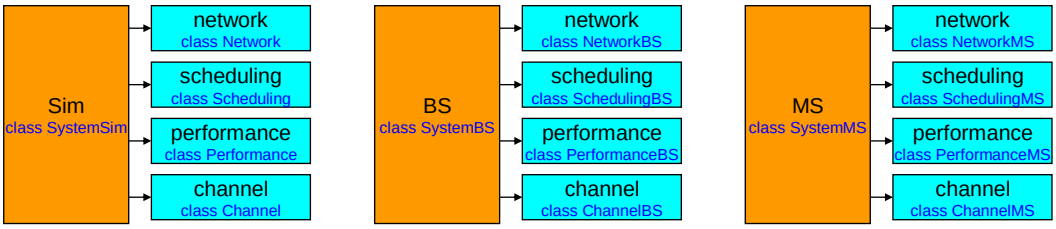

Figure 3-3. Global Variables

3.2.2 Key Classes

The main classes of the simulator are shown in Figure 3-2 and Table 3-1. First, SystemSim, SystemBS, and SystemMS define global variables Sim, BS, and MS. They are responsible for simulation at the top level and include network, scheduling, performance, and channel for Network Configuration, Radio Resource Management, Link Performance, and Channel Model.

```
class SystemSim
{
  ...
  class Network *network;              // Parameters related to network configuration
  class Scheduling *scheduling;        // Parameters related to radio resource management
  class Performance *performance;  // Parameters related to link performance
  class Channel *channel;     // Parameters related to channel

  void Initialize(string fileName);      // Initialization
  void InitializeIteration(int iteration);  // Iteration initialization
  void InitializeNode();                 // Initialize BSs and MSs
  void InitializeTTI(int tti);           // TTI initialization
  void ConcludeNode();                   // Conclude BSs and MSs
  void ConcludeIteration();              // Conclude Iteration the simulation
  void Conclude();              // Concluding the simulation
};

class SystemBS
{
```

```
  class NetworkBS *network;              // Network configuration info at BS
  class SchedulingBS *scheduling;  // Scheduling info at BS
  class PerformanceBS *performance;      // Performance info at BS
  class ChannelBS *channel;              // Large scale channel correlation according to Site

  SystemBS(int bs);              // Declare constructor
  ~SystemBS();                   // Declare destructor
};

class SystemMS
{
  class NetworkMS *network;              // Network configuration info at MS
  class SchedulingMS *scheduling;        // Scheduling info at MS
  class PerformanceMS *performance;      // Link performance info at MS
  class ChannelMS *channel;              // Channel model info at MS

  SystemMS(int ms);                      // Declare constructor
  ~SystemMS();                   // Declare destructor
};
```

When a method defined in a class is used, it is used as a method dependent on the variable name. For example, it is used in the form of Sim.Initialize(). Network Configuration Modules include Network, NetworkBS, and NetworkMS.

```
class Network
{
  ...
  void PlaceMacroBS();                   // Macro BS placement
  void PlaceMicroBS();                   // Micro BS placement
  void PlaceMsOnMacroBS();               // MS placement on Macro BS
  void PlaceMsOnMicroBS();               // MS placement on Micro BS
  void MoveMS();                 // MS movement
};

class NetworkBS
{
  ...
  void PlaceMacroBS(int bs, int site, int sector);          // Macro BS placement
```

```
  void PlaceMicroBS(int bs, int site, int sector, int macroBS);  // Micro BS placement
};

class NetworkMS
{
  ...
  void Associate();                          // Association
  void PlaceMsOnMacroBS(int ms, int bs);          // MS placement on Macro BS
  void PlaceMsOnMicroBS(int ms, int bs);          // MS placement on Mciro BS
};
```

For example, if Sim.network-> PlaceMacroBS () is called, this method calls BS [bs] -> network-> PlaceMacroBS () again.

Radio Resource Management modules include Scheduling, SchedulingBS, and SchedulingMS.

```
class Scheduling
{
  ...
  void Feedback();                           // MS feedback
  void Schedule();                           // Scheduling
};

class SchedulingBS
{
  ...
  void Schedule();                           // Scheduling
};

class SchedulingMS
{
  ...
  void Feedback();                           // Feedback
};
```

Link Performance modules include Performance, PerformanceBS, and PerformanceMS.

```
class Performance
{
  ...
  void Measure();                    // Throughput measurement
};

class PerformanceBS
{
  ...
  void Measure();                    // Throughput measurement
};

class PerformanceMS
{
  ...
  void Measure();                    // Throughput measurement
};
```

Channel Model modules include Channel, ChannelBS, and ChannelMS.

```
class Channel
{
  ...
  void CalculateLongTermChannel();          // Calculate long term channel
  void CalculateShortTermChannel();         // Calculate short term channel
};

class ChannelBS
{
  ...
}

class ChannelMS
{
  ...
  void CalculateLongTermChannel();          // Calculate long term channel
  void CalculateShortTermChannel();         // Calculate short term channel
```

```
  void CalculateSINR(int ms);                    // Calculate SINR
};
```

3.3 Configuring the simulator directory

The simulator consists of the following directories. The Simulation Top that manages the simulation includes the Simulation Main directory and directories of Global and Network Elements for which global variables and types are defined. The body of the simulator is composed of four modules: Network Configuration, Radio Resource Management, Link Performance, and Channel Model. Each module has three parts: Sim, BS, and MS. It has a default directory and can contain additional directories. There is also a library and a data directory correlated with I / O data.

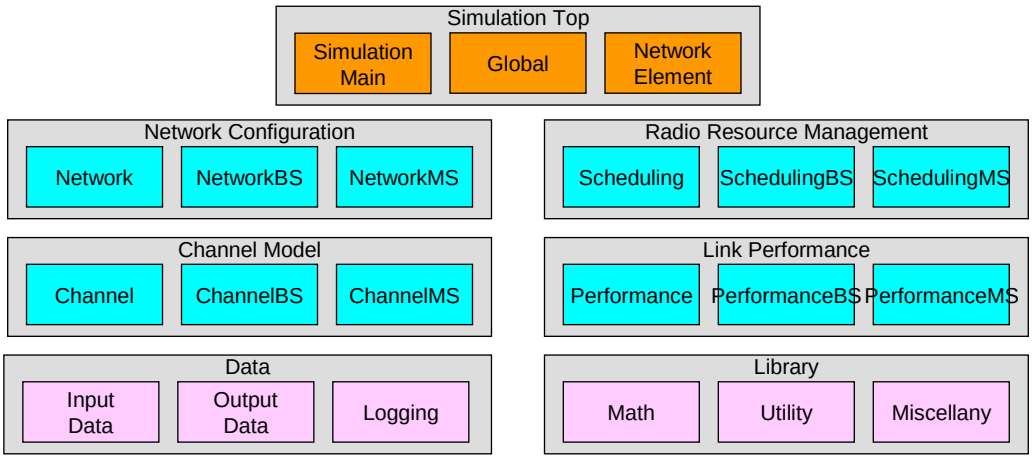

Figure 3-4. Configuring the simulator directory

3.4 Configuring the Simulator File

Each file is a pair of cpp and header and contains one object. That is, the files are organized per object. Each directory can have multiple files with the same name and different postfixes. Depending on the simulation configuration, appropriate files can be selected and used. The simplest function simulator consists of files with "_simple" postfix. For more complex simulations, the simulator is constructed using files with different postfixes, but the overall framework does not change much.

Table 3-2 Directory and file structure

Directory	Sub-Directory	File Name
SimulationTop	SimulationMain	BasicMain
	Global	SystemSimConfiguration
		SLS
		SystemSim
	NetworkElement	SystemBS
		SystemMS

NetworkConfiguration	Network	Network
	NetworkBS	NetworkBS
	NetworkMS	NetworkMS
RadioResourceManagement	Scheduling	Scheduling
	SchedulingBS	SchedulingBS
	SchedulingMS	SchedulingMS
LinkPerformance	Performance	Performance
	PerformanceBS	PerformanceBS
	PerformanceMS	PerformanceMS
ChannelModel	Channel	Channel
	ChannelBS	ChannelBS
	ChannelMS	ChannelMS
Library	Math	
	Utility	
	Debug	Debug
	Miscellany	
Data	InputData	SimParamTop.txt
		SimParamNetwork.txt
		SimParamScheduling.txt
		SimParamPerformance.txt
		SimParamChannel.txt
	OutputData	PositionBS.txt
		PositionMS.txt
		Throughput.txt

3.5 Simulator variables

3.5.1 Simulator configurations

The simulator can have multiple files with the same function, and the files used can be selected by SystemSimConfiguration.h. The simulator can be simulated in different ways by the following methods:

- Modification of input variables or selection of input variable files
- Modification of main function or selection of main function
- Selection of file to be executed

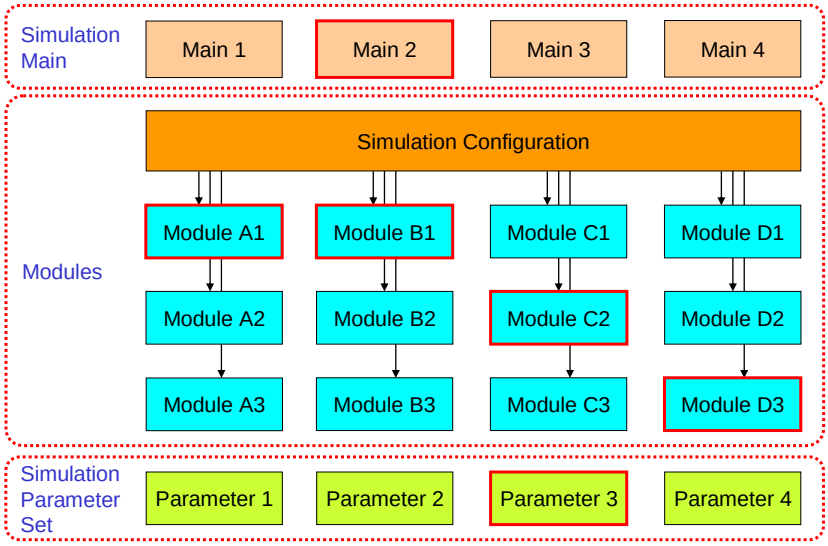

Figure 3-5. Choosing a simulator

3.5.2 Constant values

The global constants used in the simulator are mainly defined in "SLS.h". For example, the following global constants are defined.

```
#define SLS_MAX_MACRO_SITE                    19   // Max # of Macro sites
#define SLS_MAX_SECTOR_PER_MACRO_SITE 3    // Max # of sectors per Macro site
#define SLS_MAX_MS_PER_MACRO_BS          100  // Max # of MSs per Macro BS
...
```

These constants start with "SLS_" and are defined in "SLS.h" and are distinguished from similar constants defined elsewhere. Constants used in certain modules that are not used globally are defined in a specific module and also indicate where they are defined in advance so that they can be used without confusion even when constants with similar names are defined in multiple modules.

```
#define NETWORK_NUM_WRAPAROUND        7
```

3.5.3 Types

"SLS.h" defines types and enum types globally.

```
namespace SLS {
  enum BsType {
```

```
    MacroBS,
    MicroBS
  }; // Base station type

  enum Scenario {
    SimpleScenario,
    ...
  }; // Simulation scenario

  enum LinkDirection {
    Downlink,
    Uplink,
    ...
  }; // Simulation link direction
}
```

It also uses the namespace SLS to ensure that it knows exactly where it is defined, and is used in the form of SLS :: BsType. The types used in a specific module are defined in a specific module.

```
typedef struct {                    // Base Station Parameter
  int numSite;                      // # of Sites
  int numSector;            // # of sector(BS)s per Site
  int numBS;                        // # of BSs = numSector * numSite
  int numMsPerBS;                   // # of MSs per BS
  double interSiteDistance;         // (Minimum) inter site distance (m)
  double height;            // Antenna height (m)
  double minDistanceToBS;           // Minimum distance to BS (m)
} Network_BsParameter;
```

3.5.4 Global variables

The global variables used in the simulator are "Sim", which controls the simulation, and "BS" and "MS", which represent the network elements. BS and MS are an array of objects. The reason is that you have to set the maximum value of the array large enough to perform various types of simulations, but the array actually used may not be large depending on the simulator variables in the input file. Therefore, it is necessary to allocate memory when the simulation starts (or dynamically during the simulation).

```
extern SystemSim Sim;               // Simulation parameters and configurations
```

```
extern SystemBS* BS[SLS_MAX_BS];      // Base station (actually they are sectors or cells)
extern SystemMS* MS[SLS_MAX_MS];      // Mobile station
```

3.6 Skeleton Design

3.6.1 Simulator main

The skeleton of the simulator is as follows. main () is in "SimulationTop / SimulationMain".

```
int main()
{
  Sim.Initialize("Parameter_simple.txt");              // Simulation initialization
  Sim.InitilizeNode();                                 // Memory allocation for BSs and MSs
  for (int i = 0; i < Sim.numIteration; i++) {
    Sim.InitializeIteration(i);                        // Iteration initialization
    Sim.network->PlaceMacroBS();                       // Macro BS placement
    Sim.network->PlaceMsOnMacroBS();    // MS placement on Macro BS
    Sim.channel->CalculateLongTermChannel();       // Generate long term parameters
    for (int j = 0; j < Sim.numTTI; j++) {
      Sim.InitializeTTI(j);                        // TTI initialization
      Sim.channel->CalculateShortTermChannel(); // Generate short term parameters
      Sim.scheduling->Feedback();                  // MS feedback
      Sim.scheduling->Schedule();                  // Scheduling
      Sim.performance->Measure();                        // Throughput measurement
    }
    Sim.ConcludeIteration();                           // Iteration conclusion
  }
  Sim.ConcludeNode();                                  // Delete BSs and MSs
  Sim.Conclude();                                  // Simulation conclusion
  return 0;
}
```

If there is a Micro cell with a Macro cell, the main function can be modified as follows.

```
int main()
{
  Sim.Initialize("Parameter_simple.txt");              // Simulation initialization
```

```
  Sim.InitilizeNode();                              // Memory allocation for BSs and MSs
  for (int i = 0; i < Sim.numIteration; i++) {
    Sim.InitializeIteration(i);                     // Iteration initialization
    Sim.network->PlaceMacroBS();                    // Macro BS placement
    Sim.network->PlaceMicroBS();                    // Micro BS placement
    Sim.network->PlaceMsOnMacroBS();    // MS placement on Macro BS
    Sim.network->PlaceMsOnMicroBS();                // MS placement on Micro BS
    Sim.channel->CalculateLongTermChannel();        // Generate long term parameters
    for (int j = 0; j < Sim.numTTI; j++) {
      Sim.InitializeTTI(j);                 // TTI initialization
      Sim.channel->CalculateShortTermChannel(); // Generate short term parameters
      Sim.scheduling->Feedback();           // MS feedback
      Sim.scheduling->Schedule();           // Scheduling
      Sim.performance->Measure();                   // Throughput measurement
    }
    Sim.ConcludeIteration();                        // Iteration conclusion
  }
  Sim.ConcludeNode();                               // Delete BSs and MSs
  Sim.Conclude();                           // Simulation conclusion
  return 0;
}
```

3.6.2 Simulation Initialization

Simulation first reads the input file, sets experimental variables, allocates memory of BS and MS, and initializes variables.

```
SystemSim::InitializeNode()
{
  for (bs = 0; bs < network->numBS; bs++) {         // # of BSs
    BS[bs] = &BS(bs);                               // Create new BS
  }
  for (ms = 0; ms < netwok->numMS; ms++) {          // # of MSs
    MS[ms] = &MS(ms);                       // Create new MS
  }
}
```

3.6.3 Main operation

After the initialization of the simulation, the simulation is performed inside a loop. According to the network configuration, the BS and the MS are arranged and the performance is measured while scheduling and performing necessary operations according to the TTI.

In the simulator main, mainly the method of Sim is called to use the method of BS or MS again to perform necessary operation. For example, when Sim.network.PlaceMacroBS () is called, this method performs PlaceMacroBS () on each BS.

```
void Network::PlaceMacroBS()
{
  for (int bs = 0, int site = 0; site < macro.numSite; site++) {          // # of Macro Sites
    for (int sector = 0; sector < macro.numSector; sector++, bs++) {      // # of sectors per sites
      BS[bs]->network->PlaceMacroBS(bs, site, sector);                    // Place Macro BS
    }
  }
}
```

When Sim.network.PlaceMsOnMacroBS () is called, this method performs PlaceMsOnMacroBS () for each MS.

```
void Network::PlaceMsOnMacroBS()
{
  for (int ms = 0, int bs = 0; bs < macro.numBS; bs++) {          // # of Macro BS
    for (int i = 0; i < macro.numMsPerBS; i++, ms++) {            // # of MSs of Macro BS
      MS[ms]->network->PlaceMsOnMacroBS(ms, bs);                  // Place MS
      MS[ms]->network->Associate();                               // Perform association
    }
  }
}
```

When Sim.channel.CalculateShortTermChannel () is called, the channel value is calculated for each MS.

```
void Channel::CalculateShortTermChannel()
{
  for (int ms = 0; ms < Sim.network.numMS; ms++) {
    MS[ms]->channel->CalculateShortTermChannel(); // Calculate short term parameters
  }
```

```
}
```

When Sim.scheduling.Schedule () is executed, this method schedules for each BS.

```
void Scheduling::Schedule()
{
  for (int bs = 0; bs < Sim.network->numBS; bs++) {
    BS[bs]->scheduling->Schedule();
  }
}
```

3.6.4 End of Simulation

When the simulation loop finishes, free the memory, export the experiment results to a file, and end the experiment.

```
void SystemSim::ConcludeNode()
{
  for (bs = 0; bs < network->numBS; bs++) {          // # of BSs
    ~BS(bs);                                        // Delete BS
  }
  for (ms = 0; ms < network->numMS; ms++) {         // # of MSs
    ~MS(ms);                                        // Delete MS
  }
}
```

3.7 Simulator input and output

3.7.1 Simulator input

The input of the simulator is read in the form of a file. Input files are separated by module so that they can be modified according to the module.

- SimParamTop_simple.txt
- SimParamNetwork_simple.txt
- SimParamScheduling_simple.txt
- SimParamPerformance_simple.txt
- SimParamChannel_simple.txt

For example, assuming that the Micro BS is not used, the experimental parameters associated with the Micro BS are meaningless and all have a value of zero. The following table is an experimental variable except Micro BS.

Table 3-3 Directory and file structure

Modules	Parameters	Values	Description
Sim	numTTI	100	# of TTIs for simulation
	numIteration	10	# of iterations
	scenario	SimpleScenario	Scenario
	linkDirection	DownlinkOnly	Link direction
Network	numLayer	1	1: Single layer, 2: Two layers
	macro.numSite	19	# of Macro Sites
	macro.numSector	3	# of sectors per Macro site
	macro.numMsPerBS	10	# of MSs per Macro BS
	macro.interSiteDistance	200	Inter site distance for Macro BS (m)
	macro.height	25	Antenna height for Macro BS (m)
	macro.minDistanceToBS	30	Minimum distance to BS (m)
Scheduling	trafficModel	FullBuffer	Traffic model
	trafficLoad	1	Traffic load (0 ~ 1)
	dataSize	4000	Data size (Kbytes) for non-full-buffer
	algorithm	SimpleScheduling	Scheduling algorithm
	numRB	1	# of resource blocks for scheduling
	realisticFeedback	0	0: Ideal, 1: Realistic
Performance	realisticChannelEstimation	0	0: Ideal, 1: Realistic
Channel	macroBS.carrierFrequency	4	Macro BS carrier frequency (GHz)
	macroBS.bandwidth	20	Macro BS bandwidth (MHz)
	macroBS.txPower	44	Macro BS TX power (dBm)
	macroBS.noiseFigure	5	Macro BS noise figure (dB)
	macroBS.antennaGain	14	Macro BS antenna gain (dBi)
	macroBS.antenna.num	1	Macro BS # of antennas
	macroBS.model	SimpleChannel	Macro BS channel model
	macroMS.txPower	23	MS TX power (dBm)
	macroMS.noiseFigure	5	MS noise figure (dB)
	macroMS.antennaGain	0	MS Antenna gain (dBi)
	macroMS.antenna.num	1	MS # of antennas

3.7.2 Simulator output

Simulator output consists of BS and MS location, throughput result, and so on.

4. Expansion to 5G Simulators

This chapter describes the extension to the 5G eMBB / URLLC simulator using the simple version-based system level simulator described in Chapter 3. The biggest difference between 5G eMBB / URLLC simulator and simple SLS is the detail of each module. For example, dense urban, and indoor hotspot environment are considered, and 3D spatial channel mode standardized in 3GPP for channel specification, MIMO technology is considered. Chapter 4.1 and 4.2 gives simple description of implementation of 5G eMBB / URLLC simulator. From chapter 4.3 to 4.5 gives overall description of component of 5G eMBB / URLLC simulator.

4.1 5G eMBB simulator design

Figure 4-1 shows a complete module diagram of our 5G eMBB simulator.

In addition to the functions implemented for each module, the network module implements additional Dense urban, indoor hotspot, rural environment, and implemented the above 6GHz 3D SCM channel model in the channel module.

Furthermore, we implemented a hybrid beamforming structure for MIMO support in the above 6GHz channel in radio resource management, and implemented the 5G eMBB system level simulator to obtain the spectral efficiency of the system, which is a basic performance evaluation required by ITU-R.

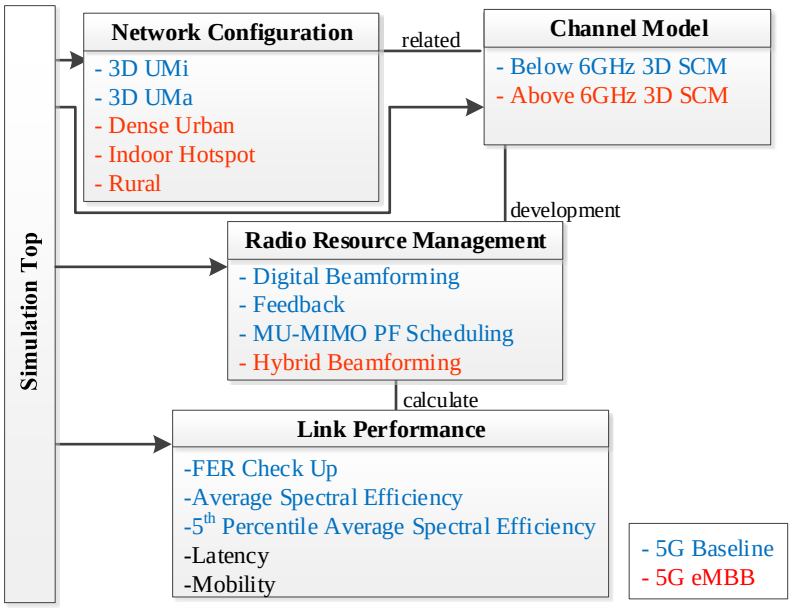

Figure 4-1. 5G eMBB Module Diagram

4.1.1 Simulator main

The 5G eMBB simulator design is as follows. main () is in "SimulationTop / SimulationMain".

```
int main()
{

        Sim.Initialize("NReMBB_IndoorOffice");  // NReMBB_UrbanMacroCell, NReMBB_DenseUrban, NReMBB_IndoorOffice, NReMBB_RuralMacroCell
        Sim.network->PlaceBS();  // BS placement

        for (int i = 0; i < Sim.numIteration; i++)
        {

            Sim.network->PlaceFullBufferMS();  // Full buffer MS placement

            //Sim.network->PlaceNonFullBufferMS();  // Non Full buffer MS placement
            Sim.network->PlaceWraparound();
            Sim.channel->LongTermChannel();

            for (Sim.TTI = 0; Sim.TTI < Sim.numTTI; Sim.TTI++)
            {

                Sim.channel->ShortTermChannel(); // DFT

                if (Sim.TTI % Sim.numTTIperFrame == 0) // Dynamic TDD scheduling
                {
                    Sim.scheduling->FrameStructure(SLS::DynamicTDD);
                }
                if (Sim.TTI % Sim.feedbackPeriod == 0)
                    Sim.scheduling->Feedback(); // Feeback
                Sim.scheduling->Schedule(); // Scheduling
                Sim.scheduling->ReceivedSINRCalculation(); // Scheduling
                Sim.performance->FERCheckup(); // FER Checkup
                Sim.performance->HARQInformation(); // HARQ Information update
                Sim.performance->Measurement(); // Throughput measurement

            }
        }

        Sim.Demonstration();
```

```
		Sim.Conclude();


		return 0;


}
```

4.1.2 Simulation Initialization

Simulation first reads the input file, sets experimental variables, allocates memory of BS and MS, and initializes variables.

```
void SystemSim::Initialize(string fileName)
{
	// Random seed
	arma::arma_rng::set_seed_random();


	// Open the file
	ifstream inFile("../../Simulator/Data/Input Data/SimulationParameter/SystemSimParameter/SimParamTop_" + fileName + ".txt",ios::out);
	char inputString[100];
	string* result;


	if (inFile.is_open())
	{
		while (!inFile.eof())
		{


			inFile.getline(inputString, 100);
			result = strSplit(inputString, "	");
			if (result[0] == "numTTI") numTTI = stoi(result[1]);
			else if (result[0] == "numerologyParameter") numerologyParameter = stoi(result[1]);
			else if (result[0] == "numSymbolperTTI") numSymbolperTTI = stoi(result[1]);
			else if (result[0] == "numIteration") numIteration = stoi(result[1]);
			else if (result[0] == "feedbackPeriod") feedbackPeriod = stoi(result[1]);
			else if (result[0] == "scenario") scenario = SLS::Scenario(stoi(result[1]));
			else if (result[0] == "linkDirection") linkDirection = SLS::LinkDirection(stoi(result[1]));


		}
```

```
        }
        numTTIperFrame = pow(2.0, numerologyParameter) * 10;
        subcarrierSpacing = pow(2.0, numerologyParameter) * 15;

        network = new Network();
        scheduling = new Scheduling();
        performance = new Performance();
        channel = new Channel();



        network->Initialize(fileName); // Parameter initialization for network configuration
        channel->Initialize(fileName); // Parameter initialization for channel
        scheduling->Initialize(fileName); // Parameter initialization for radio resource management
        performance->Initialize(fileName); // Parameter initialization for link performance

        Sim.RateTTI.zeros(Sim.network->numMS,Sim.numTTI);
        Sim.LatencyTTI.zeros(Sim.network->numMS, Sim.numTTI);
        Sim.BufferTTI.zeros(Sim.network->numMS, Sim.numTTI);

}
```

4.1.3 Main operation

After the initialization of the simulation, the simulation is performed inside the loop. According to the network configuration, the base station and the UE are arranged and the performance is measured while scheduling and performing necessary operations according to the TTI.

In the simulator main, mainly the method of Sim is called, but these methods call the method of BS or MS again to perform necessary operation. For example, when Sim.network.PlaceMacroBS () is called, this method performs PlaceMacroBS () on each BS.

```
void Network::PlaceMacroBS()
{
  for (int bs = 0, int site = 0; site < macro.numSite; site++) {              // # of Macro Sites
    for (int sector = 0; sector < macro.numSector; sector++, bs++) {      // # of sectors per sites
void Network::PlaceBS()
{
        numBS = 0;
```

```
int bsID = 0;
if (Sim.network->NetworkModel == NETWORK::IndoorOffice)
{
	for (int i = 0; i < numMacroBS; i++) // # of Indoor Sites
	{
		for (int j = 0; j < numSector; j++) // # of Indoor Sectors
		{
			BS[bsID] = new SystemBS(bsID, i, j, SLS::IndoorBS);
			bsID++;
		}
	}
}
else // hexagonal greed
{
	for (int i = 0; i < numMacroBS; i++) // # of Macro Sites
	{
		for (int j = 0; j < numSector; j++) // # of Macro Sectors
		{
			BS[bsID] = new SystemBS(bsID, i, j, SLS::MacroBS);
			bsID++;
			if (numMicroBS != 0)
			{
				for (int k = 0; k < numMicroBS; k++) // # of Micro Sites
				{
					for (int l = 0; l < numSector; l++) // # of Micro Sectors
					{
						BS[bsID] = new SystemBS(bsID, k, l, SLS::MicroBS);
						bsID++;
					}
				}
			}
		}
	}
}
numBS = bsID;
numSite = numBS / numSector;
```

```
	setcolor(10, 0);
	cout << "[Network  30%]: Base Stations placement" << endl;
}

void Network::PlaceFullBufferMS()
{
	for (int msID = 0, i = 0; i < numBS; i++) // # of Macro BS
	{
		BS[i]->network->numAttachedMS = 0;
		for (int j = 0; j < numMsPerSector; j++, msID++) // # of MSs of Macro BS
		{
			BS[i]->network->numAttachedMS++;
			MS[msID] = new SystemMS(msID, i); // Create a new MS and perform initialization
		}
	}
	cout << "[Network  70%]: Full Buffer Mobile Station placement" << endl;
}

void Scheduling::Feedback()
{
	for (int msID = 0; msID < Sim.network->numMS; msID++)
	{
		MS[msID]->scheduling->MuMimoFeedback(msID, 0); //MU-MIMO Feedback(msID)
	}
	//for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
	//{
	//	BS[bsID]->scheduling->MuMimoFeedback(bsID);
	//}
	setcolor(13, 0);
	cout << "[Scheduling]: Mobile Station CSI feedback" << endl;
}

void Scheduling::Schedule()
{
	for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
	{
```

```
                BS[bsID]->scheduling->MuMimoSchedule(bsID, 0, BS[bsID]->scheduling->scheduledMS,
BS[bsID]->scheduling->framestructure(Sim.TTI % Sim.numTTIperFrame));
        }
        setcolor(13, 0);
        cout << "[Scheduling]: Base Station scheduling " << endl;
}
```

4.1.4 End of Simulation

When the simulation loop finishes, free the memory, export the experiment results to a file, and end the experiment.

```
void SystemSim::Conclude ()
{
  for (bs = 0; bs < network->numBS; bs++) {          // # of BSs
    ~BS(bs);                                        // Delete BS
  }
  for (ms = 0; ms < network->numMS; ms++) {         // # of MSs
    ~MS(ms);                                        // Delete MS
  }
}
```

4.2 5G URLLC simulator design

To implement URLLC simulator efficiently, we reconfigure and modify the existing eMBB system level simulator of 5G K-SimSys. As in Fig. 4-2 we modify the existing simulator described in Fig. 4-1. The edited / changed functions in the module are highlighted in red in the figure.

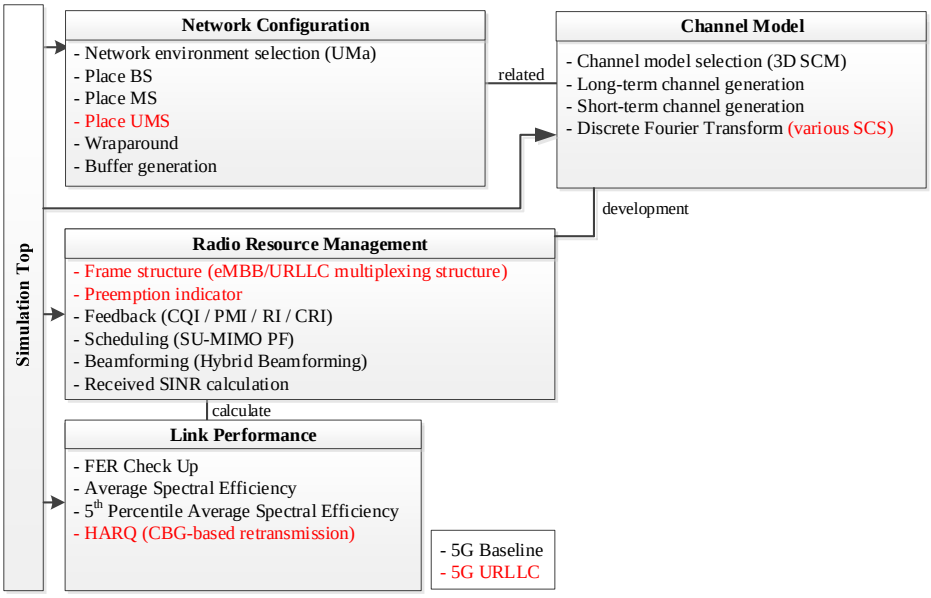

Figure 4-2. 5G URLLC Module Diagram

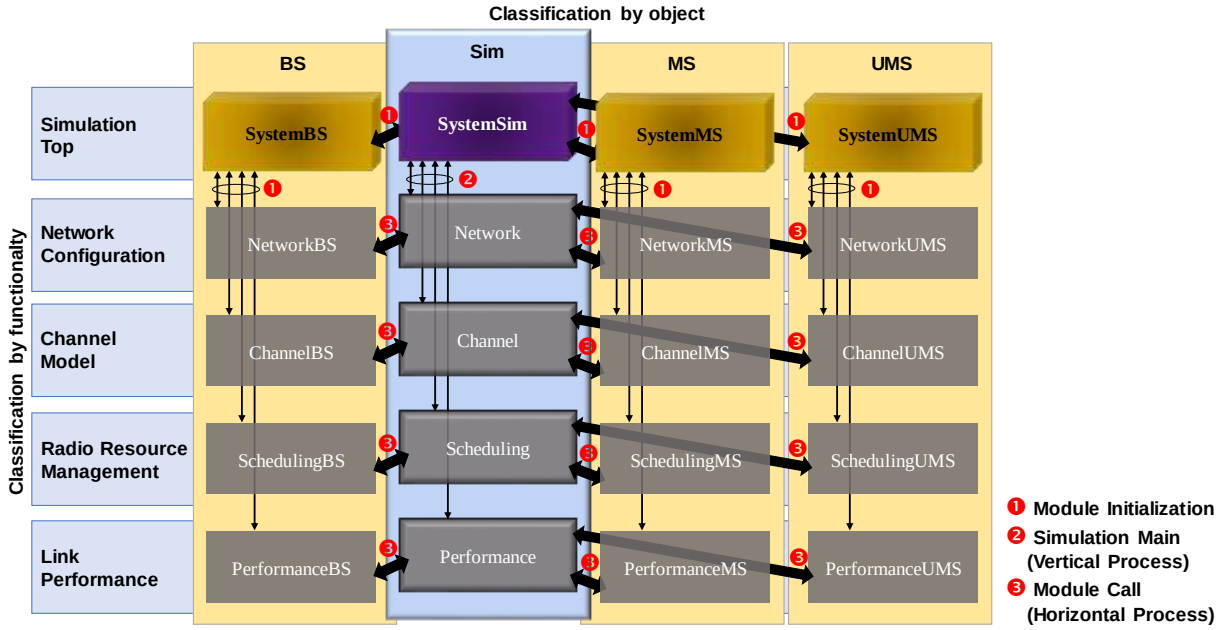

Figure 4-3. 5G URLLC Module Diagram

For the simulator module, since the URLLC simulator need one more separate class of UE to the eMBB simulator, we added new class of MS name UMS for URLLC mobile stations. By defining two MS classes, the eMBB and URLLC UE can simultaneously configured in one simulator. Next, by changing the parameter, the sub-carrier spacing of the OFDM symbol can be modified flexibly. The sub-carrier spacing in the simulator have one unique value for each simulator, e.g. 30 KHz.

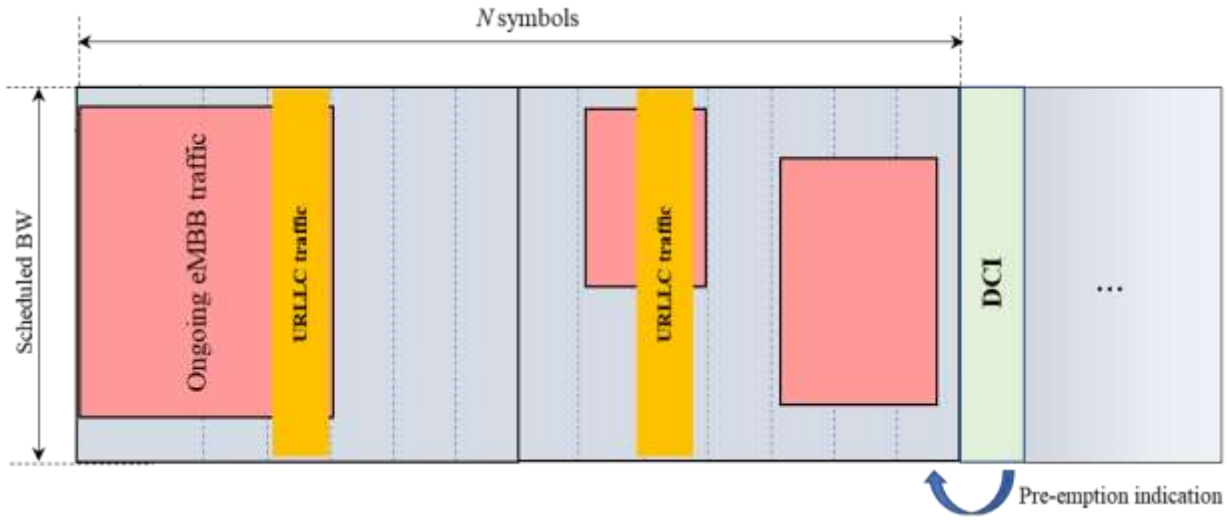

Figure 4-4. Example: Pre-emption based eMBB/URLLC Multiplexing

For radio resource management module we implement pre-emption based eMBB / URLLC multiplexing as depicted in Fig 4-4. In 3GPP specification, URLLC traffic are scheduled to on-going eMBB traffic by puncturing (pre-emption). To inform to the MSs about pre-emption, 3GPP defines downlink control information such as DCI 1_1 or DCI 2_1 with pre-emption indicator and pre-emption bit-map. By knowing the pre-emption information eMBB / URLLC MSs can reflect the information to the process of HARQ combining. In our simulator, for simplicity, we assume ideal pre-emption indication to the MSs. Therefore eMBB / URLLC are able to know their scheduled information and/or pre-emption information and combine correctly in the HARQ buffer.

4.2.1 Simulator main

The 5G URLLC simulator design is as follows. main () is in "SimulationTop / SimulationMain".

```
int main()
{
	Sim.Initialize("NRuRLLC_UrbanMacroCell"); // Simulation initialization
	Sim.network->PlaceMacroBS(); // Macro BS placement
	for (int i = 0; i < Sim.numIteration; i++)
	{
		Sim.network->PlaceEMBBMS(); // eMBB MS placement (Full buffer)
		Sim.network->PlaceURLLCMS(); // uRLLC MS placement (non-Full buffer)
		Sim.network->PlaceWraparound();
		Sim.channel->LongTermChannel();

		cout << "********************Simulation Start********************" << endl;
```

```
		for (Sim.TTI = 0; Sim.TTI < Sim.numTTI; Sim.TTI++)
		{
			Sim.scheduling->BufferUpdate(); // System buffer update
			if (Sim.TTI == 0 || Sim.TTI % Sim.feedbackPeriod == 0)
			{
				Sim.scheduling->Feedback(); // MS feedback
			}

			Sim.scheduling->Schedule(); // Scheduling
			Sim.performance->Measure(); // Throughput measurement
			cout << "******TTI " << Sim.TTI << "******" << endl;
			Sim.Demonstration();
		}
		Sim.ConcludeIteration(); // Iteration conclusion
	}

	Sim.Conclude(); // Simulation conclusion
	return 0;
}
```

4.2.2 Simulation Initialization

Simulation first reads the input file, sets experimental variables, allocates memory of BS, MS and UMS, and initializes variables.

```
void SystemSim::Initialize(string fileName)
{
	// Random seed
	arma::arma_rng::set_seed_random();

	// Open the file
	ifstream inFile("../../Simulator/Data/Input Data/SimulationParameter/SystemSimParameter/SimParamTop_" + fileName + ".txt",ios::out);
	char inputString[100];
	string* result;
```

```
	if (inFile.is_open())
	{
		while (!inFile.eof())
		{

			inFile.getline(inputString, 100);
			result = strSplit(inputString, "	");
			if (result[0] == "numTTI") numTTI = stoi(result[1]);
			else if (result[0] == "numerologyParameter") numerologyParameter = stoi(result[1]);
			else if (result[0] == "numSymbolperTTI") numSymbolperTTI = stoi(result[1]);
			else if (result[0] == "numIteration") numIteration = stoi(result[1]);
			else if (result[0] == "feedbackPeriod") feedbackPeriod = stoi(result[1]);
			else if (result[0] == "scenario") scenario = SLS::Scenario(stoi(result[1]));
			else if (result[0] == "linkDirection") linkDirection = SLS::LinkDirection(stoi(result[1]));

		}
	}
	numTTIperFrame = pow(2.0, numerologyParameter) * 10;
	subcarrierSpacing = pow(2.0, numerologyParameter) * 15;

	network = new Network();
	scheduling = new Scheduling();
	performance = new Performance();
	channel = new Channel();


	network->Initialize(fileName); // Parameter initialization for network configuration
	channel->Initialize(fileName); // Parameter initialization for channel
	scheduling->Initialize(fileName); // Parameter initialization for radio resource management
	performance->Initialize(fileName); // Parameter initialization for link performance

	Sim.RateTTI.zeros(Sim.network->numMS,Sim.numTTI);
	Sim.LatencyTTI.zeros(Sim.network->numMS, Sim.numTTI);
	Sim.BufferTTI.zeros(Sim.network->numMS, Sim.numTTI);

}
```

4.2.3 Main operation

After the initialization of the simulation, the simulation is performed inside the loop. According to the network configuration, the base station and the UE are arranged and the performance is measured while scheduling and performing necessary operations according to the TTI.

In the simulator main, mainly the method of Sim is called, but these methods call the method of BS or MS / UMS again to perform necessary operation. For example, when Sim.network.PlaceMacroBS () is called, this method performs PlaceMacroBS () on each BS.

```
void NetworkURLLCMS::Initialize(int umsID, int bsID)
{

	this->id = umsID; // URLLC MS ID
	this->attachedBS = bsID; // Temporary association
	this->pos3D;
	this->location;
	this->interArrivalTime.zeros(Sim.numTTI);

	// MS Placement
	if (Sim.network->NetworkModel == NETWORK::UrbanMacroCell){
		this->PlaceRandomHexagonal();
	}

	if (Sim.network->bufferModel == RRM::FullBuffer) {
		// Buffer Size
	}
	else if (Sim.network->bufferModel == RRM::NonFullBuffer) {
		for (int tti = 0; tti < Sim.numTTI; tti++) {
			this->interArrivalTime(tti) = ceil(-(1 / Sim.network->meanArrivalTime)*log(1 - arma::randu()) * 10 / 5);
		}
		this->bufferTime = 0;
		this->msBuffer = 0;
		this->arrivalTime = 0;
	}

}
```

4.2.4 End of Simulation

When the simulation loop finishes, free the memory, export the experiment results to a file, and end the experiment.

```
void SystemSim::Conclude ()
{
  for (bs = 0; bs < network->numBS; bs++) {          // # of BSs
    ~BS(bs);                                         // Delete BS
  }
  for (ms = 0; ms < network->numMS; ms++) {          // # of MSs
    ~MS(ms);                                         // Delete MS
  }
 for (ums = 0; ums < network->numUMS; ums++) {  // # of UMSs
    ~UMS(ums);                                       // Delete UMS
  }

}
```

4.3 Network configuration

In 5G eMBB, we consider the dense urban and indoor-hotspot environments in addition to the existing hexagonal grid-based network structure.

Dense urban environment is considered in 5G for performance evaluation in two layer situations where both Macro / Micro layer is used. Dense urban environment deploys base stations according to the 19 cell structure of wraparound based UMa environment and deploys 3 ~ 9 Micro TRPs according to ISD between Micro TRP in each sector (hereinafter referred to as Macro TRP).

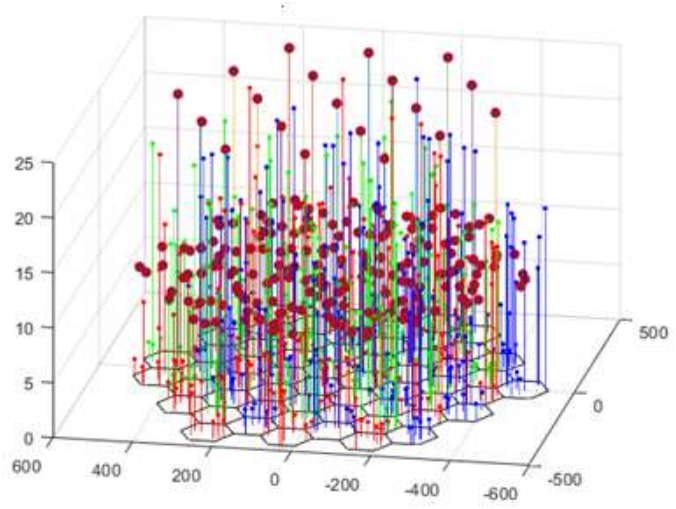

Figure 4-5 Network Configuration of Dense Urban

Also, the Indoor-open office environment has six sectors with a ISD of 20m and a hBS of 3m in a 50m x 120m space and has a layout with three sectors. Unlike wrap-around structures are not considered.

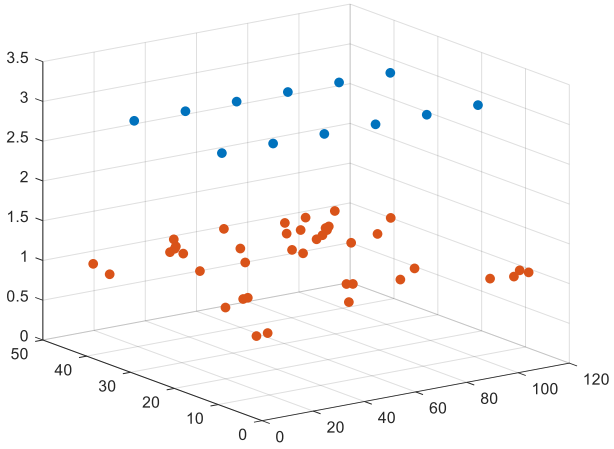

Figure 4-6 Network Configuration of Indoor Hotspot

The initialize function to select multiple network structures is shown below.

```
void NetworkMS::Initialize(int msID, int bsID)
{

        this->id = msID; // MS ID
        this->attachedBS = bsID; // Temporary association
        this->pos3D;
        this->location;
        this->msBuffer = 0;
        this->arrivalTime = 1;
        this->interArrivalTime.zeros(Sim.numTTI);
```

```
	// MS Placement
	if (Sim.network->NetworkModel == NETWORK::UrbanMacroCell){
		this->PlaceRandomHexagonal();
	}
	else if (Sim.network->NetworkModel == NETWORK::DenseUrban) {
		this->PlaceRandomHexagonal();
		this->PlaceRandomCircular();
	}
	else if (Sim.network->NetworkModel == NETWORK::IndoorOffice) {
		this->PlaceRandomRectangular();
	}
	else if (Sim.network->NetworkModel == NETWORK::RuralMacroCell) {
		this->PlaceRandomHexagonal();
	}

	if (Sim.network->bufferModel == RRM::FullBuffer) {
		// Buffer Size
	}
	else if (Sim.network->bufferModel == RRM::NonFullBuffer) {
		for (int tti = 0; tti < Sim.numTTI; tti++) {
			this->interArrivalTime(tti) = ceil(-(1 / Sim.network->meanArrivalTime)*log(1 -
arma::randu()) * 1000 / 5) * 5;
		}
	}
}
```

4.4 Channel

First, the simulation environment and the antenna model such as UMi / UMa / indoor-open office are determined, and the channel between the BS and the MS are generated through the procedure shown in Fig 4-6. Channel modeling for millimeter waves is as shown in Fig 4-6, with generation of general parameter generation, small scale parameter generation, coefficient generation, and additional features.

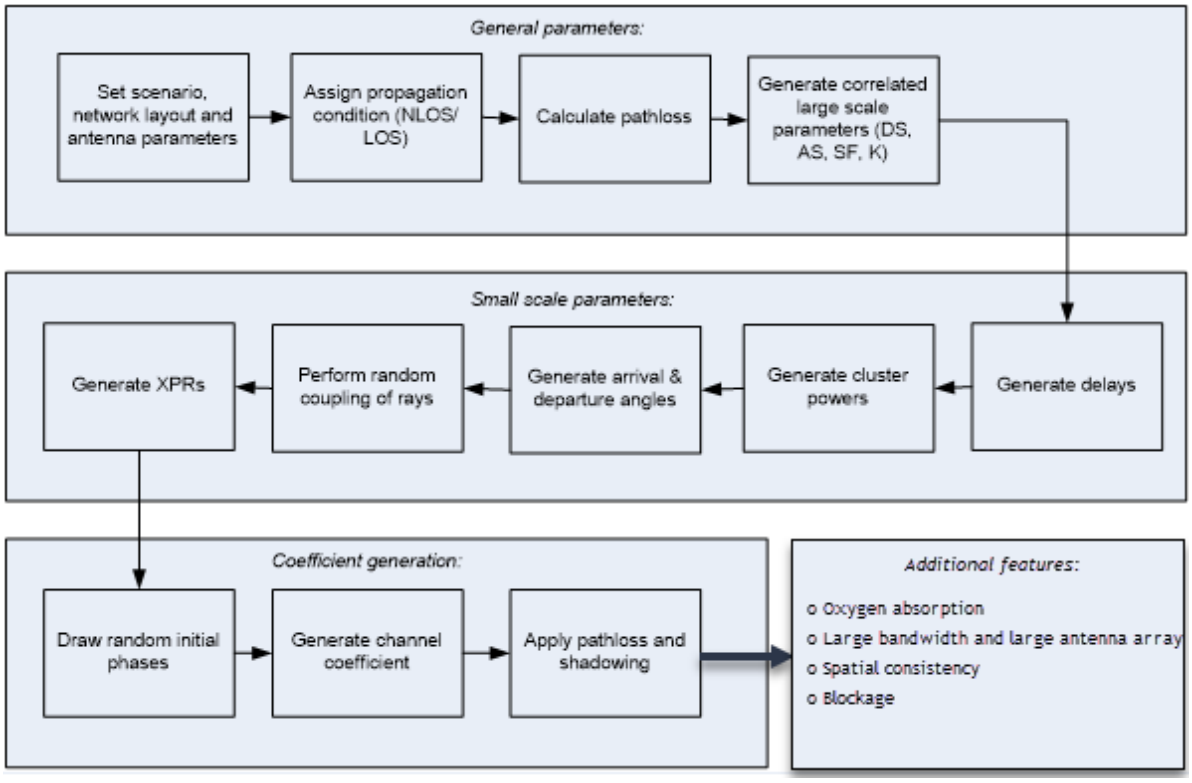

Figure 4-7 Channel generation process

$$H_{u,s,n}(t) = \sqrt{P_n / M} \sum_{m=1}^{M} \begin{bmatrix} F_{rx,u,\theta}\left(\theta_{n,m,ZOA}, \varphi_{n,m,AOA}\right) \\ F_{rx,u,\varphi}\left(\theta_{n,m,ZOA}, \varphi_{n,m,AOA}\right) \end{bmatrix}^T \begin{bmatrix} \exp\left(j\Phi_{n,m}^{\theta\theta}\right) & \sqrt{\kappa_{n,m}^{-1}} \exp\left(j\Phi_{n,m}^{\theta\varphi}\right) \\ \sqrt{\kappa_{n,m}^{-1}} \exp\left(j\Phi_{n,m}^{\varphi\theta}\right) & \exp\left(j\Phi_{n,m}^{\varphi\varphi}\right) \end{bmatrix}$$
$$\begin{bmatrix} F_{tx,s,\theta}\left(\theta_{n,m,ZOD}, \varphi_{n,m,AOD}\right) \\ F_{tx,s,\varphi}\left(\theta_{n,m,ZOD}, \varphi_{n,m,AOD}\right) \end{bmatrix} \exp\left(j2\pi\lambda_0^{-1}\left(\hat{r}_{rx,n,m}^T . \bar{d}_{rx,u}\right)\right) \exp\left(j2\pi\lambda_0^{-1}\left(\hat{r}_{tx,n,m}^T . \bar{d}_{tx,s}\right)\right) \exp\left(j2\pi v_{n,m} t\right)$$

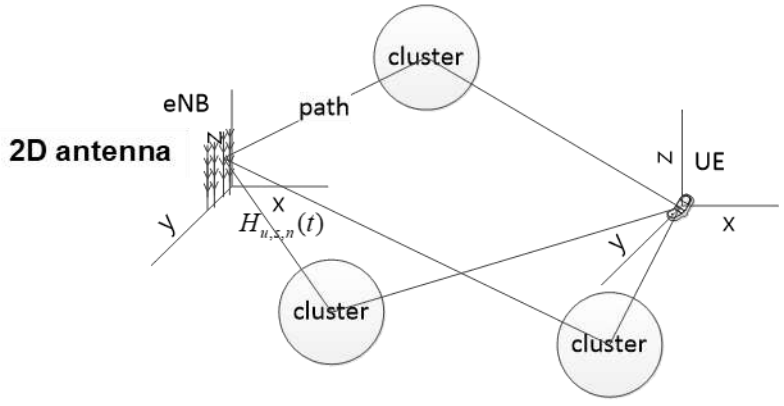

Figure 4-8 3D spatial channel model

It is possible to generate a long term channel using the procedure shown in Fig. 4-7, and a short term channel according to time according to the moving speed of the terminal can be generated through the long term channel as follows.

```
void Channel::LongTermChannel()
{

	for (int msID=0; msID<Sim.network->numMS; msID++)
	{
		TemporaryGlobalVariableInitialize(msID);
		MS[msID]->channel->LongTermChannel(msID);

		setcolor(11, 0);
		double longTermChannelProcess;
		longTermChannelProcess = 95 * msID / (Sim.network->numMS);
		cout << "[Channel " << longTermChannelProcess << "%]: Long Term Channel of MS ID " << msID << " Calculated" << endl;
	}
}

void Channel::ShortTermChannel()
{
	for (int msID = 0; msID < Sim.network->numMS; msID++)
	{
		MS[msID]->channel->ShortTermChannel(msID);
	}
		setcolor(11, 0);
		cout << "[Channel]: Short Term Channel of MS(DFT) calculated" << endl;
}
```

4.5 Scheduling

For 5G scenarios, various MIMO technologies such as hybrid beamforming and beam management are considered. Each scheduling process consists of feedback, schedule, and received SINR calculations.

```
void Scheduling::Feedback()
{
```

```
	for (int msID = 0; msID < Sim.network->numMS; msID++)
	{
		//MS[msID]->scheduling->Feedback(msID); //Feedback(msID)
		MS[msID]->scheduling->MuMimoFeedback(msID); //MU-MIMO Feedback(msID)
	}
	for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
	{
		BS[bsID]->scheduling->MuMimoFeedback(bsID);
	}
	setcolor(13, 0);
	cout << "[Scheduling]: Mobile Station CSI feedback" << endl;
}

void Scheduling::Schedule()
{
	for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
	{
		for (int subband = 0; subband < Sim.numSubband; subband++)
		{
			for (int time = 0; time < Sim.numTTIperFrame; time++)
			{
				BS[bsID]->scheduling->MuMimoSchedule(bsID, subband, BS[bsID]->scheduling->scheduledMS, BS[bsID]->scheduling->framestructure(time));
			}
		}
	}
	cout << "[Scheduling]: Base Station scheduling " << endl;
}

void Scheduling::ReceivedSINRCalculation()
{
	for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
	{
		for (int time = 0; time < Sim.numTTIperFrame; time++)
		{
			if (BS[bsID]->scheduling->framestructure(time) == 0)
			{
```

```
                                        for (int msID = 0; msID < BS[bsID]->scheduling->scheduledMS; msID++)
                                                MS[msID]->scheduling->ReceivedSINR(BS[bsID]->scheduling->scheduledMS, BS[bsID]->scheduling->framestructure(time), time);
                                }
                                else
                                {
                                        for (int msID = 0; msID < BS[bsID]->scheduling->scheduledMS; msID++)
                                                BS[msID]->scheduling->ReceivedSINR(BS[bsID]->scheduling->scheduledMS, BS[bsID]->scheduling->framestructure(time), time);
                                }
                        }
                }

        cout << "[Scheduling]: Received SINR calculation " << endl;
}
```

In the feedback function, analog digital precoding for beamforming is determined and it can be set freely according to the scheduling technique. The user can modify this part to take advantage of the performance gain of the proposed technique.

```
analogCodebook = PrecodingMatrix(RRM::Analogbeamforming, analogCodebook);
digitalCodebook = PrecodingMatrix(RRM::Digitalbeamforming, digitalCodebook);
```

4.6 Simulator input and output

4.6.1 Simulator input

The input of the simulator is read in the form of a file. Input files are separated by module so that they can be modified according to the module. For example, in eMBB simulator, the input files are:

- SimParamTop_NReMBB.txt
- SimParamNetwork_ NReMBB.txt
- SimParamScheduling_ NReMBB.txt
- SimParamPerformance_ NReMBB.txt
- SimParamChannel_ NReMBB.txt

For example, assuming that the Micro BS is not used, the experimental parameters associated with the Micro BS are meaningless and all have a value of zero. The following table is an experimental variable except Micro BS.

Table 4-1 Input variables: Example

Modules	Parameters	Values	Description
Sim	numTTI	100	# of TTIs for simulation
	numIteration	10	# of iterations
	scenario	SimpleScenario	Scenario
	linkDirection	DownlinkOnly	Link direction
Network	numLayer	1	1: Single layer, 2: Two layers
	macro.numSite	19	# of Macro Sites
	macro.numSector	3	# of sectors per Macro site
	macro.numMsPerBS	10	# of MSs per Macro BS
	macro.interSiteDistance	200	Inter site distance for Macro BS (m)
	macro.height	25	Antenna height for Macro BS (m)
	macro.minDistanceToBS	30	Minimum distance to BS (m)
Scheduling	trafficModel	FullBuffer	Traffic model
	trafficLoad	1	Traffic load (0 ~ 1)
	dataSize	4000	Data size (Kbytes) for non-full-buffer
	algorithm	SimpleScheduling	Scheduling algorithm
	numRB	1	# of resource blocks for scheduling
	realisticFeedback	0	0: Ideal, 1: Realistic
Performance	realisticChannelEstimation	0	0: Ideal, 1: Realistic
Channel	macroBS.carrierFrequency	4	Macro BS carrier frequency (GHz)
	macroBS.bandwidth	20	Macro BS bandwidth (MHz)
	macroBS.txPower	44	Macro BS TX power (dBm)
	macroBS.noiseFigure	5	Macro BS noise figure (dB)
	macroBS.antennaGain	14	Macro BS antenna gain (dBi)
	macroBS.antenna.num	1	Macro BS # of antennas
	macroBS.model	SimpleChannel	Macro BS channel model
	macroMS.txPower	23	MS TX power (dBm)
	macroMS.noiseFigure	5	MS noise figure (dB)
	macroMS.antennaGain	0	MS Antenna gain (dBi)
	macroMS.antenna.num	1	MS # of antennas

4.6.2 Simulator output

Simulator output consists of BS and MS location, throughput result, and so on.

5. How to Use

We will use simple versions of the OMF structure for SLS training purposes and aim to expand to 5G scenarios. If you understand the whole module structure and simulator flow through the simple version, you will be able to configure the simulator flexibly according to the scenario you want. As we saw in the previous announcement, our

simulator is modular, and these are library-structured in files. The module is divided into simulation top, network configuration, channel model, RRM, and link performance. It consists of header and cpp file.

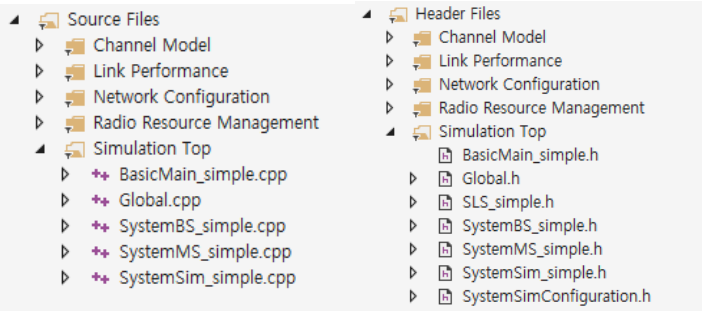

Figure 5-1 Library-structure

First, enter SystemSimConfiguration.h to see all cpp and header files according to the simulator scenario. They have the ability to combine the necessary cpp and header freely according to the user's purpose. We have divided them into a simple version for educational purposes, NR eMBB / URLLC, and NRLAA to be added later.

```
#ifndef CONFIG_H
#define CONFIG_H


/*---------------------------------------------------------------------------*/
/*                                                                           */
/*                        SIMULATION TYPE DECLARATION                        */
/*                                                                           */
/*---------------------------------------------------------------------------*/

//#define SYSTEM_SIM_CONFIG_LINK // Link level simulation
#define SYSTEM_SIM_CONFIG_SIMPLE // Simple simulation
//#define SYSTEM_SIM_CONFIG_FDMIMO // FDMIMO simulation
//#define SYSTEM_SIM_CONFIG_NReMBB// NReMBB simulation
//#define SYSTEM_SIM_CONFIG_NRLAA // NRLAA simulation


/*---------------------------------------------------------------------------*/
/*                                                                           */
/*                        INCLUDE FILES FOR MODULE                           */
/*                                                                           */
/*---------------------------------------------------------------------------*/

#ifdef SYSTEM_SIM_CONFIG_SIMPLE

#include "../../Simulation Top/Global/SLS_simple.h"
#include "../../Simulation Top/Global/Global.h"
#include "../../Channel Model/Antenna/Antenna_simple.h"
#include "../../Channel Model/LargeScale Channel/LargeScaleChannel_simple.h"
#include "../../Channel Model/Channel/Channel_simple.h"
#include "../../Channel Model/ChannelMS/ChannelMS_simple.h"
#include "../../Channel Model/ChannelBS/ChannelBS_simple.h"
#include "../../Network Configuration/Network/Network_simple.h"
#include "../../Network Configuration/NetworkMS/NetworkMS_simple.h"
#include "../../Network Configuration/NetworkBS/NetworkBS_simple.h"
#include "../../Radio Resource Management/Scheduling/Scheduling_simple.h"
#include "../../Radio Resource Management/SchedulingMS/SchedulingMS_simple.h"
#include "../../Radio Resource Management/SchedulingBS/SchedulingBS_simple.h"
```

Figure 5-2 Library-structure

If you look at the Simulation Top section, you will see a file called main function contains the actual flow of this simulator, including BS and MS placement, channel creation, scheduling, and performance verification.

```
int main()
{
    Sim.Initialize("simple"); // Simulation Initialization
    Sim.network->PlaceMacroBS(); // Macro BS placement
    cout << "Placed Macro BS" << endl;
    for (int i = 0; i < Sim.numIteration; i++)
    {
        Sim.network->PlaceMicroBS(); // Micro BS placement
        cout << "Placed Micro BS" << endl;
        Sim.network->PlaceFullBufferMS(); // Full buffer MS placement
        cout << "Placed Full Buffer MS" << endl;
        Sim.channel->LongTermChannel();
        cout << "Long Term Channel Generated" << endl;

        for (int j = 0; j < Sim.numTTI; j++)
        {
            //Sim.network->MoveMS(); // for mobility
            //cout << "MS Moved" << endl;
            Sim.network->PlaceNonFullBufferMS(); // Non-full buffer MS placement
            cout << "placed Non Full Buffer MS" << endl;
            Sim.channel->ShortTermChannel();
            cout << "Short Term Channel Generated" << endl;
            Sim.scheduling->Feedback(); // MS feedback
            cout << "Feedback Success" << endl;
            //cout << BS[0]->network->id;
            Sim.scheduling->Schedule(); // Scheduling
            cout << "Scheduling Success" << endl;
            Sim.performance->Measure(); // Throughput measurement
            cout << "Measure Success" << endl;
        }
        //Sim.ConcludeIteration(); // Iteration conclusion
    }
    Sim.Conclude(); // Simulation conclusion
    return 0;
}
```

Figure 5-3 Main function in Simple scenario

Looking closely, it is divided into simulation initialization, main operation, and termination. In case of initialization, it receives global memory initialization and variable inputs required for other modules belonging to Sim class as well as simulation Top.

```
void SystemSim::Initialize(string fileName)
{
    // Open the file
    ifstream inFile("../Simulator/Data/Input Data/SimulationParameter/SystemSimParameter/SimParamTop_" + fileName + ".txt");
    char inputString[100];
    string* result;

    if (inFile.is_open())
    {
        while (!inFile.eof()) {
            inFile.getline(inputString, 100);
            result = strSplit(inputString, "    ");
            if(result[0] == "numTTI") numTTI = stoi(result[1]);
            else if (result[0] == "numIteration") numIteration = stoi(result[1]);
            else if (result[0] == "scenario") scenario = SLS::Scenario(stoi(result[1]));
            else if (result[0] == "linkDirection") linkDirection = SLS::LinkDirection(stoi(result[1]));

        }
    }

    network = new Network();
    scheduling = new Scheduling();
    performance = new Performance();
    channel = new Channel();

    network->Initialize(fileName); // Parameter initialization for network configuration
    scheduling->Initialize(fileName); // Parameter initialization for radio resource management
    performance->Initialize(fileName); // Parameter initialization for link performance
    channel->Initialize(fileName); // Parameter initialization for channel

}
```

Figure 5-4 Initialize function in SystemSim

So, if you go into this part, the simulator scenario is simple version, so you input the input file name as naming and input the simple text to read the necessary variables. (See Simulator -> Data -> Input Data -> SimulationParameter -> SystemSimParameter)

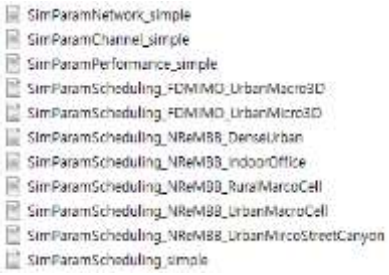

Figure 5-5 Input data: Example

When you open the actual text file, you can enter the input variables and their values as shown below. Depending on the scenario, various input variables exist as follows. After this initialization, you are now taken to the Network configuration module.

```
void NetworkBS::Initialize(int bs, int site, int sector, SLS::BsType bsType)
{
    this->networkModel = Sim.network->NetworkModel; // Network model information
    this->bsType = SLS::BsType(bsType);             // BS type
    this->id = bs;                                  // BS ID
    this->site = site;                              // Site ID
    this->sector = sector;                          // Sector number (0, 1, 2)
    this->pos3D;
    this->numAttachedMS = 0;

    // BS position
    if (Sim.network->NetworkModel == NETWORK::UrbanMacroCell)
    {
        switch (bsType)
        {
            case SLS::MacroBS: PlaceHexagonal(); break;
            case SLS::MicroBS: PlaceHexagonal(); break;
        }
    }
    else if (Sim.network->NetworkModel == NETWORK::DenseUrban)
    {
        PlaceHexagonal();
        PlaceCircular();
    }
    else if (Sim.network->NetworkModel == NETWORK::IndoorOffice)
    {
        PlaceRectangular();
    }
    else if (Sim.network->NetworkModel == NETWORK::RuralMacroCell)
    {
        switch (bsType)
        {
            case SLS::MacroBS: PlaceHexagonal(); break;
            case SLS::MicroBS: PlaceHexagonal(); break;
        }
    }

}
```

Figure 5-6 Initialize various network configuration in NetworkBS

This is where the actual BSs and MSs are located. Different networks are configured according to the scenario. Typical uses are 19 cell structure based on hexagonal grid, dense urban with small cell for 5G, indoor for indoor use. In this simple version, we considered 19 cell structure based on hexagonal grid. When the base station and the terminal are placed in the network module, the BS and MS class modules are initialized. Therefore, the arrangement of the base station and the terminal is detailed as shown below. In the case of the 5G simulator, it is subdivided according to the new network structure and can be selected as needed.

When you get back to main, there is also a buffer control among the roles that the network module does. That is, according to the scenario, the terminal has a full buffer or a non-full buffer. There are various types of non-full buffer generation. For example, if you look at the FTP model that is mainly used in 3GPP, there may be a case where a buffer is updated or a new MS is newly arranged according to a TTI. For example, if you have two TTIs, you have to place a new MSs.

Next is channel creation. Channels are separated by long term channel and short term channel. In the long term channel, the channel is not influenced by the TTI according to the location of the terminal. In the short term channel,

it reflects the change according to the TTI in consideration of the mobility of the MS. Since 5G uses 3D SCM, it can be confirmed that the channel has a complex structure as follows.

```
void Channel::LongTermChannel()
{

    for (int msID=0; msID<Sim.network->numMS; msID++)
    {
        TemporaryGlobalVariableInitialize(msID);
        MS[msID]->channel->LongTermChannel(msID);

        setcolor(11, 0);
        double longTermChannelProcess;
        longTermChannelProcess = 95 * msID / (Sim.network->numMS);
        cout << "[Channel " << longTermChannelProcess << "%]: Long Term Channel of MS ID " << msID << " Calculated" << endl;
    }
}

void Channel::ShortTermChannel()
{
    for (int msID = 0; msID < Sim.network->numMS; msID++)
    {
        MS[msID]->channel->ShortTermChannel(msID);
    }
        setcolor(11, 0);
        cout << "[Channel]: Short Term Channel of MS(DFT) calculated" << endl;
}
```

Figure 5-7 Long/short term Channel generation

After the channel is created, it will go through the feedback and scheduling process. Feedback is used to report channel information based on well-known CQI or PMI. In the 5G simulator, we consider hybrid beamforming according to SU-MIMO or MU-MIMO and feedback PMI for analog / digital beamforming. Based on this information, the base station performs scheduling. In this case, round robin or PF is performed.

```
void Scheduling::Feedback()
{
    for (int msID = 0; msID < Sim.network->numMS; msID++)
    {
        //MS[msID]->scheduling->Feedback(msID); //Feedback(msID)
        MS[msID]->scheduling->MuMimoFeedback(msID); //MU-MIMO Feedback(msID)
    }
    for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
    {
        BS[bsID]->scheduling->MuMimoFeedback(bsID);
    }
    setcolor(13, 0);
    cout << "[Scheduling]: Mobile Station CSI feedback" << endl;
}

void Scheduling::Schedule()
{
    for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
    {
        for (int subband = 0; subband < Sim.numSubband; subband++)
        {
            for (int time = 0; time < Sim.numTTIperFrame; time++)
            {
                BS[bsID]->scheduling->MuMimoSchedule(bsID, subband, BS[bsID]->scheduling->scheduledMS, BS[bsID]->scheduling->framestructure(time));
            }
        }
    }
    cout << "[Scheduling]: Base Station scheduling " << endl;
}
```

Figure 5-8 Feedback & schedule function in RRM

Based on the scheduling information, we calculate the effective SINR and calculate the performance in the link performance module using the BLER curve obtained from link2system mapping. For example, there are spectral efficiency, UPT, etc., and you can get the output you want.

```
void Performance::Measure()
{
    for (int i = 0; i < Sim.network->numMS; i++)
        MS[i]->performance->instantThroughput = 0;

    for (int bsID = 0; bsID < Sim.network->numBS; bsID++)
    {
        BS[bsID]->performance->Measure(bsID);
    }


    double averagedPfWindowSize = 50.0;
    setcolor(15, 0);
    cout << "******** Throughput of TTI: "<<Sim.TTI<<" ******** << endl;
    for (int msID = 0; msID < Sim.network->numMS; msID++)
    {
        MS[msID]->scheduling->downlinkaveragedThroghput = MS[msID]->scheduling->downlinkaveragedThroghput*(Sim.TTI) / (Sim.TTI+1) + MS[msID]->performance->throughput / (Sim.TTI+1);
        cout << "[Performance]: Downlink Throughput msID " << msID << " - " << MS[msID]->scheduling->downlinkaveragedThroghput << endl;
        MS[msID]->scheduling->uplinkaveragedThroghput = MS[msID]->scheduling->uplinkaveragedThroghput*(Sim.TTI) / (Sim.TTI + 1) + MS[msID]->performance->throughput / (Sim.TTI + 1);
        cout << "[Performance]: Uplink Throughput msID " << msID << " - " << MS[msID]->scheduling->uplinkaveragedThroghput << endl;
    }

    for (int msID = 0; msID < Sim.network->numMS; msID++)
    {
        Sim.RateTTI(msID, Sim.TTI) = MS[msID]->performance->instantThroughput;
    }

}
```

Figure 5-9 Feedback & schedule function in RRM

Finally, at the end of the simulation, the memory is free and the result is exported to a file, and the simulation ends.

```
void SystemSim::Conclude()
{
    Sim.channel->Conclude();
    Sim.performance->Conclude();
    Sim.scheduling->Conclude();
    Sim.network->Conclude();

}
```

Figure 5-10 Conclude of simulation

6. References

[1] System Level Simulator – Requirement Specification

[2] System Level Simulator – High Level Design

[3] System Level Simulator – Module Design