User Manual

The Vienna 5G System Level Simulator

Institute of Telecommunications, TU Wien

Authors

Martin Müller, Fjolla Ademaj, Agnes Fastenbauer, Thomas Dittrich, Blanca Ramos Elbal, Armand Nabavi, Lukas Nagel, Stefan Schwarz and Markus Rupp

Vienna, October 4, 2018
The Vienna 5G System Level Simulator is part of the Vienna Cellular Communications Simulators (VCCS) software suite. The simulator is currently available under a non-commercial, academic use license. For download and license information of the simulator, please refer to our license agreement.
<table>
<thead>
<tr>
<th>7.4</th>
<th>Scheduling</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5</td>
<td>Feedback</td>
<td>39</td>
</tr>
<tr>
<td>7.6</td>
<td>Parallelization</td>
<td>40</td>
</tr>
<tr>
<td>7.7</td>
<td>Lite Version</td>
<td>41</td>
</tr>
<tr>
<td>7.8</td>
<td>Post Processing</td>
<td>41</td>
</tr>
</tbody>
</table>
1 Introduction

In cellular communications, simulations are an inevitable tool for understanding the mutual interactions of all involved players in the network. Especially for gaining insight in the performance of a large-scale scenario, a real-world measurement approach becomes too costly and laborious. Therefore, system level simulators were developed along with the standardization process of the current mobile communications standard Long Term Evolution (LTE).

Our research group originally started off in 2009 with a freely-available LTE link-level simulator and supplemented it with an LTE system-level simulator. These simulators received quite some attention and kept growing over time, adding more features as research and standardization evolved further. Thanks to the open-source nature of our simulators and the vivid exchange between developers and active users via an online forum, the Vienna LTE simulators have been downloaded more than 50,000 times in total.

Following the ongoing discussion about the the 5th generation of mobile networks (5G) we have introduced new 5G simulators to remain at the forefront of the latest developments. We again follow the approach to split this project in a link-level and a system-level simulator. The scope of this user manual is to give an overview of the capabilities and the general purpose of the Vienna 5G System Level (SL) Simulator, introduce its structure and describe the key features and their implementation details.

With the addition of the 5G System Level Simulator to the family of the Vienna Cellular Communications Simulators (VCCS), we tackle the need for simulating large scale networks, capturing the change in network layouts and physical transmission, coming up with the expected 5G standard. While the 5G standard has not yet been fully specified as of this writing, it is commonly agreed upon that future networks will become more heterogeneous. Therefore, our simulator allows to create networks of arbitrary layout with several tiers of Base Stations (BSs) and various user types in the same simulation.

The implementation is done in MATLAB and uses Object Oriented Programming (OOP). In general we made sure that the code structure is easy to expand, also with respect to the unknown prerequisites from the upcoming 5G standard. Due to the abstract structure of the code, we also provide backwards compatibility to our LTE system level simulator (if not in the full extent of all available options). The usage of parallel computing is also supported by our simulator, since individual simulation chunks are defined after an initial pregeneration step.

Our simulator performs Monte-Carlo simulations in order to achieve an average network performance. Therefore, we average over many spatial constellations and channel realizations and thus obtain results for average
throughput per user/BS, average Signal to Interference and Noise Ratio (SINR) performance and ratio of successful transmissions.

To determine the quality of each individual link, the instantaneous SINR is evaluated. For each transmission, the received power of all transmitters (desired and interfering) is calculated by combining distance dependent path loss, channel realization, antenna pattern and shadowing. It is possible to choose from several models and options for each of these individual propagation effects. Additionally, this is not a static choice that is set for the whole simulation, but is chosen dependent on the link conditions (e.g., Line-Of-Sight (LOS)/Non Line of Sight (NLOS)). These link-types can again be distinguished by different means. To stay with the LOS example, options are, e.g., to use pregenerated spatially correlated maps or to find obstruction of the transmission link from explicitly placed blockages in the scenario (a more explicit description can be found in Section 7.3).

Since the complexity of a simulation increases rapidly with the considered network size and the resulting large number of individual network elements, we perform an abstraction step for the actual transmission. Therefore, performance curves from the 5G link level (LL) Simulator are used for an SINR to Bit Error Ratio (BER) mapping to assess the success of each individual transmission, without the need to simulate all aspects of the Physical (PHY) channel.

The current version of our 5G System Level Simulator supports heterogeneous networks with an arbitrary number of BS tiers and user types, including mobile users. Thanks to the construction of the BSs objects with attached antenna objects, BSs with Remote Radio Heads (RRHs) and Distributed Antennas Systems (DASs) are available for simulations. Regarding the network geometry, not only BSs and users can be placed, but also 3-D blockages, resembling walls and buildings. Consequently, randomly generated cities can be created, such as a Manhattan grid layout or randomly placed buildings with arbitrary orientation. The new transmission features of 5G, such as mmWave and massive Multiple-Input Multiple-Output (MIMO) are represented in our simulator by the corresponding channel channel model for the right frequency range.


2 Quick Start

This quick start guide explains how to run the Vienna 5G SL Simulator for the first time.

1. Switch to the simulator’s root directory. Make sure the file simulate.m is present in your current working directory.

2. Open one of the scripts in launcherFiles. For illustration, let’s consider one of the predefined launchers, launcherFiles.launcherExample.m

3. In this launcher file, the scenario to be used is specified:

```matlab
% select scenario
result = simulate(@scenarios.basicScenario, parameters.
    setting.SimulationType.local);
```

4. Simulations are defined by their scenario files. In the folder +scenarios we find basicScenario.m which contains all the parameters that are needed to perform a simulation. To run your own simulation you can adapt this or one of the other scenarios to your needs or create your own scenario file. To start the simulation, replace basicScenario in step 3 with your chosen scenario. More information about scenarios can be found in Section 4.

5. In case you want to make use of multiple processor cores and parallelize the simulation, you can use SimulationType.parallel instead of SimulationType.local in step 3 (cf. Section 7.6).

6. After specifying the scenario to simulate, the results will be saved into the variable `result`. Various plotting functions can be called, depending on the postprocessor class specified in the scenario file (see Section 7.8 for more details).

```matlab
% plots the SINR ECDF of all user SINRs
result.showAllPlots;
```

7. Run the script launcherFiles/example.m. Various results are plotted when the simulation is finished.
3 Simulation Methodology

The Vienna 5G SL Simulator allows to investigate the performance of future large scale wireless networks. It is implemented in an abstract fashion by using OOP. The performance evaluation is done through Monte-Carlo simulations with a large number of randomly sampled realizations of a scenario that is generated according to the parameters specified prior to the simulation. A detailed description of the simulator can be found in Section 6.

The individual link quality is determined according to the geometry and the resulting relative position of receiver and transmitter as well as several propagation effects. The received power of all links is then combined in a SINR value, which is used later on in the actual transmission function (cf. Section 7.2).

An arbitrary number of BS and user types can be defined and placed according to a predefined placement function (cf. Section 7.1). The simulator provides several options, such as the classical hex-grid or random placement according to a Poisson Point Process (PPP).

The propagation effects that largely influence the quality of the transmission are split into different functions the appropriate options and models can be chosen independently per scenario and BS and user type. In our simulator, we provide the following options:

- Individual transmission power for different tiers
- Antenna patterns
- Large scale path loss (including distinction between different link conditions) (cf. Section 7.3.1)
- Shadowing (blockages are modeled explicitly or implicitly) (cf. Sections 7.1.3 and 7.3.2)
- Small scale fading in terms of channel models Section 7.3.3

The Multiple Access Channel (MAC) layer is represented by the scheduler function and by applying Adaptive Modulation and Coding (AMC). It utilizes the previously calculated SINR of the active links to determine the resource allocation and appropriate Modulation and Coding Scheme (MCS) for transmission (cf. Section 7.4).

In order to limit the complexity, the actual transmission is abstracted and several steps in the transmitter chain (e.g., coding or modulation) and receiver chain (e.g., decoding and demapping) are combined in two functions, namely the Link Quality Model (LQM) and the Link Performance Model (LPM). A
more thorough explanation of these functions can be found in Section 7.2. This abstraction step is the key enabler for supporting the simulation of very large scale scenarios with several thousand different nodes in a single simulation.

The acquired results are then selectively stored (dependent on the chosen settings) and can later be combined in average values. Additionally, the empirical cumulative distribution function (ecdf) of, e.g., the average user throughput or the user SINR are plotted.
4 Predefined Scenarios

In Section 2 we described how to run a simulation. To define a simulation we use so called scenario files. The desired parameters for a specific scenario are set in these files. Undefined parameters will be initialized with their default values. This way all options are concentrated in one spot and are not distributed in hidden configuration files. To keep simulations reproducible, we recommend saving (a copy of) the scenario file for each simulation that might be needed later on.

By opening a scenario file one can see that they contain a function. This function expects one parameter of the class `parameters.Parameter` and returns the same object filled with all the required settings. To start the simulation a function handle to the intended scenario e.g.: `@scenarios.basicScenario` needs to be passed to the `simulate` function in the simulator’s root folder.

4.1 Customization

For defining new scenarios we recommend to start from the existing scenario that is most similar to the intended simulation and modify a copy of it. One benefit of the configuration through functions is its flexibility. The only constraints are that the function has to take a `parameters.Parameter` object as input and returns this object with the properties set are required. Besides this, one can execute arbitrary MATLAB code in these functions. For example one can use calculations to find the right parameter values, call other functions or simply load data from configuration/data files. This flexibility allows to customize the scenario to a large degree.

4.2 Predefined Launchers and Scenario files

In the package `+launchersFiles` several launchers for simulating predefined scenarios can be found. The corresponding scenario files are defined in `+scenarios`. In the following, the specifics of these scenarios is explained, including the most important parameters and how they have to be defined.

4.2.1 Hexagonal grid with interference ring

The launcher file `launcherFiles.launcherHexRingInterferers.m` provides an example of how to simulate a hexagonal grid network of three-sector BSs with a defined number of rings of BSs and rings of interfering BSs. The launcher file starts by calling the corresponding scenario file located in
+scenarios, in this case scenarios.hexRingInterferers.m, as shown in the code-line below.

```matlab
result = simulate(scenarios.hexRingInterferers, parameters.setting.SimulationType.local);
```

The execution of this line will produce the results saved in the parameter result.

Afterwards, various plotting functions can be executed, depending on the chosen post-processing method, already defined in the scenario file. To show all plots, that are predefined, the code-line below is executed.

```matlab
result.showAllPlots;
```

Now let us have a closer look on how the scenario file is constructed for this particular example. The general network parameters are set in the scenario file with:

```matlab
interBSdistance = 280;
nRing = 2;
nRingInterferers = 1;
```

These settings are then used to define the size of the Region Of Interest (ROI) and the interference region:

```matlab
params.regionOfInterest.interference = parameters.setting.Interference.regionContinuousUser;
params.regionOfInterest.xSpan = 2 * interBSdistance * nRing;
params.regionOfInterest.ySpan = params.regionOfInterest.xSpan;
params.regionOfInterest.interferenceRegionFactor = 1 + 1/nRing + nRingInterferers/(2*nRing);
```

For this scenario a square ROI, that spans from the two most distant desired BSs is created. The interference region (cf. Section 7.1.5) is defined to be slightly larger than the distance between the outer interfering BSs.

Then, sector antennas are placed in a hexagonal grid, the antennas being defined with:

```matlab
antenna = parameters.basestation.antennas.ThreeSector;
antenna.nTX = 1;
```

The hexagonal grid is defined with:

```matlab
hexRing = parameters.basestation.HexRing;
hexRing.interBSdistance = interBSdistance;
hexRing.nRing = nRing;
hexRing.nRingInterferers = nRingInterferers;
```
The network parameters $\text{interBSdistance}$, $nRing$ and $nRingInterferers$ are the parameters defined at the beginning of the scenario file. Users are placed through a PPP.

In this scenario, some BSs from the ring of interfering BSs can be the desired BS of a user in the ROI. If this is the case, these BSs are fully simulated with full scheduling and feedback and the ROI-users attached to them produce simulation results. To show which BSs in the simulation were not only creating interference, the following code-lines mark these BSs positions in green.

```matlab
figure(1); hold on; BS = [result.networkElements.baseStationList]; BSroi = BS([result.networkElements.baseStationList.isRoi]); roiAnt = [BSroi.antennaList]; roiPos = [roiAnt.positionList]; roiPos = roiPos(:,1:10:end); scatter(roiPos(1,:), roiPos(2,:), 'MarkerFaceColor', 'green');
```

### 4.2.2 PPP distributed BSs with interference region

The launcher file `launcherFiles.launcherInterferenceRegionPPP.m` provides an example of how to simulate a network with BSs and users distributed according to a PPP with an additional interference region around the border of the ROI. The launcher file starts by calling the corresponding scenario file located in `+scenarios`, in this case `scenarios.interferenceRegionPPP.m`, as shown in the code-line below.

```matlab
result = simulate(@scenarios.interferenceRegionPPP, parameters.setting.SimulationType.local);
```

The execution of this line will produce the results saved in the parameter `result`.

Afterwards, various plotting functions can be executed, depending on the chosen post-processing method, already defined in the scenario file. To show all plots, that are predefined, the code-line below is executed.

```matlab
result.showAllPlots;
```
To create a boundary region and define its size, the `interference` and `interferenceRegionFactor` parameters have to be set in the scenario file, as is shown in the code-lines below.

```plaintext
params.regionOfInterest.interference = parameters.setting.
  Interference.regionContinuousUser;
params.regionOfInterest.interferenceRegionFactor = 1.5;
```

This parameter will be used to create the interference region in the function `createInterferenceRegion` from the class `parameters.regionOfInterest.RegionOfInterest`. The following lines of code lead to the creation of a network as shown in Fig. 8.

```plaintext
obj.interferenceRegion = parameters.regionOfInterest.Region();
obj.interferenceRegion.origin2D = obj.origin2D;
obj.interferenceRegion.xSpan = obj.xSpan * obj.
  interferenceRegionFactor;
obj.interferenceRegion.ySpan = obj.ySpan * obj.
  interferenceRegionFactor;
obj.interferenceRegion.zSpan = obj.zSpan * obj.
  interferenceRegionFactor;
```

The BS placement in the interference region is an expansion of the BS placement in the ROI, but for the user placement in the interference region, two placement options are available. The users can be placed, like the BSs in an expansion of the ROI-user placement, as they are in this scenario. Alternatively, the users can be placed uniformly in the interference region, independently from the user placement in the ROI. To place the interference region users independently, the following code-lines have to be changed or added to the scenario file.

```plaintext
params.regionOfInterest.interference = parameters.setting.
  Interference.regionIndependentUser;
```

```plaintext
interferenceUser = parameters.user.InterferenceRegion;
interferenceUser.nElements = 50;
params.userParameters('interferenceUser') = interferenceUser;
```

This user placement strategy ensures that the number of users in the ROI stays constant for moving users and the number of results produced by a chunk (cf. Section 6.2) is constant, which strongly simplifies the result handling in the post-processor.

At the end of the launcher file, the border of the ROI is added to the network plot.

```plaintext
roi = result.params.regionOfInterest;
```
4.2.3 Mobility and multiple chunks

The launcher file launcherFiles.launcherUserMovement.m shows how to enable user movement. This is done in scenarios.UserMovement with the code-line below:

```matlab
poissonUsersSISO.userMovement.type = parameters.user.MovementType.RandConstDirection;
```

where poissonUsersSISO contains the user parameters and is an object of the class parameters.user.Parameters. All supported movement types are listed in +parameters.+user.MovementType.m.

The simulation is run using the command

```matlab
result = simulate(scenarios.UserMovement, parameters.setting.SimulationType.local);
```

Afterwards, user movement is plotted. This is done using the commands

```matlab
for uu = 1:nUsersToPlot
    posList = result.networkElements.userList(uu).positionList;
    plot(posList(1,:), posList(2,:), 'b+');
    % draw arrow from start to finish
    quiver(posList(1,1), posList(2,1), posList(1,end)-posList(1,1), posList(2,end)-posList(2,1), 0, 'r');
    if uu == 1
        % plot path for the first user
        plot(posList(1,:), posList(2,:), 'k-');
    end
end
```

The variable nUsersToPlot is used to limit the number of users whose paths are plotted. This allows the individual users’ positions to be clearly discernable from each other, even if the total number of users is large. In the loop, the user positions are retrieved and plotted for each user for all time slots (TSs)
using blue plus signs. Next, a red arrow is plotted from the starting position to the end position to show the direction of movement. For the first user only, the path is shown by black lines connecting the user’s positions in the right order. This is particularly useful for random movement schemes where the path cannot simply be deduced from the positions and the arrow only. Showing the path only for the first user is done to avoid overcrowding the plot. If you want to plot the path for more users, you simply need to adapt the condition in the if statement. The next step zooms in around the first user so that its movement becomes clearly visible. This is often necessary since for typical simulation settings, the user speed and the simulation time lead to traveled distances that are very small compared to the size of the ROI.

```matlab
posList = result.networkElements.userList(1).positionList;
distXY = [posList(1,end)-posList(1,1), posList(2, end)-
posList(2, 1)]; % traveled distance in x and y
minX = min(posList(1,:));
minY = min(posList(2,:));
maxX = max(posList(1,:));
maxY = max(posList(2,:));
axis([minX-0.2*abs(distXY(1)) maxX+0.2*abs(distXY(1)) minY-
-0.2*abs(distXY(2)) maxY+0.2*abs(distXY(2))])
```

For demonstration purposes, the user speed is set to a high value in this simulation so that the traveled distance increases. If you want to display the movement of any subset of users, you only need to hand a list of the desired users to the for-loop in the first code fragment and comment out the second code fragment.

When using non-random movement schemes, a large gap between groups of plus signs (which correspond to user positions) is visible. This is due to the pause between simulation chunks. It is assumed that the user continues to travel at the same speed and in the same direction during this pause. Please refer to sections 6.2 and 6.3 for more details on TSs, segments and chunks.

### 4.2.4 Multi-tier and multi-user heterogeneous network

The launcher file `launcherFiles.launcherHetNet` provides an example of how to simulate a heterogeneous network consisting of different BS types and different user types. The launcher file starts by calling the corresponding scenario file located in `+scenarios`, in this case `scenarios.HetNet`, as shown in the code-line below.

```matlab
result = simulate(@scenarios.HetNet, parameters.setting.
SimulationType.local);
```
The execution of this line will produce the results saved in the parameter `result`. Afterwards, various plotting functions can be executed, depending on the chosen post-processing method, already defined in the scenario file. For example, using the command

```plaintext
result.showAllPlots;
```

the simulated network consisting of all its network elements can be plotted, as well as SINR and throughput ecdfs. It is possible to also add desired plotting functions in the launcher file. To demonstrate that, we give an example how to plot user-to-BS association based on the maximum received power.

As can be seen, all the input parameters characterizing a particular simulation scenario have to be included in the corresponding scenario file – in our example of `scenarios.HetNet`.

Various BS types can be defined. BS types can differ based on the distribution type, density, type of antenna, transmit power etc. Three different BS types are defined:

- **macro** - distributed according to predefined positions denoted with parameter `posMacro`, with a height of 25m and transmit power of 40W

- **pico** - distributed according to predefined positions denoted with parameter `posPico` along a straight street, with a height of 5m and transmit power of 2W

- **femto** - distributed according to a PPP with a given density, a height of 1.5m and transmit power of 0.1W.

The corresponding parameters for each of the three types are given below.

```plaintext
% macro \acp{BS}
predefPosMacro = parameters . basestation . PredefinedPositions ();
predefPosMacro . positions = [posMacro ;30*zeros (1 , size (posMacro ,2))];

% pico \acp{BS}
predefPosPico = parameters . basestation . PredefinedPositions ();
predefPosPico . positions = [posPico ;30*zeros (1 , size (posPico ,2))];
```
Accordingly, for each BS type, the antenna parameters are specified, such as antenna radiation pattern, height, number of transmit and receive antennas and transmit power. For example, for a femto BS we consider an omni-directional antenna pattern and the following parameters.

```plaintext
poissonFemtoBS.antenna = parameters.basestation.antennas.Omnidirectional;
poissonFemtoBS.antenna.height = 1.5;
poissonFemtoBS.antenna.nTX = 1;
poissonFemtoBS.antenna.nRX = 1;
poissonFemtoBS.antenna.transmitPower = 0.1;
```

Next, various user types can be added, differing in distribution, speed and movement pattern, as represented with the code below.

```plaintext
poissonUsersSISO = parameters.user.Poisson2D();
poissonUsersSISO.speed = 0;
poissonUsersSISO.userMovement.type = parameters.user.MovementType.ConstPosition;
```

Additionally, with the parameter

```plaintext
poissonUsersSISO.indoorDecision = parameters.indoorDecision.Random(0.1);
```

it can be specified that some of the users are placed indoors – 0.1 in our example means that 10% of users are indoors.

Having such a highly heterogeneous network, several link-specific path loss models can be utilized. This is possible via a predefined look-up table (cf. Section 7.3.1) of possible path loss models specified for specific link conditions, e.g., LOS/NLOS or indoor/outdoor. In the example below it is illustrated how this loop-up table is defined for our heterogeneous scenario.

```plaintext
params.pathlossModelContainer.setModel(
    parameters.setting.BaseStationType.macro, parameters.
    setting.Indoor.indoor, ...
    parameters.setting.Los.LOS, parameters.
    setting.PathlossModel.UMa3D);
params.pathlossModelContainer.setModel(
    parameters.setting.BaseStationType.macro, parameters.
    setting.Indoor.indoor, ...
)
We chose the 3rd Generation Partnership Project (3GPP) Urban Macro cell (UMa)-3-dimensional (3D) pathloss model for links from macro BSs, a free-space path loss model for links from pico BSs, and an indoor path loss model for links from femto BSs.

### 4.2.5 Manhattan city layout

The launcher file `launcherFiles.launcherManhattanGrid.m` provides an example of how to simulate buildings and streets arranged according to a Manhattan grid with BSs placed on the rooftop of buildings and users distributed along the streets. The launcher file starts by calling the corresponding scenario file located in `scenarios`, in this case `scenarios.ManhattanGridScenario`, as shown in the code-line below.

```
result = simulate (@scenarios.ManhattanGridScenario, parameters.setting.SimulationType.local);
```

The execution of this line will produce the results saved in the parameter `result`. Afterwards, various plotting functions can be executed, depending on the chosen post-processing method, already defined in the scenario file. For example, using the command

```
result.showAllPlots;
```

the simulated network consisting of all its network elements can be plotted, as well as SINR and throughput ecdfs. It is possible to also add desired plotting functions in the launcher file. To demonstrate that, we give an example on how to plot the entire scenario consisting of buildings, streets, BSs and users, as well as illustrating LOS and NLOS users attached to one of the BSs in the scenario.
As it can be seen, all the input parameters characterizing a particular simulation scenario have to be included in the corresponding scenario file. In our example of `scenarios.ManhattanGridScenario`, we show how to set up these input parameters.

In this scenario we use a city layout for distributing the buildings by using the following code,

```python
manhattanCity = parameters.city.Manhattan();
manhattanCity.blockLength = 50;
manhattanCity.blockWidth = 50;
manhattanCity.streetWidth = 35;
manhattanCity.minBuildingHeight = 10;
manhattanCity.maxBuildingHeight = 25;
manhattanCity.wallLossDB = 10;
params.cityParameters('manhattan') = manhattanCity;
```

where the size of the building block is specified by the parameters `blockLength` and `blockWidth`, the width of streets by the parameter `streetWidth` as well as the minimum and maximum building heights with the parameters `minBuildingHeight` and `maxBuildingHeight`, respectively. The penetration loss per wall is specified in the parameter `wallLossDB` given in dB.

BSs in this scenario are macro BSs and are placed on top of the buildings by using the code-line

```python
macroOnBuildings = parameters.basestation.MacroOnBuildings();
macroOnBuildings.occupationProbability = 0.2;
macroOnBuildings.margin = 2;
macroOnBuildings.type = parameters.setting.BaseStationType.macro;
macroOnBuildings.antennaHeight = 1;
macroOnBuildings.antenna = antenna;
params.baseStationParameters('macroOnBuildings') = macroOnBuildings;
```

where BSs are randomly distributed according to the occupation probability that we set to 20% in our example.

Antenna parameters are defined as

```python
antenna = parameters.basestation.antennas.Omnidirectional;
antenna.nTX = 1;
antenna.nRX = 1;
antenna.height = 25;
antenna.transmitPower = 40;
antenna.transmitPowerSignaling = 0;
```

Users are distributed on ground level according to a PPP and are assumed to be pedestrians with a speed of 5km/h.
poissonUsersSISO = parameters.user.Poisson2D();
poissonUsersSISO.nTX = 1;
poissonUsersSISO.nRX = 1;
poissonUsersSISO.density = 0.006;
poissonUsersSISO.height = 0.01;
poissonUsersSISO.indoorDecision = parameters.indoorDecision.Random(0.1);
poissonUsersSISO.speed = 5/3.6;
poissonUsersSISO.transmitPower = 1;
poissonUsersSISO.transmitPowerSignaling = 0;
params.userParameters('poissonUser') = poissonUsersSISO;

The simulator gives the possibility to save additional information at the end of the simulation such as a map indicating LOS/NLOS positions by setting

params.save.losMap = true;

and a map indicating whether the user is indoor or outdoor by setting

params.save.isIndoor = true;

### 4.2.6 IoT with clustered nodes

The launcher file `launcherFiles.launcherIoTclusteredUser.m` provides an example of how to simulate a large network with Internet of Things (IoT) devices that are active in bursts at regular intervals. The IoT users are placed in clusters around femto BSs, additionally users and macro BSs are placed in the network according to a PPP. The launcher file starts by calling the corresponding scenario file located in `+scenarios`, in this case `scenarios.IoTclusteredUser.m`, as shown in the code-line below.

result = simulate(@scenarios.IoTclusteredUser, parameters.setting.SimulationType.local);

The execution of this line will produce the results saved in the parameter `result`.

Afterwards, various plotting functions can be executed, depending on the chosen post-processing method, already defined in the scenario file. To show all plots, that are predefined, the code-line below is executed.

result.showAllPlots;

With the creation of the timeline (cf. Section 6.2) of this scenario, the activity bursts of the IoT devices is defined in the scenario file.
In this scenario, we simulate 10 bursts of activity, set with the parameter `numberOfChunks`, that each last 20 ms, as is indicated in `slotsPerChunk` and happen once in a minute, which is set in `timeBetweenChunksInSlots` with one slot being 1 ms long.

To create users in clusters with a femto BS at the center of the cluster the following lines of code have to be added to the scenario file.

```plaintext
clusteredUser = parameters.user.UniformCluster;
clusteredUser.nRX = 1;
clusteredUser.density = 25e-6;
clusteredUser.clusterRadius = 50;
clusteredUser.clusterDensity = 3e-3;
clusteredUser.transmitPower = 1;
clusteredUser.withFemto = true;
clusteredUser.femtoParameters.antenna = antennaFemto;
params.userParameters('clusterUser') = clusteredUser;
```

The antenna at the center of the user clusters is defined as a simple omnidirectional antenna with the following lines in the scenario file.

```plaintext
antennaFemto = parameters.basestation.antennas.Omnidirectional;
antennaFemto.nTX = 1;
antennaFemto.height = 10;
```

The transmit power of the femto BSs is not set in the scenario file, it is automatically set according to the BS type at BS creation in `networkElements.bs.BaseStation.setGenericParameters` with:

```plaintext
switch baseStationParameters.type
    case parameters.setting.BaseStationType.macro
        baseStationParameters.antenna.transmitPower = 20; \ % W
    case parameters.setting.BaseStationType.pico
        baseStationParameters.antenna.transmitPower = 2; \ % W
    case parameters.setting.BaseStationType.femto
        baseStationParameters.antenna.transmitPower = 0.1; \ % W
end
```

If a transmit power is set in the scenario file, these lines of code are not executed and the BS is simulated with the transmit power set in the BS
creation properties.

Additionally users and macro BSs are placed according to a PPP in the simulation region.

To display more detailed results, an additional plot is created in the launcher file for this scenario. It shows the user throughput of the IoT users and the PPP users separately:

```matlab
figure();
hold on;
clusterUser = result.params.userParameters('clusterUser').indices;
pppUser = result.params.userParameters('pppUser').indices;
clusterThroughput = result.userThroughputMBitPerSec.DL(clusterUser,:);
pppThroughput = result.userThroughputMBitPerSec.DL(pppUser,:);
ecdf(clusterThroughput(:));
ecdf(pppThroughput(:));
```

As it can be seen above, the results for the different types of users can easily be extracted from the results, since the indices of the users in the list of all users are saved for each user type. The results for different BS types can be extracted in the same manner.

### 4.2.7 Lite version

The launcher file `launcherFiles.launcherLiteSimulation.m` shows how to run a lite simulation. The corresponding scenario file for this launcher is `scenarios.basicLiteScenario.m`. As explained in Section 7.7, the lite mode can be used in all types of simulations. In this example in particular, BSs ans users are deployed according to a PPP. All parameters can be changed, but in this type of simulation some parameters like the scheduler are irrelevant. The line of code that defines, whether a simulation is lite or not is:

```matlab
params.postprocessor = simulation.postprocessing.LiteWithNetworkPP;
```

By default, the `postprocessor` parameter is set to PartialPP, which runs the full simulation. To select the lite mode, we can change this parameter either to LiteWithNetworkPP or to LiteNoNetworkPP. The difference between this lite postprocessors is explained in Section 7.8.

Once the simulation is done, the the function `showAllPlots` is called:

```matlab
result.showAllPlots;
```
The plots given by this function depend on the lite postprocessor employed for the simulation. If the postprocessor is LiteWithNetworkPP, the function showAllPlots is defined in simulation.results.ResultsLiteWithNetwork, and it plots the network elements positions and the users SINR. On the other hand, if the postprocessor is LiteNoNetworkPP, just the users SINR are depicted as shown in the function showAllPlots of simulation.results.ResultsLiteNoNetwork.

4.2.8 Simulation of several chunks in parallel

The launcher file launcherFiles.launcherParallelSim utilizes the basic parameters set in scenarios.basicScenario.m, but simulates the same scenario twice, once with the local simulation type (for serial simulation) and once with the parallel simulation type. The displayed simulation duration at the end of the simulation demonstrates the speed-up through parallelization. More info on this topic can be found in Section 6.2 and Section 7.6.

```matlab
% This line launches the simulation with the scenario defined in % scenarios.basicScenario with simulation type % parameters.setting.SimulationType.local.
fprintf('Linear simulation:\n')
result_lin = simulate(@scenarios.basicScenario, parameters.
    setting.SimulationType.local);

% This line launches the simulation with the scenario defined in % scenarios.basicScenario with simulation type % parameters.setting.SimulationType.parallel.
fprintf('\n\nParallel simulation:\n')
result_par = simulate(@scenarios.basicScenario, parameters.
    setting.SimulationType.parallel);
```
5 Comparison to LTE-A SL Simulator

The aim of this comparison is to show that results obtained with the Vienna Long Term Evolution-Advanced (LTE-A) SL Simulator can be reproduced with the Vienna 5G SL Simulator. For this purpose, the same parameter set is used in both simulators. The chosen parameters can be found in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>base stations</td>
<td>hexagonal layout, 1 ring, 7 BSs</td>
</tr>
<tr>
<td>users</td>
<td>50, uniform density</td>
</tr>
<tr>
<td>pathloss model</td>
<td>COST231 Urban Macro (UMa)</td>
</tr>
<tr>
<td>channel model</td>
<td>Pedestrian A PDP</td>
</tr>
<tr>
<td>TTI/slots</td>
<td>100</td>
</tr>
<tr>
<td>feedback delay</td>
<td>3</td>
</tr>
<tr>
<td>user speed</td>
<td>30 km/h</td>
</tr>
</tbody>
</table>

Table 1: Simulation parameters used to compare the Vienna LTE-A SL Simulator and the Vienna 5G SL Simulator

The simulations were run 200 times to make sure that the results can be compared fairly. Several metrics are used for this comparison.

At first, it is important to show that the user placement works the same way in both simulators. This is demonstrated by the ecdf of the distances between the users and their assigned BS as depicted in Fig. 1.

Next, the ecdf of the wideband SINR as shown in Fig. 2 is examined. The close match of the curves indicates that the statistics of the macroscopic path loss computed by both simulators are very similar to each other.

Finally, the ecdf of the average user throughput obtained by both simulators is compared in Fig. 3.

Here, small discrepancies between the curves can be noticed. They are caused by minor implementation differences between the simulators. Nevertheless, these simulations still show that the channel models and the function of the MAC layer behave similarly in both simulators, even with a non-zero feedback delay and a non-zero user speed.
Figure 1: ecdf of the distances between users and their assigned BS.

Figure 2: ecdf of the wideband SINR.
Figure 3: ecdf of the average user throughput.
6 Simulator Structure

6.1 A Typical Simulation

The Vienna 5G SL Simulator is written in MATLAB and is utilizing OOP. Individual parts of the simulator are separated into different packages and, e.g., network elements, such as BSs, are defined in classes. In general, the simulator is written in a modular fashion, such that new functions can be added easily, without the need to alter other parts of the code.

The simulator's structure is defined by four major parts, which are displayed in Fig. 4.

In the following, the most important steps of a simulation are explained and the corresponding functions to the four major parts of the simulator are introduced.

Each simulation begins with running the desired simulation launcher file (cf. Section 6.1). This already fixes the parameter set which is going to be used for this particular simulation. In the file simulate.m, you find the line

```matlab
localSimulation = simulation.LocalSimulation(params);
```

where the simulation object is created and the predefined parameters are attached to this object.

Next, the simulation time line (cf. Section 6.2) is generated, as well as the network element objects (cf. Section 7.1). Additionally, the fast fading traces are pregenerated, if necessary. This is done in

```matlab
localSimulation.setup();
```

All of this is stored in `localSimulation.simulationSetup`.

Now that the pregeneration is done, the actual simulation is carried out. The following line contains the main simulation loop (over chunks and TSs - cf. Section 6.3):

```matlab
localSimulation.run();
```

Here, random samples in time (represented by channel realizations and scheduling decisions) and/or in space (represented by the geometry of the network elements) are simulated and the individual results for each TS are stored.

\[\text{Note that we describe on a local simulation - i.e., we set parameters.setting.SimulationType.local. The explained steps are the same for a parallel simulation.}\]
The postprocessing step then follows in

```matlab
1 % get results
2 simulationResults = localSimulation.simulationResult;
```

where the individual results from the main simulation loop are extracted and stored in the results object. Which results are actually calculated and stored also has to be specified in the simulation parameters. After processing and storing the results, selected results are plotted, which is integrated in the simulation launcher.

### 6.2 The Simulator Time Line

The time line of the simulator is divided into three different units, namely *time slots (TSs)*, *segments* and *chunks*. The TS is the shortest unit and also corresponds to the scheduling granularity. Thus it corresponds to one iteration of the inner simulation loop (cf. Section 6.3). It has a constant length, e.g., 1 ms to represent an LTE-A subframe, but can otherwise be specified freely. A segment consists of various chunks and corresponds to the time (and distance) in which the macroscopic fading (MF) is assumed to be constant (e.g., the user association, large scale path loss values, but not the channel realization). Thus, the MF values are only updated at the beginning of a segment, including the user association. The length of a segment depends on the user speed and trajectory, as well as the correlation distance of the MF values. This means that for a stationary scenario, only a single segment is created. A chunk consists of a fixed number of TSs and on one or more segments (depending on the maximum user speed). For creating the user trajectory, a consecutive generation is assumed, but also a considerably long distance between chunks, which leads to uncorrelated user positions among
N chunks in total

<table>
<thead>
<tr>
<th>Chunk 1</th>
<th>Chunk 2</th>
<th>...</th>
<th>Chunk N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M segments per chunk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>K TS per segment</td>
</tr>
</tbody>
</table>

Figure 5: The different time units utilized in the simulator.

chunks. Even for stationary scenarios, channel coefficients are assumed to be uncorrelated, due to the assumption of a non-constant scattering environment. The chunks are also the basis for parallelization, since all necessary data is independent for each chunk and the results can be calculated without knowledge of the simulation result of the previous chunks. A more detailed description of the parallelization can be found in Section 7.6.

The following parameters are utilized to set up the time line:

```matlab
params.time.numberOfChunks = 2;
params.time.slotDuration = 1e-3; % duration in seconds
params.time.slotsPerChunk = 10; % each chunk consists of that many slots
params.time.timeBetweenChunksInSlots = 50; % timespan between two simulated time chunks
```

In the function `simulation.ChunkSimulation.m`, the following function is contained:

```matlab
function setNewSegmentIndicator(obj)
% creates a logical array that is true for all slots that are the first in a segment
% Marks all slots, for which a user has moved further than the
% maximum correlation distance. Large scale parameters are
% constant for a segment.
```
This means that based on the trajectory and correlation distance an indicator is set, where a new segment starts.

### 6.3 The Main Simulation Loop

The main simulation loop contains a loop over chunks, whereas each chunk contains a loop over TSs (cf. Fig. 6). The loop over chunks is contained in the function `simulation.LocalSimulation.run` for local simulations:

```matlab
1  % initialize chunk result list
2  chunkResultList(obj.parameters.time.numberofChunks) = ...
3     simulation.ChunkResult;
4
5  % run simulation for each chunk
6  for ii = 1:obj.parameters.time.numberofChunks
7     fprintf('simulating chunk %d...\n', ii);
8     obj.chunkSimulationList = [obj.chunkSimulationList, ...
9         simulation.ChunkSimulation(obj.simulationSetup.
10        chunkConfigList(ii))];
```
chunkResultList(ii) = obj.chunkSimulationList(ii).runSimulation();
end

and in the function simulation.ParallelSimulation.run for parallel simulations:

```matlab
% prepare chunk simulation list
chunkSimulationList = [];
for ii = 1:(obj.parameters.time.numberOfChunks)
    chunkSimulationList = [chunkSimulationList, simulation.ChunkSimulation(obj.simulationSetup.chunkConfigList(ii))];
end

% run simulations for all chunks
parfor ii = 1:obj.parameters.time.numberOfChunks
    fprintf('simulating chunk %d...\n', ii);
    chunkResultList(ii) = chunkSimulationList(ii).runSimulation();
end
```

For both options, all necessary data is distributed among chunks. This individual configuration per chunk is stored in chunkSimulationList. Irrespective of local or parallel simulation, each chunk contains the inner loop over TSs. This loop can be found in the function simulation.ChunkSimulation.runSimulation, where the loop begins after several initializing steps:

```matlab
% main simulation loop
for iSlot = 1:obj.nSlots
    if obj.chunkConfig.isNewSegment(iSlot)
        % In a new segment the cell association can change and
        % handovers need to be performed.
        obj.updateUsersAttachedToBaseStations(iSlot);
        obj.filterPureInterferenceCells(iSlot);
        % perform handovers - clear feedback buffers of users in new
        % cell
        obj.performHandover(iSlot);
    end % if this slot is the first one in a segment
```
Another segment update is performed per user and sets the appropriate MF values for each user for the current segment:

```matlab
if obj.chunkConfig.isNewSegment(iSlot)
    bb = obj.userToBSassignmentArrayDL(iUE, obj.getiSegment(iSlot));
    desired = false(1, obj.nAntennas);
    desired(bb) = true; % only works for one antenna per bs

    % get new macroscopic parameters
    thisShadowFadingdB = obj.shadowFadingdB(:, iUE, iSlot).';
    pathlossdB = obj.pathLossTableDL(:, iUE, obj.getiSegment(iSlot)).' + obj.wallLossdB(:, iUE, obj.getiSegment(iSlot)).';
    gaindB = obj.antennaGaindB(:, iUE, obj.getiSegment(iSlot)).';

    % update macroscopic parameters
    LQMDL(iUE).updateMacroscopic(desired, pathlossdB, gaindB, thisShadowFadingdB);
end
```
7 Overview of Key Functionalities

7.1 Generation of Network Elements and Geometry

System level simulations aim to evaluate the performance of a large network comprising a substantial number of BSs and users. In this regard, at its core the Vienna 5G SL Simulator simulates the communication between users and BSs. Due to its modular structure, the simulator allows the coexistence of different BS and user types. This, on one hand, allows to simulate multi-tier networks, and on the other hand, by also supporting different user types, more diverse and realistic scenarios are supported. Additionally to various propagation models that can be used, there is the option to distribute blockage objects and use the geometry to calculate different propagation parameters.

7.1.1 Base Stations

The simulator distinguishes between the entity BS and antenna. Each BS can have one or more antennas attached, which can be seen as physical entities with a position \( \{x, y, z\} \) in the 3D space. The antenna object represents an antenna or an antenna array with \( N_{\text{Tx}} \) transmit and \( N_{\text{Rx}} \) receive antennas. BSs do not have physical locations but instead, the physical positions are specified on the assigned antenna objects. This structure also enables the simulation of DASs and RRHs without additional extensions.

The BS object is defined in `+networkElements/+bs/BaseStation.m` and is characterized by the following properties:

- \textbf{antennaList} - list of all attached antennas
- \textbf{attachedUsers} - a list of users that are connected to this BS containing all user properties
- \textbf{type} - type of the BS, i.e., macro, pico, femto
- \textbf{isRoi} - indicator for BS that is in the ROI

Different placement methods that can be used to define placement of BSs are given in `+networkGeometry`. Which placement method is to be used, is specified in the scenario file (see 4.2.4 for more details).

Each BS type is indicated with an integer, specified in the corresponding file `+parameters/+setting/BaseStationType.m`. If a new BS type is to be added in the simulator, then it has also to be indicated with a corresponding enumeration. This has to be added to the following list:

```
1 enumeration
2 % NOTE: numbers have to be 1,2,3, ... n (they are mapped to a cell array)
```
The \textit{antenna} object is defined in \texttt{networkElements/+bs/Antenna.m} and is characterized by the following properties:

- \texttt{id} - integer identifier of antenna
- \texttt{baseStationType} - type of BS that this antenna belongs to
- \texttt{usedCCs} - carriers on which this BS transmits
- \texttt{nTX} - number of transmit antennas
- \texttt{nRX} - number of receive antennas
- \texttt{alwaysOn} - logical index that indicates if BS is always transmitting. If this is true the antennas attached to this BS are always transmitting and generating interference even if no users are scheduled at the BS in this slot.
- \texttt{rbGrid} - scheduling information and details about resource blocks
- \texttt{gaindBmax} - maximum antenna gain in dBi
- \texttt{azimuth} - azimuth angle in degrees in which the antenna has its maximum gain

Various antenna types are defined in \texttt{networkElements/+bs/+antenna.m}.

### 7.1.2 Users

The user object, as the other endpoint of the communication link is defined in \texttt{networkElements/+ue/User.m}. Properties that characterize a user object are:

- \texttt{id} - integer identification number
- \texttt{nRX} - number of receive antennas of this user
- \texttt{nTX} - number of transmit antennas of this user
- \texttt{txMode} - transmission mode
- \texttt{transmitPower} - transmit power
channelModelTypes - possible channel model types this user could use during the simulation. The supported channel models are defined in parameters.setting.ChannelModel.

currentChannelModel - channel model that is valid for the user in the current slot

speed - user speed given in m/s

Different placement methods that can be used to define placement of users are given in +networkGeometry. Which placement method is to be used, is specified in the scenario file (see [4.2.4] for more details how to set different user types within one scenario).

7.1.3 Blockages

On top of the network element generation, the Vienna 5G SL simulator supports the generation of blockages. The super class of blockage elements is defined in +blockages/Blockage.m. The basic building block to represent signal blocking objects, is a rectangular wall with arbitrary dimensions and orientation in 3D. Buildings are created by combining five walls (four walls on the sides and a ceiling). With these basic elements, arbitrary city layouts can be generated, such as a Manhattan grid with streets and building blocks.

In +blockages/RectangularBuilding.m a rectangular building is defined. It has the following properties that need to be specified:

- length - length defined in the direction of x-axis
- width - width defined in the direction of y-axis
- height - height defined in the direction of z-axis
- angle - angle in radians according to which the building is rotated with respect to the ground plane
- center - position in (x, y) of the building center.

An example of such a building can be found in Fig. 7.

Additionally there is the option to generate buildings together with streets, such as for example in the object ManhattanCity.m where buildings and streets are generated according to a Manhattan grid layout.

7.1.4 Cell association

In the Vienna 5G SL simulator, two different strategies for user-to-BS association are possible. These are:
The cell association strategy has to be chosen in the scenario file, by setting one of the following lines,

```
1   % maximum SINR
2   params.downlinkAssociationStrategy = parameters.
       setting.CellAssociationStrategy.maxSINR;
```

or

```
1   % maximum received power
2   params.downlinkAssociationStrategy = parameters.
       setting.CellAssociationStrategy.maxReceivePower;
```

The cell association strategy is applied per segment and is calculated in the main simulation loop. The respective function is located in +simulation.ChunkConfig.m. After the network element placement, the path loss, antenna gain and penetration loss are calculated for each possible link (BS-to-user). Afterwards in the association step, for each user the link (and corresponding BS) with highest SINR or maximum received power is chosen. Due to user mobility, the cell association is updated on segment basis whenever there is change in the network that affects the cell association conditions.
7.1.5 Interference Region

In SL simulations with finite simulation areas the network interference is inadequately represented for users that are close to the border of the simulated region. To mitigate these border effects, an interference region can be added to the simulation area in the Vienna 5G SL simulator. The interference region is an addition to the ROI at its border, as depicted in figure 8.

The settings for size and user placement in the interference region are described in `parameters.regionOfInterest.RegionOfInterest`, the options for interference regions are listed in `parameters.setting.Interference`. The users in the interference region can either be placed in the same manner as in the ROI or a fixed number of users is uniformly distributed in the interference region. The placement of the BSs and their antennas is an extension of the BS placement in the ROI. The user placement strategy is chosen, when setting an interference region with the parameter `RegionOfInterest.interference`. This is also described in Section 4.2.6.

The size of the interference region is set in the parameter `RegionOfInterest.interferenceRegionFactor`. It indicates by which factor the total simulation region is bigger than the ROI, i.e., the diameter of the ROI is multiplied by this factor to get the diameter of the interference region.

To save computational complexity the BSs in the interference region are not fully simulated unless a user of the ROI is attached to them. Instead, simplified random scheduling and feedback are used to simulate the interference region BSs.
7.2 Link Quality and Link Performance Model

7.2.1 Link Quality Model

The Link Quality Model (LQM) quantifies the quality of a link [1] in two SINR measures: the post equalization SINR, that is calculated for each layer of each resource block, and the wideband SINR that represents the quality of all physical resources in one time slot. The dimensions of the output of the LQM are depicted in Fig. [9].

The post equalization SINR takes into account the utilized receive filter and precoder and the inter layer interference, as well as the transmit power allocation. The wideband SINR is calculated based on the large scale parameters and considers antenna gain and path loss in its calculation, as well as transmit power and noise.

To calculate these SINR values, the LQM needs constant parameters, like the receiver noise figure, that are set in the class constructor, large scale parameters, like path loss and antenna gain, that have to be updated for each segment with the function updateMacroscopic and small scale channel parameters that change for each time slot, like the channel matrix and transmit power allocation, that are updated in updateSmallScale. The example function shows how to use the LQM package. The general (simplified) functioning of the LQM is depicted in Fig. [10].

The implementation assumes that a Zero Forcing receive filter is used and that the small scale fading is constant for the duration of a time slot. It is also assumed that if no user is scheduled at an interfering BS, that this BS does not generate any interference, unless the alwaysOn feature is activated at the BS antenna. Further assumptions are that one receiver operates on a constant number of layers within a time slot and that the allocated transmit power is evenly distributed between those layers.

7.2.2 Link Performance Model

The Link Performance Model (LPM) takes the SINR values that were calculated by the LQM and the decision of the scheduler to determine the throughput of a given user in terms of the number of bits. This is done in three steps:

1. Calculate an average SINR value for every transmitted codeword over all scheduled Resource Blocks (RBs) and transmission layers.

2. Map the average SINR to a Block Error Ratio (BLER) value
Figure 9: Output of the LQM: the post-equalization SINR is calculated for each resource block and each layer, the wideband SINR is calculated for each slot.

Figure 10: Schematic of the LQM.
3. Determine the number of transmitted bits based on the Channel Quality Indicator (CQI) and the BLER.

The last step is based on a Bernoulli experiment that determines if the current transmission was successful or not. In the case of a successful transmission the number of transmitted bits is set to the maximum size of the Transport Block (TB) for the scheduled RBs and for a transmission failure it is set to zero. In the simulator it is possible to select if the LPM should perform the Bernoulli experiment in every slot or if it should give the expected value of the experiment as output.

On top of the actual throughput the LPM calculates an upper bound on the throughput. This upper bound is the throughput that would result if the scheduler decides for the CQI that would result in the highest throughput. To get this value the LPM performs the above mentioned steps for all possible CQI values and takes the maximum over the resulting throughput.

7.3 Simulation of Propagation Effects

7.3.1 Path Loss Modeling and Situation dependent Model Choice

The aim, when implementing the path loss modeling was to assure modularity and flexibility on one hand but a simple, straight forward usage on the other.

As for the other parts of the simulator an object oriented approach was chosen. The superclass PathlossModel ensures that all path loss models can be used in the same way with the functions set in it. Like the other models, it can be found in the package macroscopicPathlossModel.

For a description of the necessary parameters to be set, an enumeration file parameters.setting.PathlossModel lists and describes all available path loss models. Another advantage of this enumeration file is that it offers autocomplete for the path loss model names, when choosing a model. The path loss is calculated without wall loss and antenna gain.

The actual path loss values are calculated for each chunk separately in simulation.ChunkSimulation.calculatePathloss. This function is called in the initialization phase of each chunk, before the loop over TSs. First of all, the link condition is determined, e.g., LOS or NLOS link. Then, the appropriate model for the considered link type is chosen. An example of this can be found in the scenario file for the predefined scenario described in Section 4.2.4.
7.3.2 Modeling of Shadow Fading

The modeling of the Shadow Fading (SF) is based on Shadow Fading maps (SFM). Those SFMs are arrays of so called Shadow Fading Values (SFVs) which are spatially correlated Random Variables (RVs). Such an array of RVs can be generated by means of the following steps:

1. Generate an array of uncorrelated Gaussian RVs
2. Apply the FFT to the array
3. Multiply the transformed array with the Power Spectral Density (PSD) of the intended spatial correlation
4. Apply the inverse FFT

The advantage of this method is, that correlated shadow fading values for positions with minimal granularity can be generated without the need to resolve the whole ROI with the same precision. A more precise explanation on this topic can be found in [2].

7.3.3 Modeling of Small Scale Fading - Channel Models

PDP Channel Models: The used channel models are defined through a Power Delay Profile (PDP). Descriptions of the available channel models can be found in the enumeration file parameters.setting.ChannelModelType. This enumeration file also refers to the standards, in which the models are defined [3–7].

Trace Generation: For the PDP channel models, channel traces are generated before the main simulation loop. The generation of these channel traces is handled by the PDPcontainer in the smallScaleFading package. There, all possible combinations of number of receive antennas, number of transmit antennas, carrier frequency and channel model that can occur during the simulation are evaluated and channel traces for these scenarios are saved in the folder dataFiles/channelTraces. If necessary, a different folder can be specified in the SmallScaleParameters class.

To get the channel realizations for a time slot the function PDPcontainer.updatePDPcontainer loads the needed channel traces into memory and the function PDPcontainer.getChannelForAllAntennas selects a random part of the trace to get a channel realization for one time slot. The channel traces should be much longer than the total simulation time to assure that the samples taken from the trace are independent from each other, but the trace
should be as short as possible, since they require memory. The length of the channel traces has to be set in the SmallScaleParameters class.

The function smallScaleFading.example shows how to use the PDP container to generate channel traces for the link quality model and what parameters need to be set.

**Assumptions:** In general the channel is assumed to be constant in time for a time slot and a single value is calculated for the channel in the time domain. In case more different values for a TS are needed, a short block fading scenario is possible. For this the SmallScaleParameters class symbolTimes has to specify the time indices of the resource elements for which a channel realization should be calculated.

In the frequency domain the channel is assumed constant for a resource block. Values for all subcarriers are calculated but only a fraction of them is saved in the channel traces. In the SmallScaleParameters class this fraction can be specified, otherwise one value is saved for each resource block in frequency.

When the option correlated fading is chosen, the channel model is created according to [8] and a constant user speed has to be set to generate a channel trace. This constant user speed is used to calculate the Doppler frequency for the channel.

### 7.4 Scheduling

The basic version of scheduling is always done separately for every BS. Thus, there is always a one to one relation between an instance of the Scheduler class and a BS. In the instance of Scheduler the corresponding BS is called attached BS.

As mentioned before, the scheduling in this simulator is built on top of the class Scheduler. This class is an abstract class that defines the functions scheduleDL, addUserDL and removeUserDL. Those functions have to be implemented by every possible type of scheduler.

- **scheduleDL:** This function has to be called once for every simulation slot and allocates users and transmit power to RBs.

- **addUserDL:** Has to be called whenever a user connects to the attached BS. It adds users to the scheduler queue.

- **removeUserDL:** Has to be called whenever a user disconnects from the attached BS. It removes users from the scheduler queue.
For example, a round robin scheduler would add newly arriving users at the end of the queue and always assign the RBs to a certain amount of users that are at the front of the queue. Those users will then be placed at the end of the queue.

Additionally to those abstract functions, there are two more pairs of functions that are not abstract:

- scheduleDLCommon: Has to be called from within scheduleDL and performs all calculations of the scheduler that are common to all types of schedulers. For example it decides the CQI for all RBs that are scheduled for one user.

- updateAttachedUsersDL: This function can be called instead of calling the two functions addUsersDL and removeUsersDL separately. It takes a list of users that should be attached to the BS/Scheduler and calls the two separate functions if necessary.

### 7.5 Feedback

The Feedback class is used to provide the scheduler with information on channel conditions such as the Rank Indicator (RI), the Channel Quality Indicator (CQI) and the Precoding Matrix Indicator (PMI). The scheduler needs this to determine optimal transmission parameters for future TSs.

All supported feedback types are implemented as subclasses of the Feedback class. Currently, these include

- LTE DL Feedback: This feedback type corresponds to the one used in the LTE Downlink.

- Minimum Feedback: This feedback type shows the minimum amount of feedback required by the scheduler.

In our simulator, feedback is currently only supported for transmit mode 1 (Single-Input Single-Output (SISO)) and thus only the CQI is utilized.

Each feedback type implements the method calculateFeedback. For LTE, the CQI values are computed using the reported SINR values. All supported SINR to CQI mappings are implemented in subclasses of parameters.transmissionParameters.CqiParameters. They include

- LteCqiParametersTS36213NonBLCEUE1

- LteCqiParametersTS36213NonBLCEUE1withoutCalibration,

which are defined in 3GPP TS 36.213 [9].
7.6 Parallelization

In the function `simulate.m`, three different simulation modes are defined:

```matlab
% start simulation
switch simulationType
 case parameters.setting.SimulationType.local
 % create simulation object
 localSimulation = simulation.LocalSimulation(params);
 % setup simulation and generate network elements
 localSimulation.setup();
 % main simulation loop
 localSimulation.run();
 % get results
 simulationResults = localSimulation.simulationResult;

case parameters.setting.SimulationType.parallel
 % create simulation object
 parallelSimulation = simulation.ParallelSimulation(params);
 % run simulation
 parallelSimulation.setup();
 parallelSimulation.run();
 % get results
 simulationResults = parallelSimulation.simulationResult;

case parameters.setting.SimulationType.cluster
 error('Type.cluster is not yet supported.');
otherwise
 error('Please see parameters.setting.SimulationType for possible simulation types.');
end
```

The major difference between the first two modes (local and parallel) is that the loop over chunks is a `parfor` loop in parallel mode. The chunk list is prepared (almost) the same way in both cases (cf. Section 6.3). The list is prepared in such a way that all chunks can be distributed to individual workers and no result of any chunk is necessary to do the computation in any other chunk.
7.7 Lite Version

The simulator provides a lite simulation mode, which offers a runtime shorter than the complete simulation. This simulation mode is useful when the focus of investigation is on the network geometry. The simulation result is the instantaneous SINR and consequently other metrics such as outage or achievable rate. In the lite mode, not the full functionality of the simulator is utilized, including features like scheduling or the LPM/LQM.

The simulation skips the computation of the parameters within TSs and the saved parameters, i.e, SINR are updated just once per segment. In the `simulation.ChunkSimulation.runSimulation` function, some basic computation for the new segments are done for all users, including the users outside the ROI. After that the scheduling function is called with the line

```java
obj.scheduling(s, iSlot)
```

In the lite simulation mode a dummy scheduler is used. Afterwards the computation of the first segments for users within the ROI are done. Consecutively, the calculations in each time slot are skipped with the line:

```java
if ~ obj.chunkConfig.parameters.liteSimulation %
   Skip time slots in lite simulation mode
```

The parameter `params.liteSimulation` indicates if the simulation is lite, but it should not be manually modified since it is set automatically according to the post processor chosen in the Parameter file. All launcher files can be run with the lite simulation mode. To do so, the postprocessor has to be changed to one of the lite postprocessors explained in Section 7.8.

7.8 Post Processing

Once the main simulation loop is done, the results of the individual chunks are collected and combined in the post processing phase, calling the function `combineResults` of the postprocessor defined for the simulation.

Our simulator is equipped with one postprocessor for full simulations and two postprocessors for lite simulations:

- **PartialPP**: This postprocessor for full simulations saves the general results, defined by the `ResultsPartial` class plus the additional results indicated by the `SaveObject` parameter. The results saved with this postprocessor are:
  - $\text{SNR DL}$ and $\text{SINR DL}$: Signal to Noise Ratio (SNR) and SINR values - one value per segment.

- **userThroughputMBitPerSec**: contains the downlink throughput of each user within the ROI in each TS in [Mbit/s].
- **userThroughputBit**: downlink user throughput in bits.
- **widebandSinr**: contains the wideband SINR of all users within the ROI in each slot in [Mbit/s].
- **effectiveSinr**: effective downlink SINR per TSs.
- **assignedBS**: assigned BS from all users per TSs.
- **feedback**: feedback parameters from all users for each TS.
- **schedulerSignaling**: scheduling information of all users for each TS.
- **networkElements**: contains the information related to users, BSs, blockages, buildings and street geometry.
- **params**: parameters for the simulation defined at the beginning of the simulation.
- Additional results: losMap, isIndoor, antennaBsMapper and pathlossTable.

- **LiteNoNetworkPP**: This postprocessor is employed in lite simulations when we do not want to save the network elements. Since in the lite simulation the functionality is reduced, many parameters saved with the PartialPP are not computed. In the result file just SNR and SINR values, as well as the necessary parameters for the simulation stored in params are saved.

- **LiteWithNetworkPP**: Also utilized for lite simulations - similar to the LiteNoNetworkPP postprocessor but in this case the network element are also saved.

By default, in the simulator the PartialPP is set. To select the lite simulation, the parameter `postprocessor` should be change in the scenario configuration file:

```python
params.postprocessor = simulation.postprocessing.
LiteNoNetworkPP;
```

With any of the postprocessors, the variables are stored in the results folder a file in with a filename is defined in `params.filename`. By default, the name contains the carrier frequency, the bandwidth, the time and whether the simulation is lite or not. To change the name, the function `Parameters.setFilename` should be modified.
Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. This work has been co-financed by A1 Telekom Austria AG, Kathrein-Werke KG and Nokia Solutions and Networks.
References


