Scheduling over 5G Radio Access Networks

Summer School on Dynamic Resource Allocation Problems in Communication Networks

5G Wireless Technology

5G stands for the fifth generation of mobile networks, introducing a new level of services and capabilities.

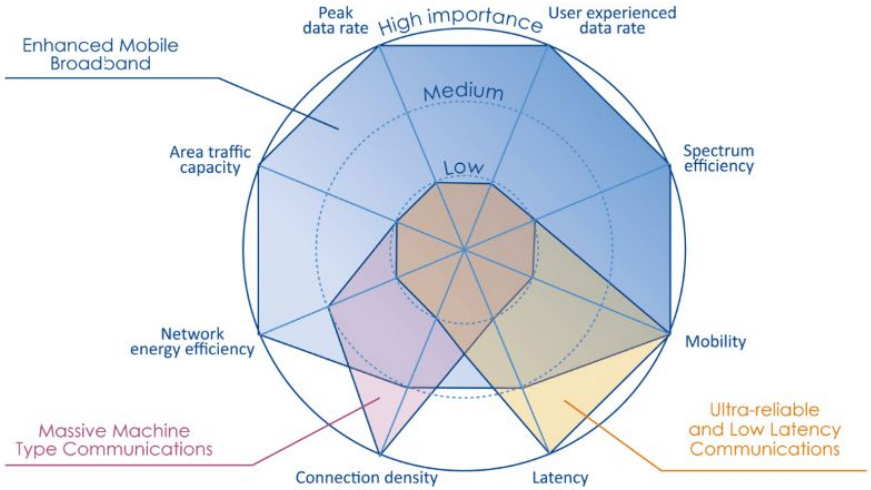
What’s New in 5G

- Lower latency and energy consumption
- Higher data rates
- Enables new applications
- Offers personalized services

5G Service Requirements: Differentiating User Profiles

Enhanced Mobile Broadband (eMBB)	High-speed data	Improved user experience
Massive Machine Type Communications (mMTC)	Supports IoT devices	Network efficiency
Ultra-Reliable Low Latency Communications (URLLC)	Critical applications	Real-time communication

5G Service Requirements: Differentiating User Profiles



Every profile shares the same network infrastructure.

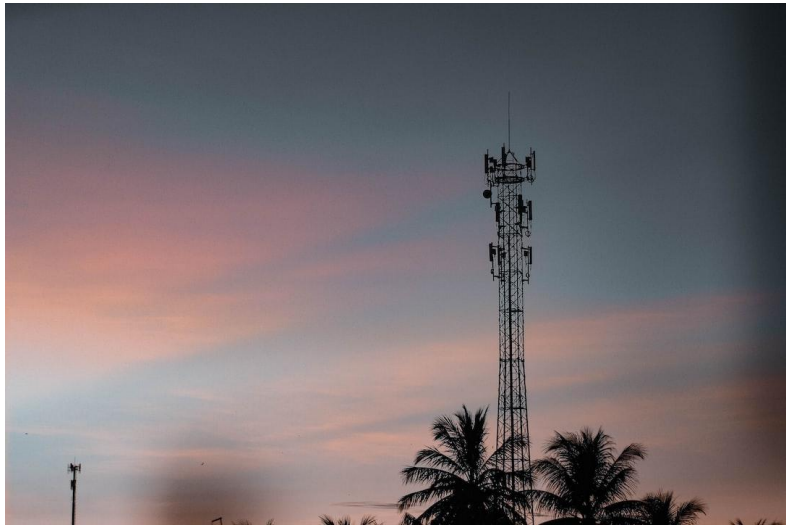
Shared Infrastructure Across 5G Use Cases

Common Network Elements:

- Core network
- Radio access network (RAN)
- Spectrum resources

Benefits of Shared Infrastructure

- Cost efficiency
- Seamless integration
- Scalable and flexible deployment



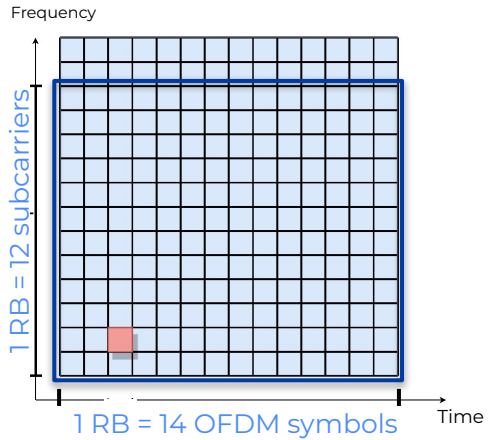
In the RAN context, how can users communicate with the network?

They are capable of data transmission and reception through the use of physical resource blocks

Physical Resource block:

- Physical Resource Block: Minimum assignable unit in 5G's RAN:
  - Defined in frequency as 12 subcarriers and typically 14 OFDM symbols in time domain
- More Resource Blocks = Better Data Rates:
  - Increased allocation leads to higher perceived data rates for users
  - The data rate also depends on the channel quality
- Faster RB Assignment = Lower Delay:
  - Rapid allocation reduces delay, enhancing user experience

Physical Resource Block Representation:



Resource Block Assignment

- **Efficient Allocation**: Resources must be allocated among users within the same network.
- **Scheduler Role**: RBs are assigned by a scheduler via control channels defined in the 5G standard.
- **Impact on Performance**: Proper RB allocation directly impacts network performance and user-perceived quality of service.

Classical Schedulers

Round Robin

- Each user is given an equal amount of resources
- Does not prioritize tasks based on their importance or urgency
- Tasks with shorter requirements might wait unnecessarily

Max CQI

- Prioritizes users with the best channel quality
- Aims to maximize the overall system throughput
- May lead to fairness issues, as users with consistently poor channel conditions might receive fewer resources.

Proportional Fair

- Maximizes network throughput while ensuring fair resource distribution
- Resources are dynamically allocated to users based on their current channel quality and historical data rate, providing a balanced approach
- Balances network throughput with fairness but does not prioritize traffic by user profiles

5G Resource Allocation Problem

Integrating 5G profiles into a single network infrastructure requires the scheduler to meet the unique needs of each user.

Leveraging Machine Learning for Optimal Resource Allocation

Adaptive Learning

Continuously learns and adapts to changing network conditions and user behaviors

Optimization

Aims to optimize overall network performance and user satisfaction by predicting the best allocation strategies

Predictive Analysis

Uses predictive analytics to anticipate future network demands and allocate resources proactively

Complex Patterns

Capable of recognizing complex patterns and correlations that traditional algorithms might miss

Enhancing Decision Making with Reinforcement Learning in MDPs

- Represents the current condition or status of the system
- e.g. Network Load, User queue packet size

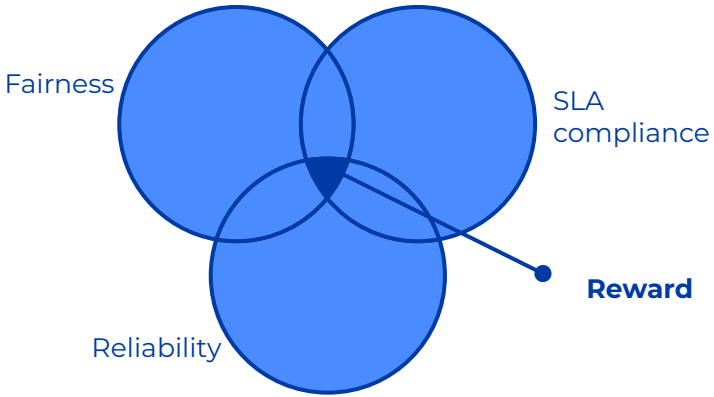
- Actions that can be taken by the scheduler
- e.g. Allocating resource blocks to users

- A feedback signal representing the immediate benefit of taking the action given the state
- E.g. improved data rate, reduced latency

The MDP definition role is crucial

- **State Selection:** Accurate states reflect true network conditions
- **Action Choices:** Effective actions leads to optimized resource utilization
- **Impact of Reward definition**: Properly defined rewards align system behavior with performance goals

There is a trade-off among different elements we want to consider in the reward function; incorporating multiple factors requires balancing them to achieve the best outcome



Drawbacks of Using Reinforcement Learning in 5G

- **Complexity:** Implementing RL algorithms can be complex and require significant computational resources
- **Training time:** RL models often require extensive training, which can be time-consuming and may not adapt quickly to dynamic network conditions
- **Exploration vs Exploitation**: Balancing exploration of new strategies and exploitation of known good strategies can be challenging, potentially leading to suboptimal performance
- **Overfitting:** Risk of overfitting to specific network conditions, leading to poor generalization in varied real-world scenarios

Pywisim

developed by UdelaR

Key Features

- **Characteristics**: Discrete-Event, System-Level Wireless Communications Simulator
- **Purpose:** Model wireless networks for performance estimation in large systems (many nodes).
- **Flexibility:** Supports 5G, LTE, IEEE 802.11, and other networks.
- **Design:**
  - Straightforward Architecture
  - High Modularization
  - Low Coupling

We will use it as a reliable 5G framework

Pywisim and Google Colab

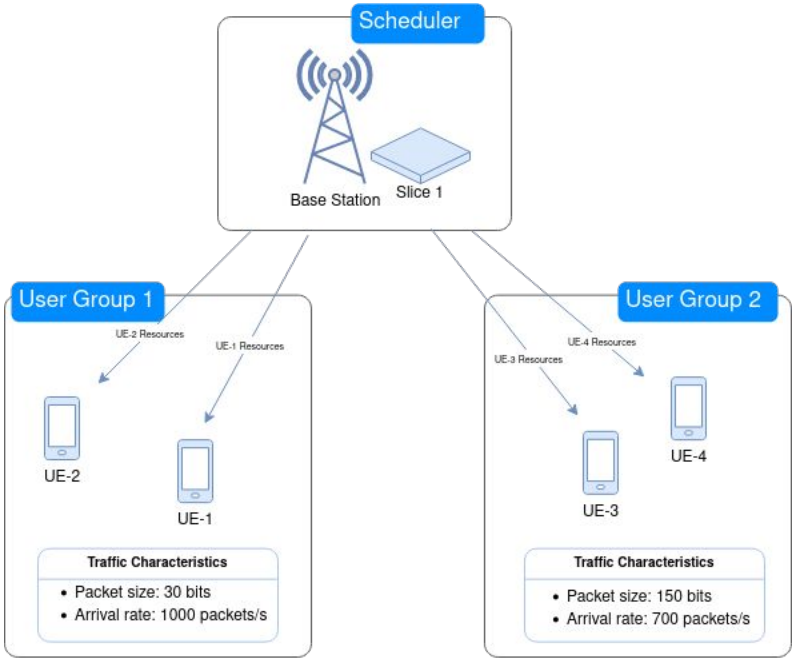
- The simulator source files for the summer schools are located on: https://github.com/lingleslogia/simnet

- The Google Colab prepared for the summer school, where different configurations and scenarios can be verified, can be found at:

Pywisim

Key Components

- Base Station
  - Central hub for connecting UEs
  - Facilitates communication
- Slices
  - Logical network divisions
- User Groups
  - Collections of UEs with similar characteristic and requirements
- UEs (User Equipments)
  - End-users of the network
  - Send and receive data



Creating a simple scenario - base station

Create the base station with the main characteristics:

```
pywisim(['--set-basestation', 'mimo', 'num_slices', 'band', 'name', 'rMCS', 'longcp', 'ul', 'dl', 'time_sim',
'name_scheduler', 'channel_type', 'channel_mode'])
```

- ‘mimo’: mimo options multiuser or single user mimo; type: String, possible values: "MU" or "SU"
- ‘num_slices’: number of slices in the base node, type: int
- 'band': the operation frequency band of the system; type: String
- ‘name': name of the base station, type: String
- ‘rMCS’: Robust Modulation and code scheme or not. , type: Boolean
- ‘longcp’: Long cyclic prefix or not; type: Boolean
- ‘ul’: If the simulation is for uplink or not; type: Boolean
- ‘dl’: If the simulation is for downlink or not; type: Boolean
- 'time_sim': simulation time; type: int
- **'name_scheduler': radio scheduler; type: String, eg. possible values: ‘round robin’, ‘maxcqi’, ‘proportional fair’ or ‘dqn’**
- **'channel_type': wireless channel model, type: String, possible values: ‘random or fixed’ or ‘file’**
- ‘channel_mode’: random

Radio Schedulers and Channel Models

Radio scheduler

- **‘maxcqi’:** Prioritizes users with the best channel quality
- **‘round robin’:** Each user is given an equal amount of resources
- **Proportional Fair**: Maximizes network throughput while ensuring fair resource distribution
- **‘dqn’ or ‘dqnlearner’** will be describe later

Channel models:

- **‘random or fixed’**
- **‘File’.** This option is based on PyWiCh. PyWiCh is a wireless communications channel simulator which has been integrated into PyWiSim as an extension. Pywich repo is: https://github.com/PyWiCh

Creating a simple scenario - UEgroups and Resources

Create the user group:

```
pywisim(['--set-usergroup, ‘index’, ‘name’, ‘par1’, ‘pkt_size’,’ inter_arrival’, ‘trgen_type’, ‘number ues’])
```

- ‘Index’: Id; type: int
- ‘name’: The name of the user group; type: String
- ‘par1’: int
- ‘pkt_size’: number of bits in each packet, type: int
- ‘inter_arrival’: mean interarrival times (ms), type: int
- ‘trgen_type’: “poisson” or “fixed”
- ‘number ues’: The number of the user group; type: int

Create the resources group:

```
pywisim(['--set-resources, 'slice_index', 'namedl', 'nameul', 'n resources downlink', 'n resources uplink', 'sym_slot'])
```

- ‘slice_Index’: Id; type: int
- 'namedl': The name of the downlink resources; type: String; typical: ‘PRB’
- 'nameul': The name of the uplink resources; type: String; typical: ‘PRB’
- ‘n resources downlink’: number of prb for downlink; type: int
- ‘n resources uplink’: number of prb for uplink;; type: int
- ‘sym_slot’: In TDD the resources are shared for downlik and uplink. This is the number of symbols used for downlik. Uplink uses 14 - sym_slot.

Creating a simple scenario - Run the simulation

View the configuration

```
pywisim(['--view-config'])
```

Run de simulation

```
pywisim(['--run-simulation, 'debug'])
```

obss: Debug can be ‘true’ or ‘false’

Example (see google colab)

- pywisim(['--set-basestation', 'SU', '1', 'n258', 'BS-1', 'False', 'False', 'False', 'True', '100', 'round robin', "random or fixed", 'Random'])
    - Creates a base station with one slice and sets the simulation duration to 100 mili seconds. The round robin scheduler is chosen and the channel is random.

- pywisim(['--set-uegroup', '0', 'UG-1', '60', '200', '1', 'fixed', '2'])
    - Creates user group UG-1 with ID 0, containing 2 users, each generating packets of 200 size 200 bits each mili second.

- pywisim(['--set-uegroup', '1', 'UG-2', '60', '300', '1', 'fixed', '2'])
    - Creates user group UG-1 with ID 0, containing 2 users, each generating packets of 300 size 200 bits each mili second.

- pywisim(['--set-resources', '0', 'PRB', 'PRB', '100', '0', '14'])
    - Creates 100 resource blocks to schedule among the users.

- pywisim(['--run-simulation', 'True'])
    - Runs the created environment.

All simulation logs and traces are saved for further processing

They can be found at:
/content/drive/MyDrive/simnet/extensions/sim5gnr/data

See the simulation results

We display the simulation's final results and queue states, show the duration, and provide a summary of total statistics using the '--process-and-graph-data' argument. This argument will lead to a specific subplot, where:

`pywisim(['--process-and-graph-data',  'option0', 'option1', 'Simulation Output Graph'])`

Option0 and option1 are related with:

```
Selection = 0 plots Bits Received
    Subselection = 1 also plots Bits Dropped
Selection = 1 plots Bits Sent
    Subselection = 0 plots Bits Lost
Selection = 2 plots TB Bits per TTI and Resources per TTI
Selection = 3 plots Bits in Queue per TTI, Packets in Queue per TTI,
            and Average Packet Delay in Queue per TTI
Selection = 4 plots TB Bits per UEGroup per TTI and Resources per UEGroup per TTI
Selection = 5 plots Bits in Queue per UEGroup per TTI, Packets in Queue per UEGroup per TTI,
        and Average Packet Delay in Queue per UEGroup per TTI
```

Simple MDP model

System Model

- **Time Division**: Time is divided into discrete time slots (TTIs).
- **System Components**:
  - N User Equipments (UEs)
  - M Resource Blocks (RBs)
- **Resource Allocation**:
  - At each time slot, each RB is allocated to one UE.
- **Goal:** minimize the delay by minimizing the queue size of the UEs.

Markov Decision Process (MDP)

- **State**:
  - UEs Buffer sizes of the UEs
- **Action**:
  - Allocation value a corresponding to the user's index which is going to have the resources.
- **Cost Function**:
  - The sum of UEs Buffer sizes.

Dqn example, when we already have the learned model

We have built a DQN algorithm taking the MDP model, that can be found at

/content/drive/MyDrive/simnet/models/scheduler/dqnscheduler

Now, creating a new scenario with the DQN scheduler using the following parameters::

```
pywisim(['--set-basestation', 'SU', '1', 'n258', 'BS-1', 'False', 'False', 'False', 'True', '100', 'dqn', "random or fixed", 'Random'])

pywisim(['--set-uegroup', '0', 'UG-1', '60', '300', '1', 'fixed', '2'])

pywisim(['--set-uegroup', '1', 'UG-2', '60', '300', '5', 'fixed', '2'])

pywisim(['--set-resources', '0', 'PRB', 'PRB', '100', '0', '14'])

pywisim(['--view-config'])
```

Dqn example, how to learn a new model by changing the scenario?

- If you feel curious and want to learn a new DQN model, simply change the scheduler type to 'dqnlearn' and set up a new environment. This requires a longer learning period. You can try:

```
pywisim(['--set-basestation', 'SU', '1', 'n258', 'BS-1', 'False', 'False', 'False', 'True', '1000', 'dqnlearn', "random or fixed", 'Random'])

pywisim(['--set-uegroup', '0', 'UG-1', '60', '300', '1', 'fixed', '2'])

pywisim(['--set-uegroup', '1', 'UG-2', '60', '300', '5', 'fixed', '2'])

pywisim(['--set-resources', '0', 'PRB', 'PRB', '100', '0', '14'])
```

Dqn example, how to learn a new model by changing the scenario?

- The dqn scheduler learner module can be found at:
  /content/drive/MyDrive/simnet/models/scheduler/dqnscheduler/dqnschedulerlearner.py
- The dqnschedulerlearner learns through trial and error, and saves the new model upon completion.
- To test the model, change the scheduler to 'dqn' and adjust the variable model_path by editing the script at /content/drive/MyDrive/simnet/models/scheduler/dqnscheduler/dqnscheduler.py.

  Specifically, change line 32 from:

  '/content/drive/MyDrive/simnet/extensions/sim5gnr/gui/models/dqn_model_sc.pth'

  To

  '/content/drive/MyDrive/simnet/extensions/sim5gnr/gui/models/dqn_model.pth'.
- Additionally, you can experiment by changing any hyperparameter in dqnschedulerlearner.py.

Hands-On