Scheduling for Delay Constrained Throughput Maximization in 5G NR using Reinforcement Learning

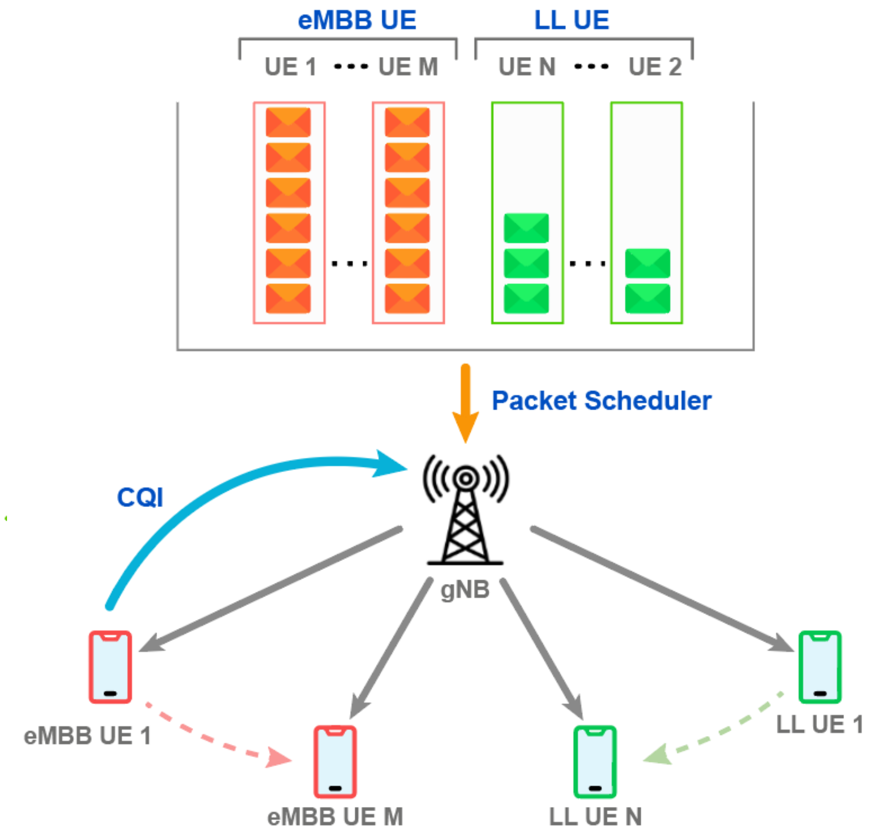
Applicable Release: NetSim v14.1 or higher
Applicable Version(s): NetSim Standard
Project download link: https://github.com/NetSim-TETCOS/Delay_Scheduling_in_5G_RL_v14.1/archive/refs/heads/main.zip

The URL has the exported NetSim scenario for the example used in this document and the python script to run the reinforcement learning simulation

Please refer to slide 12 for the steps to run the reinforcement learning (RL) algorithm.

Problem Statement

- Delay aware 5G scheduling
  - N low latency (delay constrained) UEs, with arrival rates λ_i
  - M high throughput (eMBB) UEs, with full buffer backlog traffic
  - Each UE sees a different transmission channel due to distance-based pathloss and time-varying Rayleigh fading
- Packets are queued for transmission and at every slot and the scheduler chooses UEs to serve based on:
  - Queue backlog at the low latency UEs
  - Current channel states of all UEs i.e., MCS is selected based on received SINR
- Goal: Maximize the sum throughput of eMBB UEs while meeting the delay constraints of the low latency UEs
- This is the classical *opportunistic scheduling* problem without delay constraints; the delay constrains make the problem a much more complex *Markov Decision Problem (MDP)*



The optimization problem and applying RL

Maximize the average sum throughput of M eMBB UEs

$$max\left(\lim_{K\to\infty}\frac{1}{K}\mathbb{E}\sum_{k=0}^{K-1}\sum_{i=1}^{M}R_i\,(k)\right)$$

such that for $j \in N$, the Low latency UEs, we have from Little's Law

$$\lim_{k\to\infty}\frac{1}{K}\mathbb{E}\sum_{k=0}^{K-1}Q_j(k) \le \;\lambda_j d_j$$

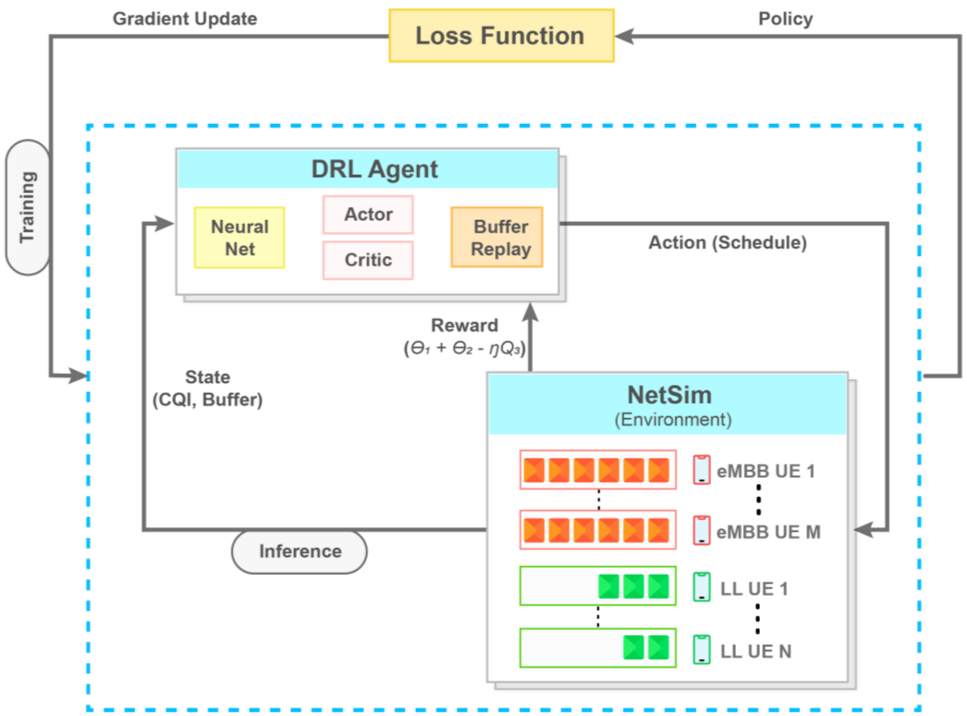
where d_j is the delay bound. Consider the Lagrangian, $\forall\,\pi \in \Pi$, with $\eta_j \ge 0$

$$L\big(\pi,(\mu_1,\mu_2,\dots\dots,\mu_N)\big)\;=\sum_{i=1}^{M}\bar{R}_i^{(\pi)}+\sum_{j=1}^{N}\eta_j\left(\lambda_j d_j-\bar{Q}_j^{(\pi)}\right)$$

And choose the optimal policy $\pi^*(n)$ maximises $L(\pi,\eta)$

$$\sum_{i=1}^{M}\bar{R}_i^{(\pi^*(n))}\;+\;\sum_{j=1}^{N}\eta_j\left(\lambda_j d_j-\bar{Q}_j^{(\pi^*(n))}\right)$$

The Lagrangian relaxation method penalizes violations of inequality constraints using the Lagrange Multiplier (η)



- The agent can be a “basic” user developed RL algorithm, e.g., Tabular Q learning, or
- An “advanced” RL algorithms by interfacing with OpenAI Gym, e.g., PPO, A2C which use deep neural networks

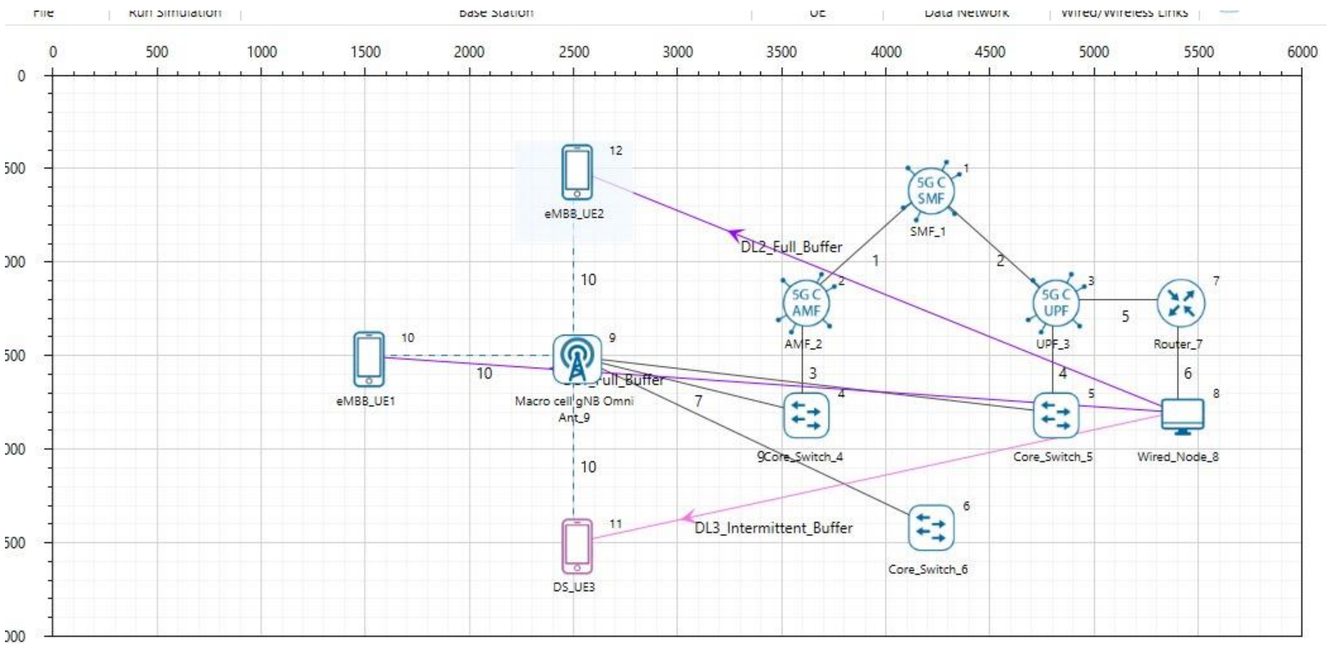
Scenario in NetSim and the reward function

- **UEs**: 2 eMBB UEs and 1 Low Latency (LL) UE
- **States**:
  - CQI of all nodes and queue length at LL node
  - NetSim passes states to RL algorithm
- **Actions:** Fractional allocation of resources per UE per slot.
  - Action constraint: sum of allocation fractions should be 1, which represents total PRBs in a slot
  - Scheduling:{(0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 0.5, 0.5), (0.5, 0. 0.5), (0.5, 0.5, 0)},
  - These 6 combinations were chosen to reduce the action space; any set of fractional combinations that sum to 1 can be set
  - RL returns actions
- **Reward** : $R = \theta_1 + \theta_2 - \eta \cdot Q_3$
  - Units: θ_1, θ_2 in Mbps, Q_3 in Bytes
  - NetSim passes reward and next state

System Parameters	
UEs	2 eMBB UEs, 1 LL UE
gNB	1 gNB serving 3 UEs
Band and BW	n78; 100 MHz
eMBB Traffic model	Full buffer UE1, UE2
LL Traffic Model	2 Mbps Download
Pathloss Model	Log Distance
Pathloss Exponent	3
Fading Model	Rayleigh
Coherence Time	30 ms
MCS	Chosen for Zero BLER
Scheduling	RL based at each TTI

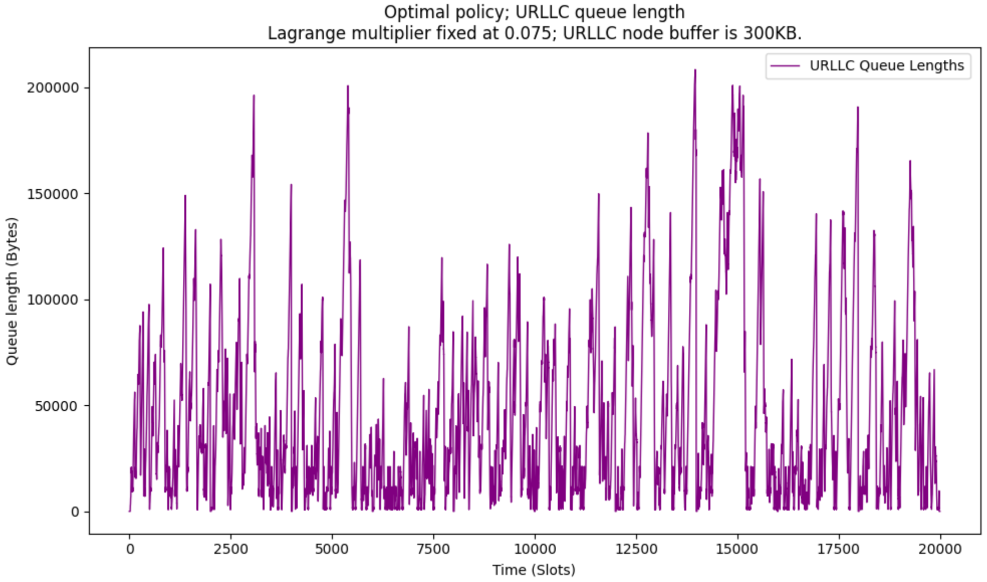
NetSim Model

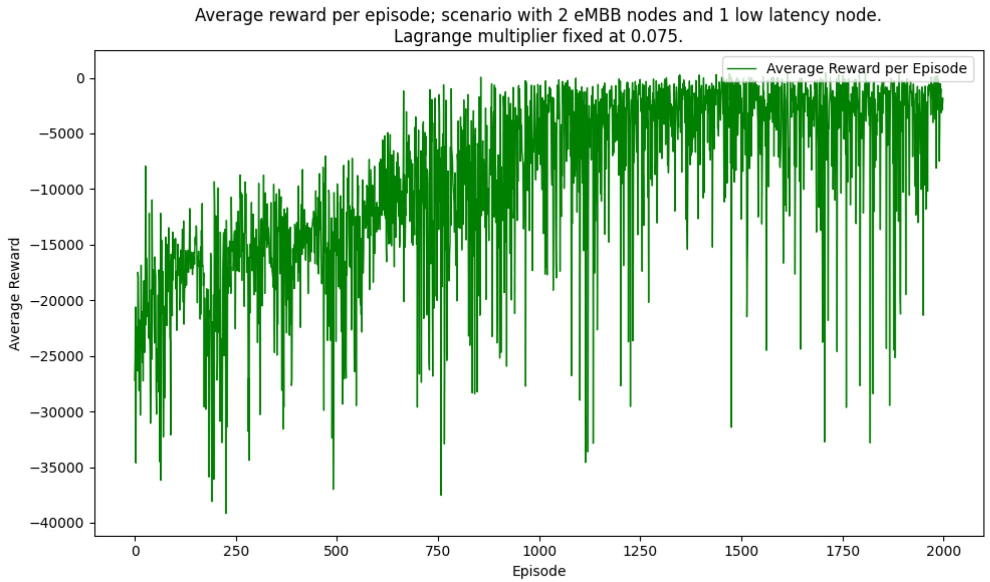
- 2 EMBB nodes and 1 Delay Sensitive node
  - 1 gNB serving all three nodes
  - Band: n78, Bandwidth:100 MHz
- Traffic model
  - eMBB Node 1: Full Buffer
  - eMBB Node 2: Full buffer
  - DS node: 2.07 Mbps download
- Pathloss values:
  - Log Distance
  - Pathloss exponent: 3
- Rayleigh fading
  - Coherence time: 10 ms
- Antenna counts $1T \times 1R$ at UE and gNB
- MCS chosen for zero BLER
- Scheduling: RL based

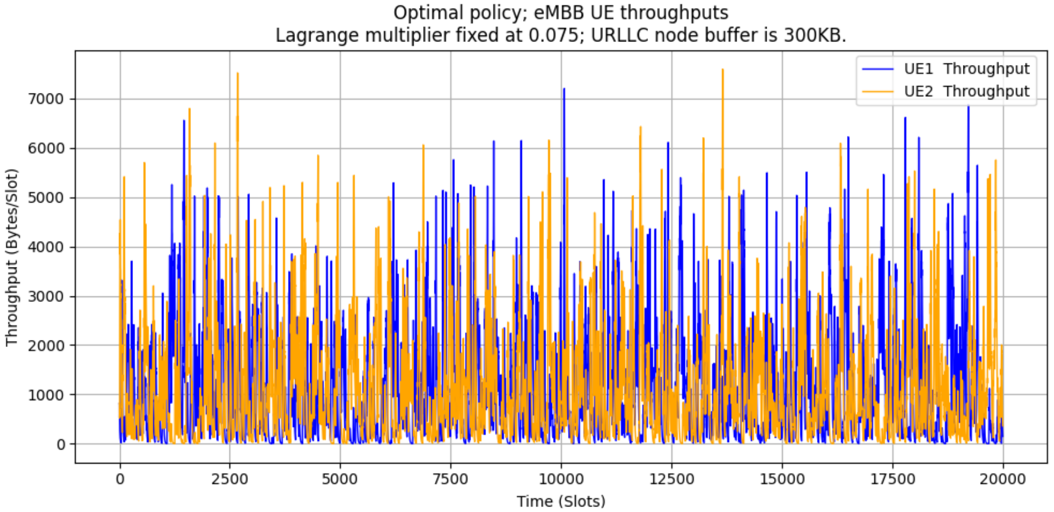


Queue length, Throughputs and the RL Training Curve.

Lagrange Multiplier set to 0.075



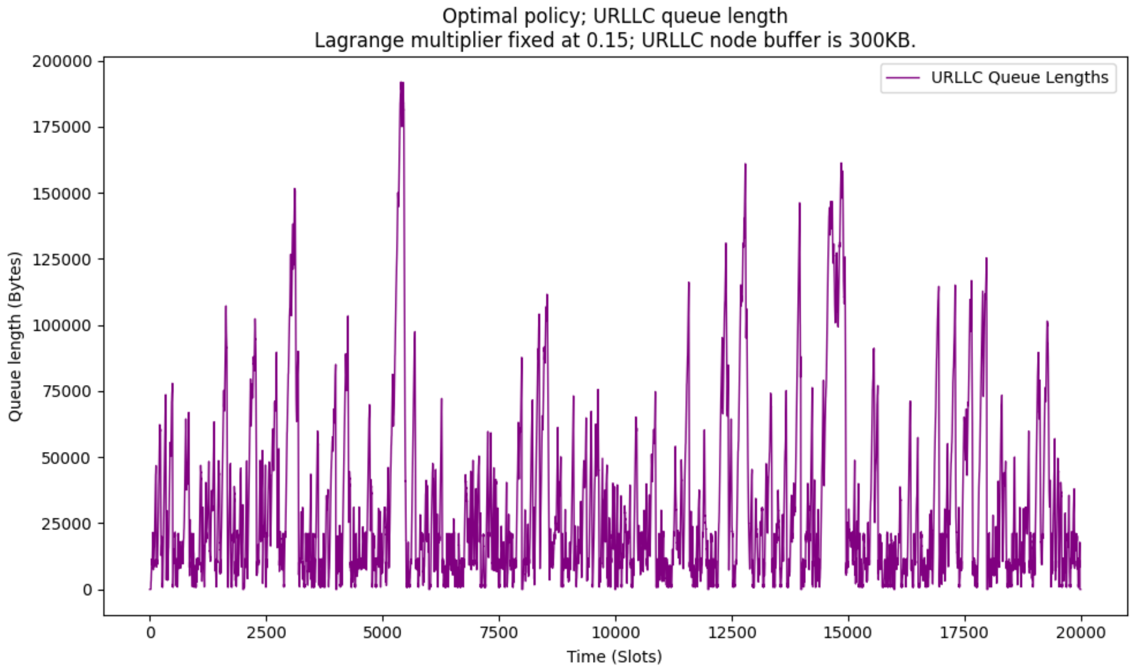


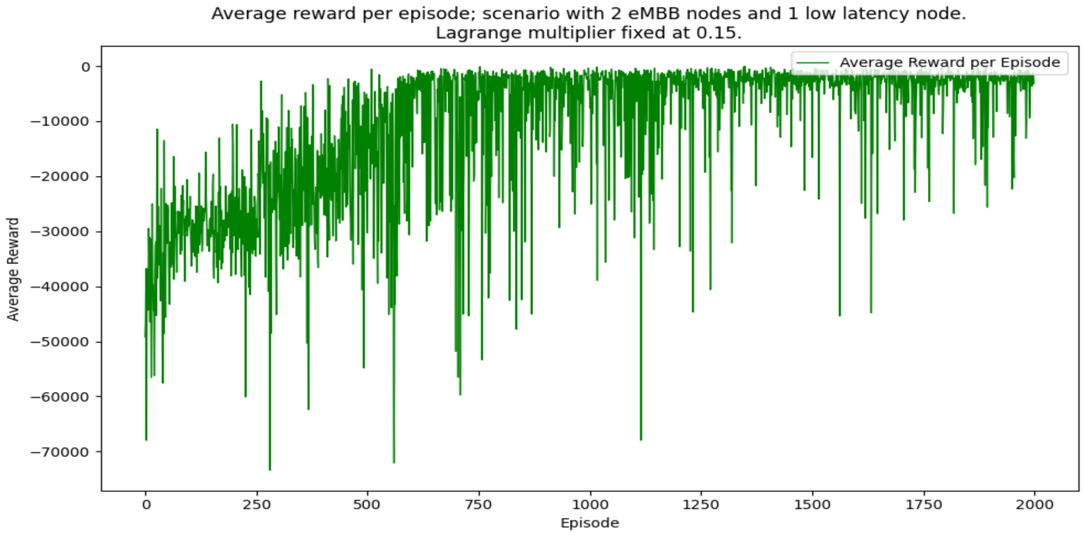


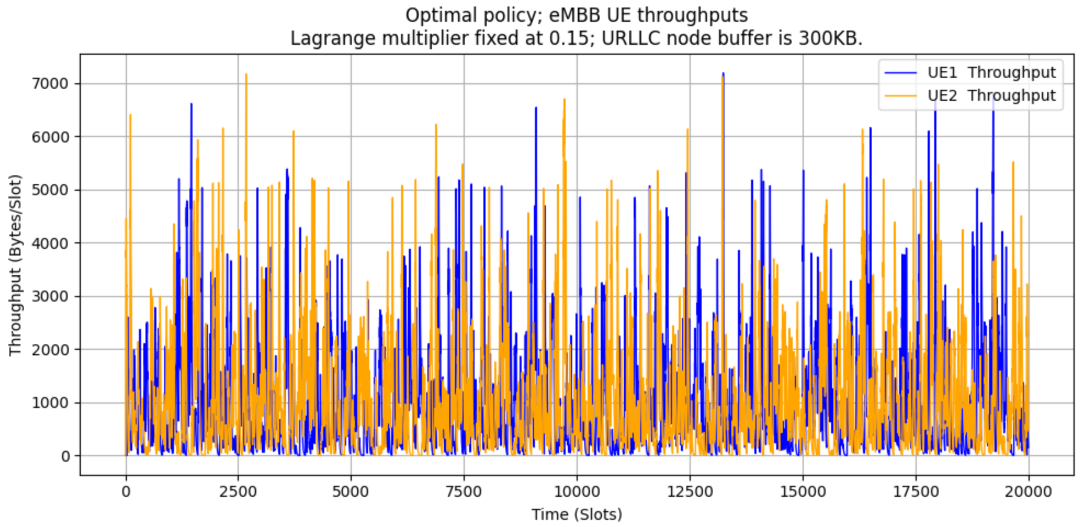
Avg. Queue Length = 29314 B; Avg. Delay = 32.7 ms;
Avg. Throughput per node = 16.62 Mbps

Queue length, Throughputs and the RL Training Curve.

Lagrange Multiplier set to 0.15



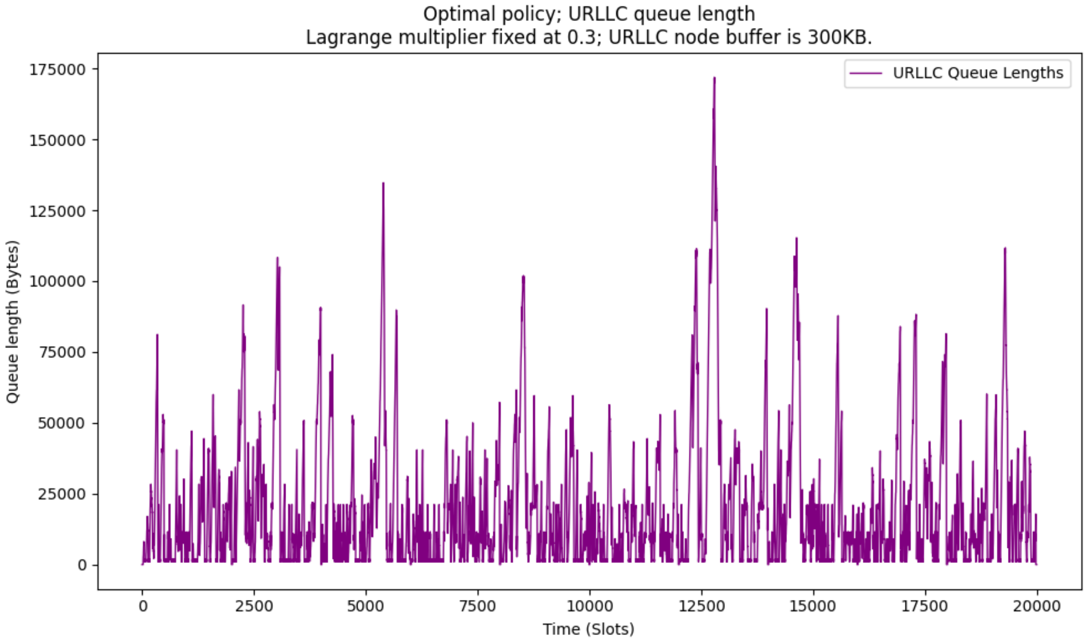


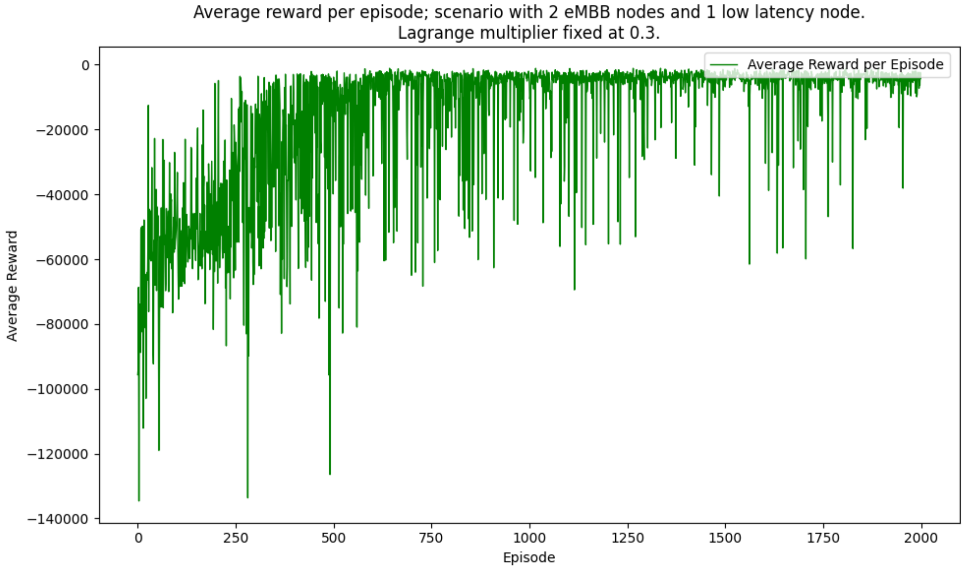


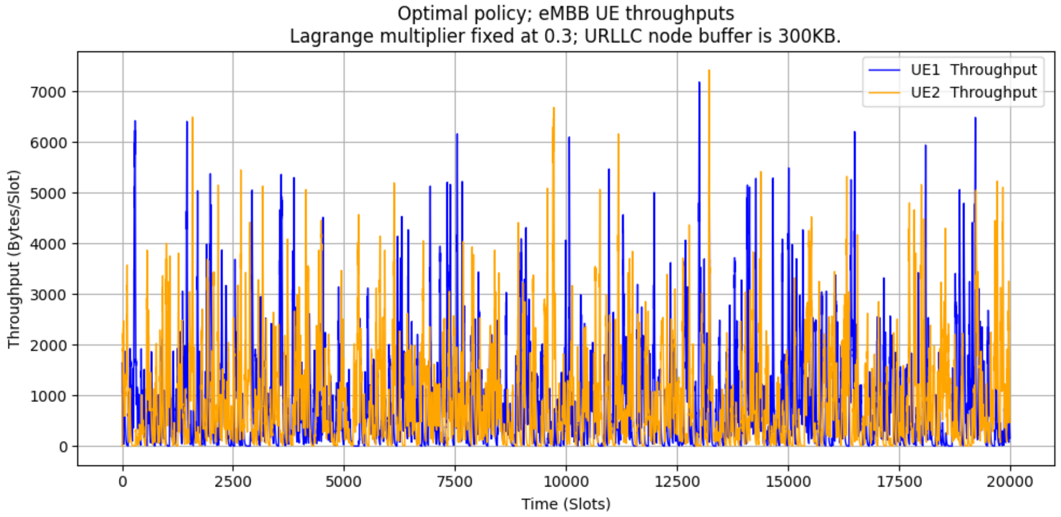
Avg. Queue Length = 31566.81 B; Avg. Delay = 11.78 ms;
Avg. Throughput per node = 16.66 Mbps

Queue length, Throughputs and the RL Training Curve.

Lagrange Multiplier set to 0.30







Avg. Queue Length = 20045.46 B; Avg. Delay = 15.73 ms;
Avg. Throughput per node = 14.14 Mbps

How close are we to the optimum? Fixed LM

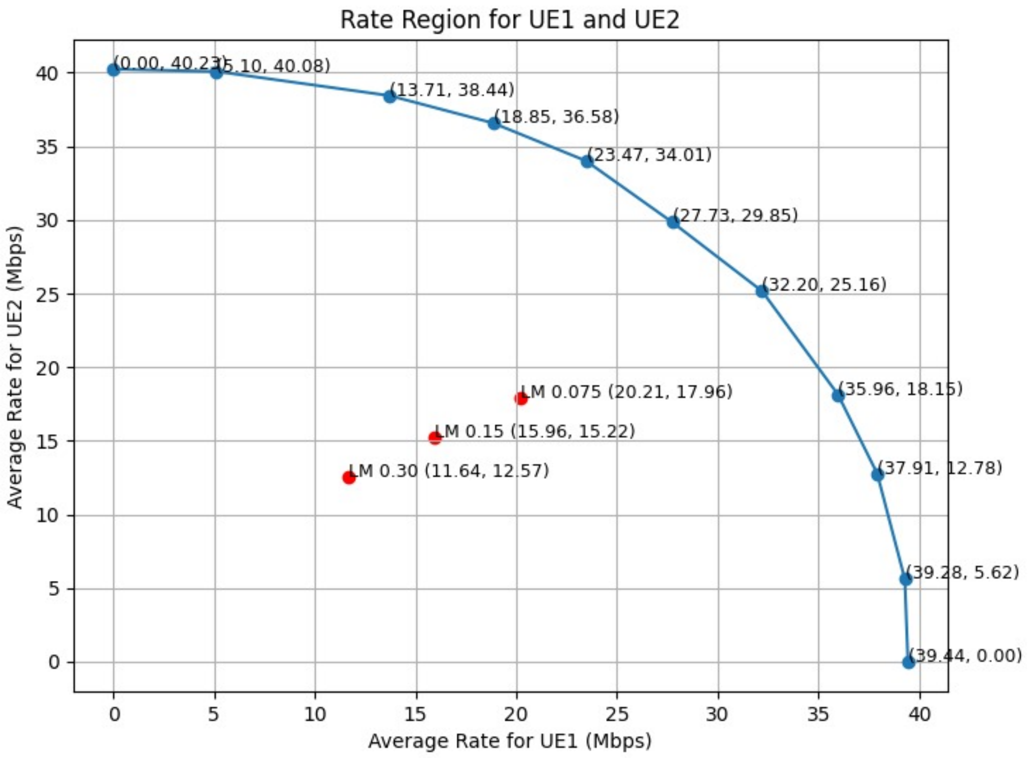


Fig: Rate region operating points from NetSim. Delay constrained operating points in red.

Fixed LM	Avg. eMBB1 Thpt. (Mbps)	Avg. eMBB2 Thpt. (Mbps)	Avg. URLLC Q length (Bytes)
0.075	18.27	18.05	46629.64
0.15	15.44	16.66	31566.81
0.30	13.08	13.02	20045.46

- The delay constraint moves the throughputs of the two UEs into the interior.
- Higher the Lagrange multiplier, further interior the throughputs.

NetSim Python Interfacing

Introduction to Python C Socket Interfacing:

- Python facilitates seamless integration with C for socket programming tasks.
- Python uses the socket module to interact with C side functions.

Python Code:

- Python creates a client socket in the host machine and connects with the C side server
- Key functions include:

```
def NETSIM_interface_queue_throughputs(action):
```

- Sends the allocations to NetSim with a header
- Receives acknowledgement integer value from NetSim
- Sends a request value to NetSim to receive the updated queue length and the reward
- Receives the next state and the reward from NetSim

NetSim Python Interfacing

NetSim C code key functions

void init_waiting_struct_socket1() :

- Initializes the Winsock library and calls the listenForPython() function

bool listenForPython() :

- Resolves the server address and port
- Creates a socket for connecting to the server
- Sets up the TCP listening socket
- Waits for client socket connection

void handle_Send_Receive(struct Queue_Length_Reward* Param1, int* action)

- Receives the message type from Python
- If message type is to “receive actions”, receives the allocations from Python and sends back an acknowledgement message
- If message type is to “send queue length and rewards”, sends back the state and reward back to Python using the send_Queue_Length_Reward_at_Time_Step() function

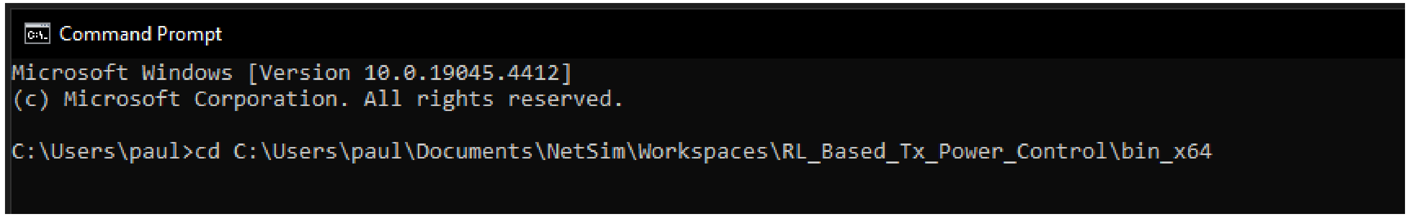
void send_Queue_Length_Reward_at_Time_Step () :

- sends the list of updated queue length of the DS node, reward, list of throughputs and sinrs received from the LTENR project in NetSim back to python

Appendix

How to run the RL simulation?

- Download the project from Github link provided in slide 1.
- Follow the instructions provided in the following link to setup the project in NetSim
  - https://support.tetcos.com/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects
- For the RL simulation, we first need to run NetSim using the command line interface(CLI)
- Open the Run menu with Windows Key + R, then type "cmd." Press "Enter" to open Command Prompt
- Note the Experiment path. Experiment path is the current workspace location of the NetSim that you want to run. The default path will be something like "***C:\Users\PC\Documents\NetSim\Workspaces\<experiment folder>***" for 64-bit.
- Change the directory to the application path using the following command, below is an example command
  >cd \<app path>



How to run the RL simulation ?

- Update the **iopath**, **apppath**, and license path or server IP address in the auto_simulate.py script:

  - **iopath**: Set this to the path of the current scenario, e.g., Max_throughput_delay_scheduling.
  - The **License** should be configured to either:
    1. The IP address of the system hosting the NetSim license server, or
    2. The path to the license file (ensure to uncomment the section for the license file). apppath: Set this to the bin_x64 folder of the workspace where NetSimCore.exe is located.

- Ensure that all necessary files and the Python script are in the experiment folder where configuration.netsim is present.

- Backup the configuration.netsim file. Save a copy as input.xml for continuous updates with seed values.

- Run the auto_simulate.py script.

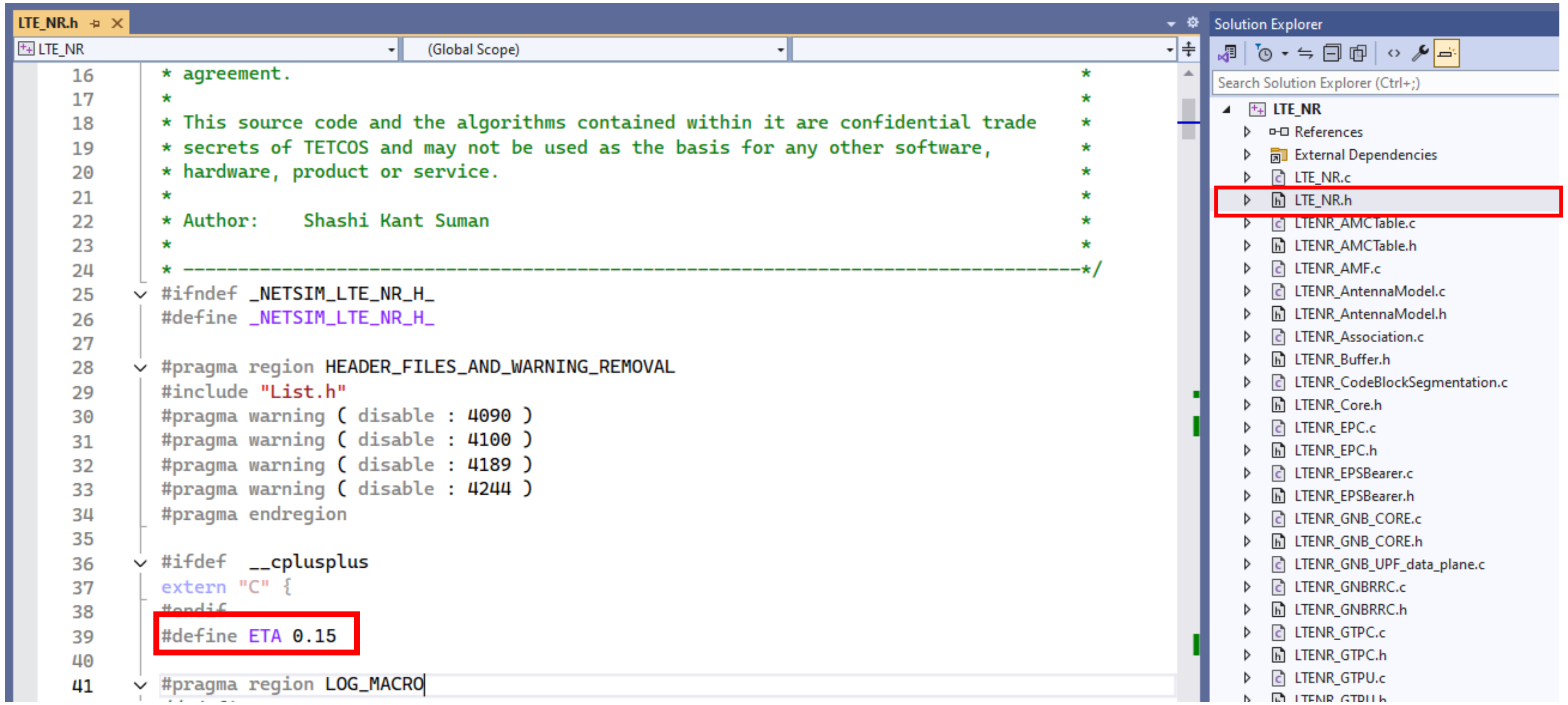
- Open a new command prompt window. Change the directory to where the Python file and requirements.txt are located.

- To run Tabular Q Learning:
  - Type python <filename>, where <filename> is RL_delay_scheduling, and press Enter.
  - The script will prompt for the number of episodes. Enter the default value, 2000, and press Enter to start the simulation.

How to run the RL simulation ?

- To change the number of episodes, we need to make two changes. First, we need to change the input we give to the python script. Second, we need to edit the looping command which we use to run NetSim. Edit the **<NUM_EPISODES>** variable in the command
- For Tabular Q Learning:
  - Upon running the simulation, the python script will create two folders, named “plots” and “logs” where it will save the result plots and log files, respectively. These folders will be created in the same working directory as the python script
  - Close the plots after viewing them, to allow them to be saved.
  - In the “logs” folder, all the log files will be saved which can be used for debugging

How to run the RL simulation ?



- To modify the Lagrange Multiplier in NetSim:
  1. Open **NetSim Source code > LTE_NR Project > LTE_NR.h**.
  2. Update the **ETA** value to the desired value (e.g., 0.075, 0.15, 0.30) as needed.
  3. Rebuild the LTE_NR project then run the RL script and NetSim