# A Comparison of Proportional Fair And Reinforcement Learning Based Wireless Resource Schedulers

**Sandeep K. Routray**
sroutray@iitk.ac.in

**Abstract**

Long Term Evolution (LTE) makes use of Orthogonal Frequency Division Multiplexing (OFDM) to achieve high downlink data rates. Based on channel conditions of users, a resource scheduler can use dynamic resource allocation to provide best resource to users and optimize system performance. Proportional Fair (PF) scheduling is a popular resource allocation algorithm that tries to maximize total throughput while providing all users at least a minimal level of service. Recent advances in reinforcement learning (RL) has encouraged efforts to formulate resource allocation as a RL problem. In this study, Deep Deterministic Policy Gradient technique of RL is used to design RL-based resource scheduler and its performance is compared with that of the PF scheduler.

## 1 Introduction

### 1.1 OFDMA

The peak throughput in a wireless system depends on the product of system bandwidth and peak spectral efficiency. Increasing system bandwidth to increase throughput becomes inconvenient as it complicates the channel estimation and equalization procedure as shown in Figure 1. OFDM emerged as an alternative solution that focuses on increasing spectral efficiency. OFDM is a multi-carrier transmission technique that divides a higher data rate signals into several lower data rate signals, which are then modulated into different frequency bands and transmitted with orthogonal subcarriers. Guard bands are used between sub-carriers to avoid interference. Orthogonality allows subcarriers to overlap and save bandwidth. Thus, increasing spectral efficiency and achieving high throughput. Orthogonal Frequency Division Multiple *Access* (OFDM*A*) is an OFDM system that also allows multiple users to share frequency resources at the same time.

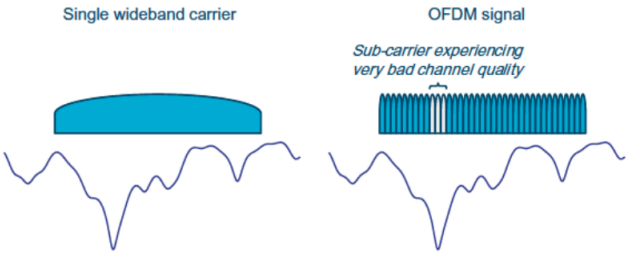


Figure 1: Single Carrier Vs OFDM System

### 1.2 The LTE System

LTE system makes use of OFDMA technique to achieve high throughput. Physical resource in LTE system is defined as a time-frequency grid which is assigned to User Equipments (UEs) for data transmission. In LTE, transmission is organized into frames of 10ms. Each frame consists of 10 subframes. Each subframe is further divided into two time

slots, each containing 7 OFDM symbols. Each subframe of 1ms is also known as a Transmission Time Interval (TTI). The LTE frame structure is shown in Figure 2.

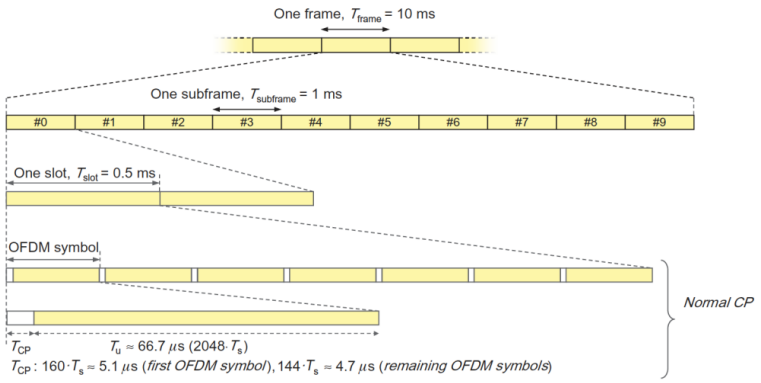


Figure 2: The LTE Frame Structure

LTE system defines a resource block (RB) as a unit consisting of 12 consecutive subcarriers and two time slots containing 14 OFDM symbols. The bandwidth of a subcarrier is 15kHz and a RB has twelve subcarriers. The total bandwidth of a resource block is 180kHz. Each LTE OFDM symbol saves 10% of the bandwidth as guard band. Assuming that there is 20MHz allocated bandwidth, the effective bandwidth is only 18MHz. Thus, there are 100 RBs to be allocated in a TTI. The system bandwidths and OFDM parameters supported by 3GPP LTE/LTE-A is shown in Figure 3

| Range | | | | | | |
|---|---|---|---|---|---|---|
| Channel bandwidth (MHz) | 1.4 | 3 | 5 | 10 | 15 | 20 |
| Resource block numbers (RBs) | 6 | 15 RBs | 25 RBs | 50 RBs | 75 RBs | 100 RBs |
| Number of subcarriers | 72 | 180 | 300 | 600 | 900 | 1200 |
| FFT size | 128 | 256 | 512 | 1024 | 1536 | 2048 |
| Subcarrier spacing | 15 kHz | | | | | |
| OFDM Samples per subframe | 7 for normal CP | | | | | |
| Frame duration | 10 ms | | | | | |

Figure 3: LTE/LTE-A Resource Division

### 1.3 Channel Quality Indicator

OFDMA also enables the use of Adaptive Modulation and Coding for *each* subchannel. Depending on subchannel conditions, different modulation schemes and code rate can be used to provide greater coverage, higher data rate and spectral efficiency. Channel Quality Indicator (CQI) is a measurement standard of channel quality. It also indicates the optimal transmission rate supported by the UE. A high CQI value indicates good channel quality which can support higher modulation techniques like QAM for high transmission rates while maintaining acceptable bit error rates. A low CQI value indicates poor channel quality, hence need to use lower modulation techniques like QPSK with low transmission rates to maintain acceptable bit error rates. LTE uses a 4-bit CQI system as shown in Figure 4.

Let the code rate be $r$. Then, efficiency = $r \log(M)$ bits/channel, where $M$ is the modulation index. If $n$ is the number of RBs allocated in a TTI, the instantaneous throughput is given as

$$\text{instantaneous throughput (bits/s)} = 1000 \times \text{efficiency} \times n \times 14 \text{ OFDM symbols} \times 12 \text{ subcarriers}$$

### 1.4 Resource Scheduler

A base station (BS) resource scheduler is responsible for controlling the uplink and downlink resource allocation. Downlink resource allocation is considered in this study. Usually, a scheduler tries to maximize the overall system throughput. An obvious way to achieve maximum throughput is to allocate all RBs in a TTI to the UE with the best channel condition. However, such a greedy strategy would not be practical as this would starve UEs with less favourable

| CQI index | modulation | code rate x 1024 | efficiency |
|---|---|---|---|
| 0 | out of range | | |
| 1 | QPSK | 78 | 0.1523 |
| 2 | QPSK | 193 | 0.3770 |
| 3 | QPSK | 449 | 0.8770 |
| 4 | 16QAM | 378 | 1.4766 |
| 5 | 16QAM | 490 | 1.9141 |
| 6 | 16QAM | 616 | 2.4063 |
| 7 | 64QAM | 466 | 2.7305 |
| 8 | 64QAM | 567 | 3.3223 |
| 9 | 64QAM | 666 | 3.9023 |
| 10 | 64QAM | 772 | 4.5234 |
| 11 | 64QAM | 873 | 5.1152 |
| 12 | 256QAM | 711 | 5.5547 |
| 13 | 256QAM | 797 | 6.2266 |
| 14 | 256QAM | 885 | 6.9141 |
| 15 | 256QAM | 948 | 7.4063 |

Figure 4: 4-Bit CQI System

channel quality. On the other hand, an allocation algorithm that gives every UE equal opportunity may cause inefficient transmission to those UEs, which have to transmit large amount traffic with strict Quality of Service (QoS) requirements. A good scheduler would use dynamic RB allocation by taking CQI into account to provide the best resource to an UE and optimize system performance. It needs to strike an optimal trade-off between maximizing throughput and ensuring fairness. In this study, two schedulers are discussed - a proportional fair scheduler and a reinforcement learning based scheduler.

## 2 Proportional Fair Scheduler

A Proportional Fair (PF) scheduler tries to maximize total throughput while providing all users at least a minimal level of service. Thus, it balances between throughput and fairness among all the UEs. The PF scheduler effectively reduces variations in user bit rates.

Mathematically, a PF scheduler $P$ satisfies the following constrained optimization problem [2]

$$P = \arg\max_{\boldsymbol{\rho}} \sum_{k=1}^{K} \log\left(1 + \frac{\sum_{n=1}^{N} \rho_{k,n} r_{k,n}(t)}{(t_c - 1) T_k(t)}\right) \quad \text{subject to constraints} \tag{1}$$

$$C1 : \rho_{k,n} \geq 0 \text{ for } k = 1, ..., K \text{ and } n = 1, ..., N$$

$$C2 : \sum_{k=1}^{K} \rho_{k,n} = 1 \text{ for } n = 1, ..., N$$

Here,

- $r_{k,n}$ is the instantaneous throughput for UE $k$ on channel $n$.
- $\rho_{k,n}$ is the assignment indicator variable. If a channel $n$ is assigned to UE $k'$, then $\rho_{k',n} = 1$ and $\rho_{k,n} = 0$ for $k \neq k'$.
- $T_k$ is the average throughput of UE $k$ updated using an exponentially weighted low-pass filter over a window size of $t_c$. Including $T_k$ in optimization Equation 2 provides proportional fairness.

$$T_k(t+1) = \left(1 - \frac{1}{t_c}\right) T_k(t) + \frac{1}{t_c} \sum_{n=1}^{N} \rho_{k,n} r_{k,n}(t) \tag{2}$$

The Lagrangian for the optimization problem given by 2 can be written as

$$L(\rho_{k,n}, \mu_{k,n}, \lambda_n) = \sum_{k=1}^{K} \log\left(1 + \frac{\sum_{n=1}^{N} \rho_{k,n} r_{k,n}(t)}{(t_c - 1) T_k(t)}\right) + \sum_{k=1}^{K} \sum_{n=1}^{N} \mu_{k,n} \rho_{k,n} - \sum_{n=1}^{N} \lambda_n \left(\sum_{k=1}^{K} \rho_{k,n} - 1\right) \tag{3}$$

where $\mu_{k,n}, \lambda_n$ are non-negative Lagrange multipliers. The following relations can be established KKT conditions.

$$\frac{\partial L}{\partial \rho_{k,n}} = \frac{r_{k,n}(t)}{(t_c - 1) T_k(t) + \sum_{n=1}^{N} \rho_{k,n} r_{k,n}(t)} + \mu_{k,n} - \lambda_n = 0 \tag{4}$$

$$\mu_{k,n}\rho_{k,n} = 0 \tag{5}$$

$$\lambda_n(\sum_{k=1}^{K}\rho_{k,n} - 1) \tag{6}$$

Eliminating $\mu_{k,n}$ from Equation 4 and Equation 5

$$\frac{r_{k,n}(t)}{(t_c-1)T_k(t) + \sum_{n=1}^{N}\rho_{k,n}r_{k,n}(t)} - \lambda_n \leq 0 \tag{7}$$

$$\rho_{k,n}\left(\frac{r_{k,n}(t)}{(t_c-1)T_k(t) + \sum_{n=1}^{N}\rho_{k,n}r_{k,n}(t)} - \lambda_n\right) = 0 \tag{8}$$

From Equation 7 and Equation 8, if subcarrier $n$ is not allocated to UE $k$ (that is $\rho_{k,n} = 0$), then $\frac{r_{k,n}(t)}{(t_c-1)T_k(t)+\sum_{n=1}^{N}\rho_{k,n}r_{k,n}(t)} - \lambda_n \leq 0$, and if subcarrier $n$ is allocated to UE $k$ (that is $\rho_{k,n} > 0$), then $\frac{r_{k,n}(t)}{(t_c-1)T_k(t)+\sum_{n=1}^{N}\rho_{k,n}r_{k,n}(t)} - \lambda_n = 0$. Based on this observation, subcarrier $n$ should be allocated to UE $k^*$ given by

$$k^* = \arg\max_k \left(\frac{r_{k,n}(t)}{(t_c-1)T_k(t) + \sum_{n=1}^{N}\rho_{k,n}r_{k,n}(t)}\right) \tag{9}$$

Using Equation 9, the following utility function is constructed based on which allocation is done.

$$U_{k,n}(t) = \frac{r_{k,n}(t)}{((t_c-1)T_k(t) + \sum_{n=1}^{N}\rho_{k,n}r_{k,n}(t))^{\alpha}} \tag{10}$$

Here, $\alpha$ represents the weightage on average throughput while allocation. Setting $\alpha = 0$ gives us a scheduler which allocates greedily based on instantaneous UE rates but leads to highly unfair allocation among UEs. Increasing $\alpha$ increases fairness but reduces overall system throughput.

Thus, the scheduling algorithm can be summarized as follows.

1. For each unallocated subcarrier $n$ and each UE $k$, calculate the utility value $U_{k,n}(t)$.
2. Choose $(k^*, n^*) = \arg\max_{k,n} U_{k,n}(t)$ and allocate subcarrier $n^*$ to UE $k^*$. In the event of ties, randomly assign among the possibilities.
3. Repeat steps 1 and 2 untill all subcarriers are allocated and update $T_k$ as per Equation 2.

## 3 Reinforcement Learning Based Scheduler

### 3.1 Background

Reinforcement Learning (RL) is an area of machine learning that addresses sequential decision making with an objective to maximize the notion of cumulative reward while interacting with the unknown environment. Some basic definitions in RL are given below.

- $s_t, a_t, r_t$: Denote the state, action, and reward at time step $t$ of one trajectory.
- $\tau = (s_0, a_0, s_1, a_1, ...)$: Denotes the trajectory followed by the agent. It is a sequence of states and actions of the agent as it interacts with the world.
- $\gamma$: Denotes the discount factor. It characterizes the penalty due to uncertainty of future rewards. $\gamma \in (0, 1]$.
- $R(\tau) = \sum_{t=0}^{\infty}\gamma^t r_t$: Denotes the cumulative reward achieved by following a trajectory $\tau$.
- $\pi(s, a)$ or $\mu(s)$: Denotes the policy. It is a rule used by an agent to decide what actions to take in a given state. $\pi(s, a)$ denotes a stochastic policy whereas $\mu(s)$ denotes a deterministic policy.
- $Q^{\pi}(s, a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a]$: Denotes the on-policy action-value function, also called the Q-function. It gives the expected return if the agent starts in a state $s$, takes an arbitrary action $a$, and then forever after acts according to policy $\pi$.

- $Q^*(s,a) = \max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a]$: Denotes the optimal action-value function. It gives the expected return if the agent starts in a state $s$, take an arbitrary action $a$, and then forever after acts according to the optimal policy in the environment.

Mathematically, RL models the problem of sequential decision making as as a Markov Decision Process (MDP). At each discrete time step t, the agent observes some representation of the environment state $s_t$ from state space $\mathcal{S}$, and then selects an action $a_t$ from the action set $\mathcal{A}$. The goal of learning is to find the optimal action $a^*(s)$ that maximizes the expected accumulative rewards from any initial state $s$. If the optimal action-value function $Q^*(s,a)$ is known, then in any given state, the optimal action $a^*(s)$ can be found by solving

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{11}$$

The optimization problem in 11 can be easily solved for finite and discrete action space. But when the action space is continuous, exhaustively evaluating the action space becomes infeasible, and solving the optimization problem is highly non-trivial. Deep Deterministic Policy Gradient (DDPG) [1] is RL technique that is used to address such problems.

### 3.2 Deep Deterministic Policy Gradient

DDPG is a RL algorithm that learns the Q-function and the policy concurrently. DDPG interleaves learning an approximator to $Q^*(s,a)$ with learning an approximator to $a^*(s)$, and it does so in a way which is specifically adapted for environments with continuous action spaces. Because the action space is continuous, the function $Q^*(s,a)$ can be assumed to be differentiable with respect to the action argument. This paves way to set up an efficient, gradient-based learning rule for a policy $\mu(s)$ which exploits that fact. Then, instead of running an expensive optimization subroutine each time to solve optimization problem in Equation 11, it can be approximated with $\max_a Q(s,a) \approx Q(s,\mu(s))$.

The approximator of the Q-function is denoted as $Q_\phi(s,a)$, which can be a neural network with parameters $\phi$. It is also called the *critic* network. The approximator of the deterministic policy is denoted as $\mu_\theta(s)$, which can also be a neural network with parameters $\theta$. It is also called the *actor* network. The approximator of the deterministic policy $\mu_\theta(s)$ should be such that such that $\max_a Q_\phi(s,a) \approx Q_\phi(s,\mu_\theta(s))$. The training algorithm of DDPG has two parts: learning a Q function, and learning a policy.

The Bellman equation describing the optimal action-value function $Q^*(s,a)$ forms the core the DDPG algorithm.

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s'\sim P}\Big[r(s,a) + \gamma \max_{a'} Q^*\left(s',a'\right)\Big] \tag{12}$$

where $s' \sim P$ means the next state, $s'$, is sampled by the environment from a distribution $P(\cdot|s,a)$, which is the transition probability distribution of the MDP model of the environment. It is unknown to the agent. The Bellman equation is the starting point for learning an approximator to $Q_\phi(s,a)$ and determining the parameters $\phi$ of the actor network. Let $\mathcal{D}$ be a set of transitions $(s,a,r,s',d)$ (where $d$ is a binary variable, $d = 1$ indicates that state $s'$ is terminal). The mean-squared Bellman error (MSBE) function is an estimate of roughly how closely $Q_\phi$ comes to satisfying the Bellman equation

$$L(\phi,\mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)\right)\right)^2\right] \tag{13}$$

and the parameters $\phi$ are learned through MSBE minimization. Two important aspects of the training algorithm are mentioned below.

- *Replay Buffer*: DDPG is an off-policy algorithm and maintains old experiences in the replay buffer $\mathcal{D}$, even though they might have been obtained using an outdated policy. It is possible to use these for learning because the Bellman equation 12 for the optimal Q-function should be satisfied for all possible transitions $(s,a,r,s',d)$. So any transitions experienced can be used to fit a Q-function approximator via MSBE minimization. In order for the algorithm to have stable behavior, the replay buffer should be large enough to contain a wide range of experiences, but it should not be too large at the same time. Using only recent experience for training will result in overfitting. Using too much experience may slow down the learning. The length of replay buffer is an important hyperparameter and should be tuned for the problem at hand.
- *Target Networks*: The term $r + \gamma(1-d)Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)$ is known as the *target*. The minimization of the MSBE loss tries to make the Q-function close to the target. But the target also depends on the same parameters of training: $\phi$, $\theta$. This makes MSBE minimization unstable. The solution is to use a set of parameters which comes close to $\phi$, $\theta$, but with a time delay to determine the target. That is, a second set of networks with

parameters $\phi_{\text{targ}}$ and $\theta_{\text{targ}}$ called the target networks, which lags the first. In DDPG, the target networks are updated once per main networks update using a "soft update" rule.

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi, \qquad \theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta \tag{14}$$

where $\rho$ is a hyperparameter between 0 and 1 (usually close to 1).

Policy learning in DDPG involves learning parameters $\theta$ of the deterministic policy $\mu_\theta(s)$ which gives the action that maximizes $Q_\phi(s,a)$. Because the action space is continuous, the Q-function is differentiable with respect to action and can be optimized by performing gradient ascent with respect to policy parameters only. Note that the Q-function parameters are treated as constants here.

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathbb{E}} \left[Q_\phi\left(s, \mu_\theta(s)\right)\right]$$

Since DDPG trains a deterministic policy, the agent might not try a wide enough variety of actions to find useful strategies. So, noise $\epsilon \sim \mathcal{N}$ is added to the actions at training time to make DDPG policies explore better. Generally, the time-correlated Ornstein–Uhlenbeck (OU) noise or the mean-zero Gaussian noise is used for this. The scale of the noise can be reduced over the course of training to facilitate getting higher-quality training data. At test time, noise is not added to the actions as the goal is to see how well the policy exploits what it has learned to maximize rewards.

The complete training algorithm is given in Algorithm 1.

**Algorithm 1** Deep Deterministic Policy Gradient

Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
**repeat**
  Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute $a$ in the environment
  Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
  Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
  If $s'$ is terminal, reset environment state
  **if** it's time to update **then**
    **for** however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1-d)Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s,a) - y(r,s',d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

    **end for**
  **end if**
**until** convergence

# 4 Experiments

## 4.1 System Description

The following system model is used to test the allocation algorithms.

- The simulation are carried out assuming a 20MHz OFDMA-based LTE system. Thus, there are 100 RBs to be assigned in one TTI.
- A set of random number of UEs $N_{\text{UE}}$ are generated at the same time from a Uniform $U[1, 30]$ distribution. That is, $N_{\text{UE}} \sim U[1, 30]$.
- For each UE, packet size is randomly assigned a value between 30Byte to 10MB.
- In each TTI, CQI is randomly assigned to a UE by choosing one of the 14 CQI values possible.
- When all the packet reception is complete, another set of UEs is generated and the process repeats itself.

Performance of an algorithm is characterized by measuring average system throughput and average delay. Fairness of resource allocation among UEs, is characterized by the Jain's Fairness Index which is calculated over a window of 20 TTIs. For $K$ UEs, if $R_i$ is the average throughput of $i$th UE over 20 TTIs, then

$$J(R_1, ..., R_K) = \frac{(\sum_{k=1}^{K} R_i)^2}{K \sum_{k=1}^{K} R_i^2} \tag{15}$$

### 4.2 Reinforcement Learning Formulation

The reinforcement learning problem formulation is given below.

- State $s_t$: A state is represented by a $90 \times 1$ dimensional vector

  $$s_t = [R_t(1), T_t(1), P_t(1), ..., R_t(30), T_t(30), P_t(30)] \tag{16}$$

  where $R_t(i)$ is the instantaneous throughput of user $i$ calculated from the CQI value, $T_t(i)$ is the exponentially weighted average throughput of user $i$ given by Equation 2 with $t_c = 100$ and $P_t(i)$ is the remaining KB of data to be transmitted for user $i$. It is assumed that the BS can serve atmost 30 users at one time. If the number of users $N_{\text{UE}} < 30$, then $R_t(i) = 0, T_t(i) = 0, P_t(i) = 0$ for $i = N_{\text{UE}} + 1, ..., 30$.
- Reward $r_t$: Two different learning methodologies are proposed, each using a different reward mechanism.
  1. *Direct Learning*: In this methodology, the reward at time $t$ is calculated as follows

     $$r_t = 0.2D_t + \frac{0.2 J_t N_{\text{UE}}}{30} \tag{17}$$

     where $D_t$ is the total data transmitted in the $t$th TTI in KB, $J_t$ is the Jain's Fairness Index at time $t$ calculated as in Equation 15 and $N_{\text{UE}}$ is the number of UEs in the system. The agent would try to to maximize the total data transmitted till any time $t$ while giving some weightage to fairness. The weight of each term in the reward can be tuned. As the the number of UEs in the system increases, more weightage should be given to the fairness term, hence it is multiplied by $(N_{\text{UE}}/30)$.
  2. *Expert Learning*: In this methodology, a PF scheduler is used to determine the reward at time $t$.

     $$r_t = \mathbb{I}(D_t^{\text{RL}} \geq D_t^{\text{PF}}) + \frac{0.8 \sum_{k=1}^{N_{\text{UE}}} \mathbb{I}(T_t(k) \geq T_{\text{TH}})}{N_{\text{UE}}} \tag{18}$$

     where $D_t^{\text{RL}}$ is the total data transmitted by the RL agent in the $t$th TTI, $D_t^{\text{PF}}$ is the total data transmitted by a PF scheduler calculated by using the state $s_t$ at time $t$ and $T_{\text{TH}}$ is a threshold value for average throughputs of UEs. $\mathbb{I}(\cdot)$ is the Indicator function which equals 1 when its argument evaluates to True and 0 otherwise. The first term in Equation 18 would encourage the RL agent to achieve better instantaneous throughput than a PF scheduler in every TTI. As a result, the overall performance of the RL agent would become better than a PF scheduler. The second term in Equation 18 would result in the agent trying to maintain the average throughputs of UEs above the threshold $T_{\text{TH}}$. The threshold was decided to be 3Mbps after observing the average throughputs of UEs in PF setting.
- Action $a_t$: An action is represented by a $30 \times 1$ dimensional vector $[n_t(1), ..., n_t(30)]$ where $n_t(i)$ denotes the fraction of total RBs assigned to UE $i$. Note that $n_t(k) \geq 0$ and $\sum_{k=1}^{30} n_t(k) = 1$. This would be the output of the actor network.

### 4.3 Reinforcement Learning Simulation Parameters

- The Actor Network $\mu_\theta(s)$: It takes input a state $s_t$ and outputs an action $a_t$. It has the following architecture.

  $$s_t \rightarrow \text{BN} \rightarrow \text{FCN}(400) \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \text{FCN}(300) \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \text{FCN}(30) \rightarrow \tanh$$

  Here, BN denotes batch-normalization layer, FCN($n$) denotes fully connected linear layer with $n$ hidden units

- The Critic Network $Q_\phi(s, a)$: It takes input a state $s_t$ and $a_t$ and outputs the Q-value $Q(s_t, a_t)$ associated with it. It has the following architecture.

$$s_t \rightarrow \text{BN} \rightarrow \text{FCN}(400) \rightarrow \text{ReLU} \rightarrow \text{BN} \overset{+a_t}{\rightarrow} \text{FCN}(300) \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \text{FCN}(1)$$

- Exploration Noise $\epsilon \sim \mathcal{N}$: The time correlated OU noise $(\mu = 0, \sigma = 0.2)$ is used to perturb the action space and encourage exploration. The amplitude of noise is uniformly decreased from $1.0$ to $0.2$ with uniform stepping of $10^{-6}$. After adding noise, the output is then clipped to a range of $[-1, 1]$.
- To handle dynamic number of UEs, masking is used for the output of actor network. If there are $N_{\text{UE}}$ UEs in the system, then the mask is a $30 \times 1$ dimensional vector $[m(1), ..., m(30)]$ such that $m(i) = 1$ for $1 \leq i \leq N_{\text{UE}}$ and $m(i) = 0$ for $N_{\text{UE}} < i \leq 30$. The mask is multiplied by the output of the actor.
- Optimizers: Adam optmizer is used for the weight updates. Learning rate of the critic network is set to be $lr_Q^\phi = 0.0005$ and that of the actor network is $lr_\mu^\theta = 0.0001$. For soft update of the target networks, $\rho$ is set to be $\rho = 0.995$.

### 4.4 Results

In Figure 5, the performance of a PF scheduler (with $\alpha = 1$) is compared with other schedulers having different value of $\alpha$. $\alpha = 0$ case is essentially the greedy scheduler that allocates all RBs to the UE having maximum CQI.

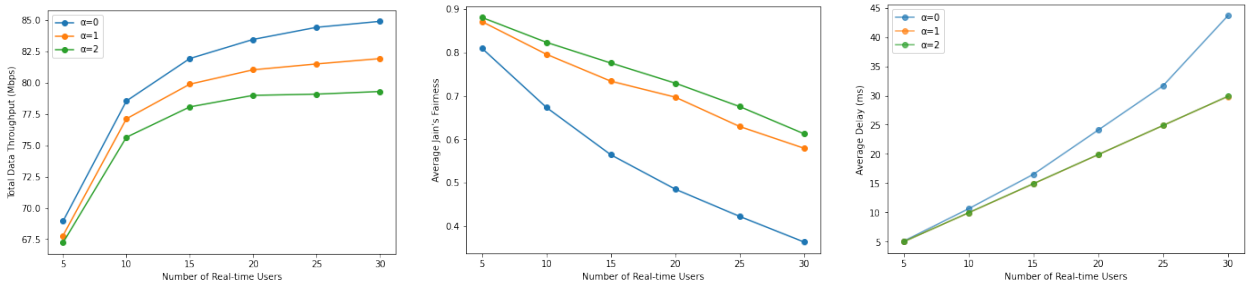


Figure 5: Effect of varying the weightage on average throughput (determined by $\alpha$)

It can be observed that the $\alpha = 0$ scheduler achieves the maximum throughput but at the cost of having lowest fairness and highest delay. The scheduler with $\alpha = 2$ has a relatively lower throughput but maintains a high fairness index. The PF schduler ($\alpha = 1$) is an intermediate between the other two schedulers achieving fairly good throughput and fairness. As expected, fairness decreases and delay increases with the increase in number of UEs.

The training progress of the RL-based scheduler trained using expert learning methodology is show in 6.

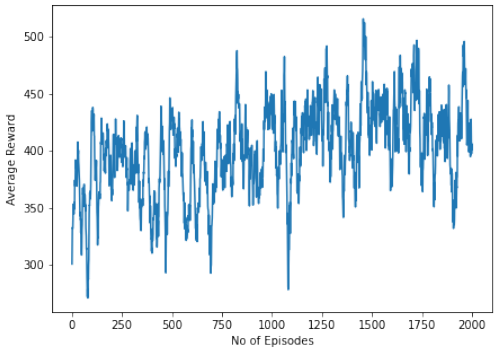


Figure 6: Training Progress

In this study, only performance comparison is performed between a RL-based scheduler trained through expert learning methodology (RL-EL) and a PF scheduler. The results are shown in Figure 7. Note that the RL-EL scheduler is not fully trained and the results can improve further after the training is complete.

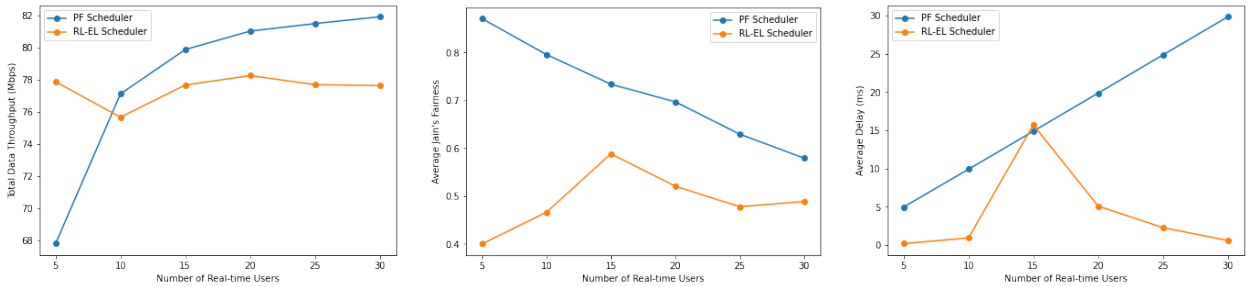


Figure 7: Comparison between a PF scheduler and a RL-based scheduler trained through expert learning (denoted as RL-EL Scheduler)

The RL-EL scheduler has lower system throughput than the PF scheduler, especially when number of UEs increases. But it manages to achieve a lower average delay thus making it suitable for real-time applications. A bit surprising is the fact that RL-EL scheduler has lower overall fairness index even though it has less average delay. This can be explained by the fact that the RL-EL scheduler allocates just enough resources to every UE to maintain the throughput threshold but after that, it performs allocation non-uniformly to maximize throughput.

### 4.5 Conclusion

This study compared the performance of the PF scheduler with that of a RL-based scheduler trained through expert learning (RL-EL scheduler). The PF scheduler has a slightly higher throughput than the RL-EL scheduler when the the number of UEs is high. On the other hand, the RL-EL scheduler maintains a good throughput without any significant variations with the number of UEs. The RL-EL scheduler achieves a very low average delay making it particularly suited for real-time applications. The results suggest that instead of replacing the conventional PF scheduler by the RL-EL scheduler, a more practical approach is a system that utilizes the strengths of both. A switch between PF and RL-EL scheduler can be designed that takes into account the number of UEs in the system and their QoS requirements. The RL-EL scheduler can learn online from the PF scheduler. The switch can be designed so that system performance is optimized while not influencing the training of RL-EL scheduler.

## References

[1] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: 1509.02971 [cs.LG].

[2] Z. Sun, C. Yin, and G. Yue. "Reduced-Complexity Proportional Fair Scheduling for OFDMA Systems". In: *2006 International Conference on Communications, Circuits and Systems*. Vol. 2. 2006, pp. 1221–1225.