Learn to Schedule (LEASCH): A Deep reinforcement learning approach for radio resource scheduling in the 5G MAC layer
&
Cellular Network Traffic Scheduling with Deep Reinforcement Learning

Presented by: Salman Mohebi

ITN WindMill Study Group Meeting

June 10, 2020

Learn to Schedule (LEASCH): A Deep reinforcement learning approach for radio resource scheduling in the 5G MAC layer.

F. AL-Tam∗, N. Correia∗, J. Rodriguez †
∗Centro de Electr´onica, Optoelectronica e Telecomunicac¸ ˜oes (CEOT) - Universidade do Algarve,
†Instituto de Telecomunicac¸ ˜oes, Universidade de Aveiro

Main contributions

- DRL model for Radio resource scheduling (RRS) in the 5G MAC layer:
  - **learn-rather-than-design (LRTD) approach**
- Pipeline for developing/training DRL agents
- Analysis the proposed model vs. baseline algorithms in different network settings

Reinforcement learning

- Markov decision process (MDP): $(\mathcal{S}, \mathcal{A}, \boldsymbol{P}, r, \gamma)$.
- Goal: maximize $G_t = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_0 = s_t\right]$
  - state-value, function $V(s) = \mathbb{E}\left[G_t | s_t = s\right]$
  - action-value function $Q(s,a) = \mathbb{E}\left[G_t | s_t = s, a_t = a\right]$

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q(s,a)$$

  - advantage function $A(s,a) = Q(s,a) - V(s)$.

Reinforcement learning

- Bellman expectation function

$$Q(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}(a) V(s')$$

- Bellman optimality equation

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}(a) \max_{a'} Q^*(s',a')$$

- Optimal policy

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a), \quad \forall s \in \mathcal{S}$$

- Problem: Transition probability is not known
  - Q- learning algorithm

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1})$$

$$\text{loss} = \left(Q(s_t,a_t;\boldsymbol{\theta}) - Q^{\text{target}}\right)^2$$

  - Approximate Q

$$Q^{\text{target}} = r(s_t,a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\boldsymbol{\theta})$$

    - Deep Q networks (DQN)
    - Mean-squared Bellman error (MSBE)

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_t + \frac{\alpha}{M}\left(Q(s,a;\boldsymbol{\theta}_t) - Q^{\text{target}}(\boldsymbol{\theta}_t)\right)\nabla_{\boldsymbol{\theta}_t} Q(s,a;\boldsymbol{\theta}_t)$$

Reinforcement learning

- Stabilize the results:
  - Two identical neural networks are: **target network** and **online network**

$$Q^{\text{target}} = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \hat{\boldsymbol{\theta}})$$

  - Experience replay memory $\hat{\boldsymbol{\theta}} = \beta\boldsymbol{\theta} + (1-\beta)\hat{\boldsymbol{\theta}}$

- Double DQN (DDQN)

$$Q^{\text{target}} = r_{t+1}(s, a) + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a, \boldsymbol{\theta}); \hat{\boldsymbol{\theta}})$$

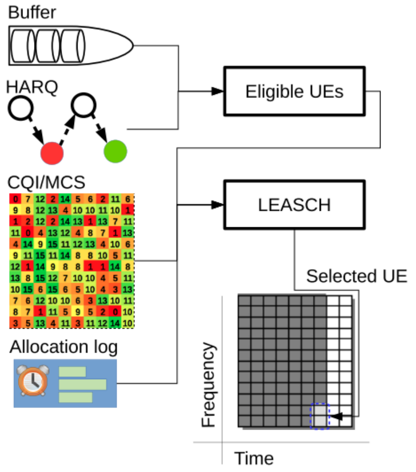
LEASCH's design-1

- Scheduler runs at the gNB at every slot
  - Grant the available resource block groups (RBGs) between UEs
  - Filling the resource grid
- Goal: jointly optimize the throughput and fairness
- State space:
  - Eligibility
  - Data Rate (d)
  - Fairness (f)

$$g_u = \begin{cases} 1, & \text{if } u \text{ is eligible} \\ 0, & \text{Otherwise} \end{cases}, \forall u \in \mathcal{U}$$

$$f_u = \begin{cases} \max(f_u - 1, 0), & \text{if } u \text{ is selected} \\ f_u + 1, & \text{if } u\text{'s buffer is not empty} \end{cases}, \forall u \in \mathcal{U}$$

- Compact State: $\hat{\boldsymbol{d}} = \boldsymbol{d} \circ \boldsymbol{g}$ $\quad \boldsymbol{s} = \left[\hat{\boldsymbol{d}} \quad \boldsymbol{f}\right]^\top$
- Normalize d^ and f to the range [0, 1]



LEASCH's design-2

- Action space: select one of the UEs in the system
- Reward:
  - Encourage the agent to transmit at the RBGs with the highest MCS (throughput)
  - Not to compromise the resource sharing between the users (fairness)

$$r(s,u;K) = \begin{cases} -K, & \text{if } u \text{ is none-eligible} \\ \hat{d}_u \times \frac{\min_u f_u}{\max_u f_u}, & \text{otherwise} \end{cases} \tag{21}$$

  - K is a threshold to penalize the scheduling an inactive UE

LEASCH training and deployment

Algorithm 1 - Training phase of LEASCH.

// *input:* $\ell_{episode}$, K, M, T ϵ, δ_ϵ, $\min_\epsilon$, $\boldsymbol{\theta}$, $\hat{\boldsymbol{\theta}}$, $\mathcal{R}$.
// *output: updated* $\{\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}, \mathcal{R}\}$.
initialize $\boldsymbol{s}$ randomly according to the ranges of $\hat{\boldsymbol{d}}$ and $\boldsymbol{f}$
for $i = 1 : \ell_{\text{episode}}$ **do**
  forward $\boldsymbol{s}$ to the on-line Q neural network and get the selected UE, u, via ϵ-greedy as:
  $u = \arg\max_{a \in \mathcal{A}} Q(\boldsymbol{s}, a; \boldsymbol{\theta})$
  anneal ϵ as: $\max\{\epsilon - \delta_\epsilon, \min_\epsilon\}$
  calculate the reward $r(\boldsymbol{s}, u; K)$ using (21)
  calculate new state s' using the equations $\boldsymbol{s} = \begin{bmatrix} \hat{\boldsymbol{d}} & \boldsymbol{f} \end{bmatrix}^\top$
  add the tuple $(\boldsymbol{s}, u, r, \boldsymbol{s}')$ to the experience replay $\mathcal{R}$
  sample M mini-batches from $\mathcal{R}$ and train the on-line Q neural network with $\boldsymbol{\theta}$ using (14) and (17)
  update the target critic Q neural network (with $\hat{\boldsymbol{\theta}}$) using $\boldsymbol{\theta}$ every T steps via smoothing (16).
  $\boldsymbol{s} \leftarrow \boldsymbol{s}'$
end for
return $\{\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}, \mathcal{R}\}$

Algorithm 2 - Deployment phase of LEASCH in 5G.

// *input: trained LEASCH.*
for each time slot **do**
  for each RBG **do**
    calculate the set of eligible UEs $\hat{\mathcal{U}}$
    if $\hat{\mathcal{U}} \neq \emptyset$ **then**
      calculate state $\boldsymbol{s}$
      forward $\boldsymbol{s}$ to LEASCH
      calculate the action u as:
      $u = \arg\max_{a \in \mathcal{A}} Q(s, a; \boldsymbol{\theta})$
      if $u \in \hat{\mathcal{U}}$ **then**
        schedule u for the current RBG
      end if
    end if
    collect statistics from the simulator
  end for
end for

Simulation parameters

Parameter	Value	Description
α	$1e^{-4}$	DNN learning rate
Optimizer	Adam	
Gradient threshold	1	
ϵ	0.99	ϵ-greedy parameter
$\min_\epsilon$	0.01	Min. allowed ϵ
δ_ϵ	$1e^{-4}$	ϵ decaying factor
$\lvert\mathcal{R}\rvert$	$1e^{6}$	Experience replay memory size
M	64	Mini-batch size
T	20	Smoothing frequency
β	$1e^{-3}$	Smoothing threshold
ℓ_{episode}	150 RBG	Episode length
No. of episodes	500	Training episodes

Parameter	Value
Radio access tech.	3GPP 5G NR
Test time	250 frames
Simulation runs	100 runs with different deployment scenarios
Numerology index μ	$\{0, 1, 2\}$
Bandwidth	{5MHz, 10MHz, 20MHz}
UEs	4
SCS	{15kHz, 30kHz, 60kHz}
No. of RBs	$\{25, 24, 24\}$ see [2]
Scheduling period	1 RGB
RBG size	2 RBs according to configuration 1 in [3]
Total tested RBGs	$250 \times 100 \times \{130, 240, 480\}$ RBGs
Channel development	Randomly changes each $\frac{1}{4}$ second
HARQ	True

Simulation setup

- Q neural networks are DNNs:
  - Two fully connected hidden layers of 128 neurons each
  - Relu activation functions
  - Input layer size: $2 \times |U|$
  - Output layer size: $|U|$
- Matlab 2019b:
  - Linux with i7 2.6GHz,
  - 32GB RAM,
  - GPU Nvidia RTX 2080Ti with 11 GB
- Key performance indicators:
  - Throughput: achievable data rate in the cell
  - Goodput: Delivered data rate at receiver
  - Fairness: Jain's fairness index

Results

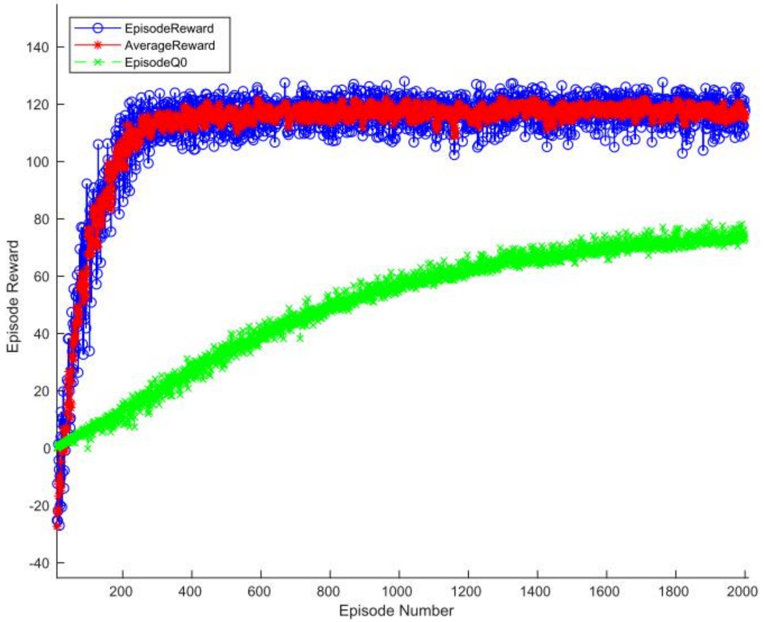
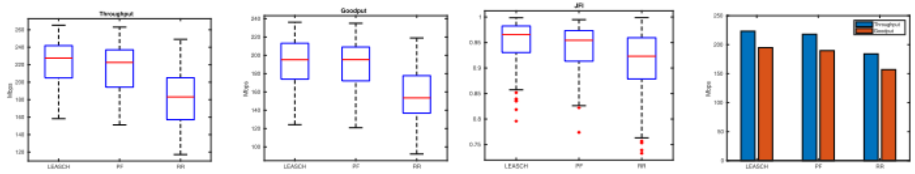
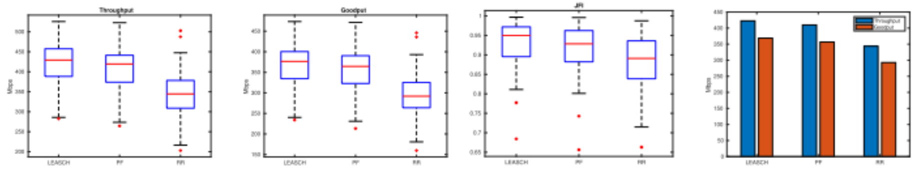


Figure 2: LEASCH learning curve for 2000 episodes.



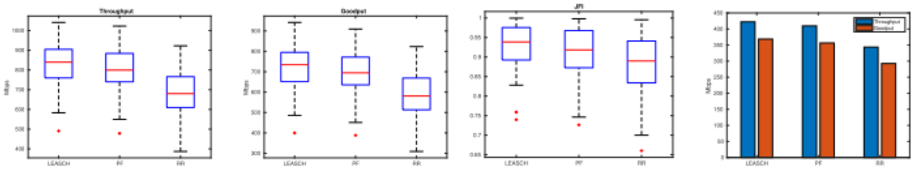
(a) Throughput (b) Goodput (c) Fairness (d) Mean throughput-goodput

Figure 3: KPIs for 250 frames of 15kHz SCS under 5MHz BW for 100 runs.



(a) Throughput (b) Goodput (c) Fairness (d) Mean throughput-goodput

Figure 4: KPIs for 250 frames of 30kHz SCS under 10MHz BW for 100 runs.



(a) Throughput (b) Goodput (c) Fairness (d) Mean throughput-goodput

Figure 5: KPIs for 250 frames of 60kHz SCS under 20MHz BW for 100 runs.

Results

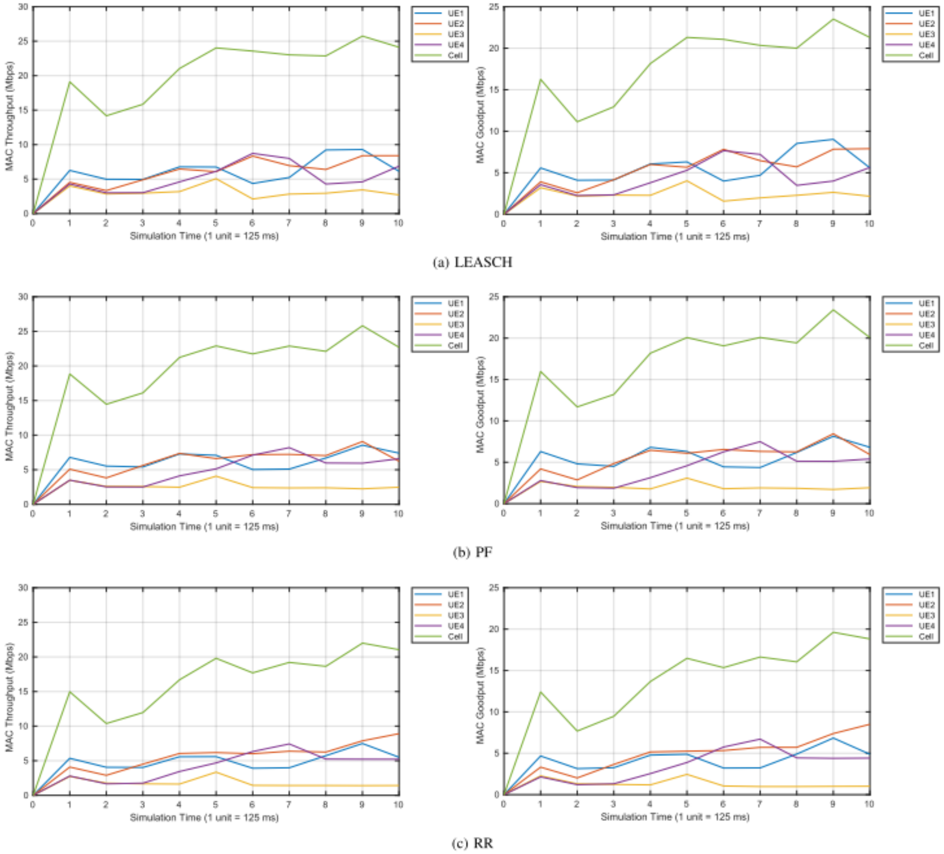


Figure 6: A random testing run for the 10MHz BW and 30kHz SCS setting. Left column: throughput; right column: goodput.

Cellular Network Traffic Scheduling with Deep Reinforcement Learning

Sandeep Chinchali [1], Pan Hu [2], Tianshu Chu [3], Manu Sharma [3], Manu Bansal [3], Rakesh Misra [3]
Marco Pavone [4] and Sachin Katti [1,2]

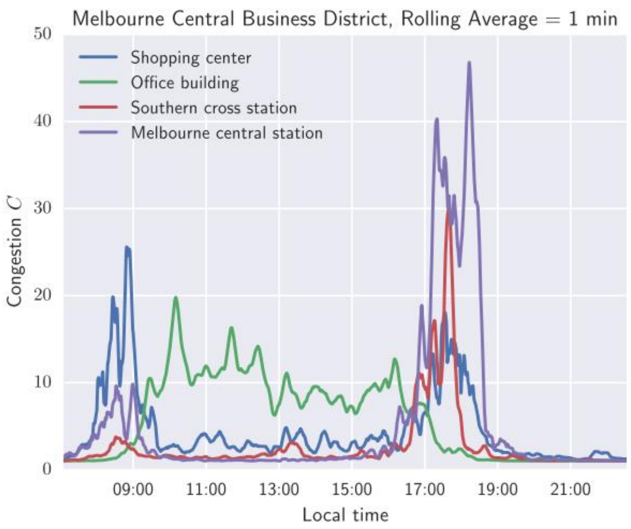
[1] Department of Computer Science, Stanford University
[2] Department of Electrical Engineering, Stanford University
[3] Uhana, Inc.
[4] Department of Aeronautics and Astronautics, Stanford University
{csandeep, panhu, pavone, skatti}@stanford.edu, {tchu, manusharma, manub, rakesh}@uhana.io

Introduction

- Focus on mobile networks,
  - Increasing request for new class of applications, driven by IoT
- High Volume Flexible Time (HVFT) applications
  - Software and data updates to mobile IoT devices:
    - Updating maps for self-driving cars or delivery drones
  - Large transfer of IoT sensor data to the cloud:
    - Energy usage measurements from a smart grid,
  - Prefetched ultra-high quality and bitrate video

Challenges

- Time-variant and non-Markovian network dynamics
- Networks exhibit non-stationary dynamics
  - ranging from short-term, minute-scale variation to daily commute patterns
- Data set: 4 weeks of data from 10 diverse cells in Melbourne, Australia



- Incorporate **past measurements** and **historical** commute patterns into our state representation to recast the problem as MDP to leverage RL methods:
  - Temporal feature mapping function

Data-driven Network Model-1

- Discrete-time, continuous state and action space MDP,
- **State space:** $$S_t = [C_t, N_t, E_t]$$
  - Cell Congestion (C): The effective number of users in the cell
  - Average Cell Efficiency (E): average cell quality
    - Total cell bandwidth, type of cellular technology deployed, distance of users from the cell tower, and …
  - Number of connections (N)

$$s_t = [S_t, \phi(S_0, \ldots, S_t, t, T)]$$

- ϕ is a **temporal feature mapping** function
  - feed-forward neural networks,
  - Long Short Term Memory (LSTM) networks,
  - Autoregressive Integrated Moving Average (ARIMA) techniques

Data-driven Network Model-2

- **Action:** $$a_t \in [0,1]$$
  - rate at which HVFT traffic can be served on top of conventional traffic

- **Reward**:

$$R(s_t, a_t) = \alpha V_t^{\mathrm{IoT}} - \beta V_t^{\mathrm{loss}} - \kappa V_t^{\mathrm{below\ limit}}$$

  - IoT traffic served
  - Loss to conventional application after adding IoT traffic
  - Bytes served below minimum throughput L

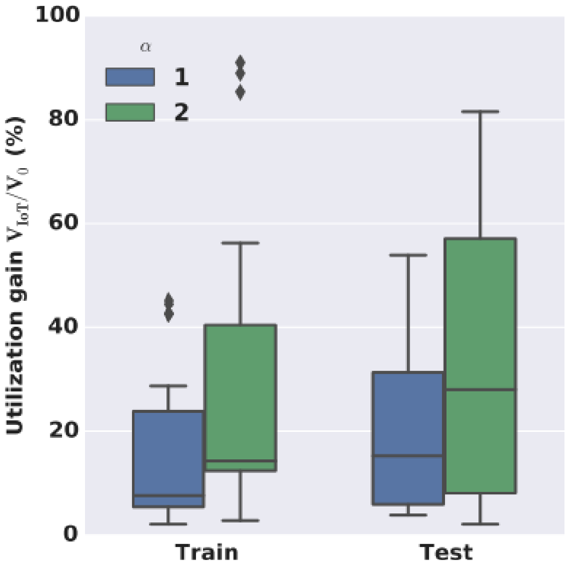
Implementation setup

- Deep Deterministic Policy Gradient (DDPG) algorithm [1]
  - Two hidden layers of sizes 400 and 300
  - Learning rate for actor and critic networks: 0.0001 and 0.001,
  - The discount factor is 0.99
  - Mini-batch size is 32
  - LSTM: two-layer architecture with 50 units per layer

[1] https://bitbucket.org/sandeep_chinchali/aaai18_deeprlcell

Utilization gain

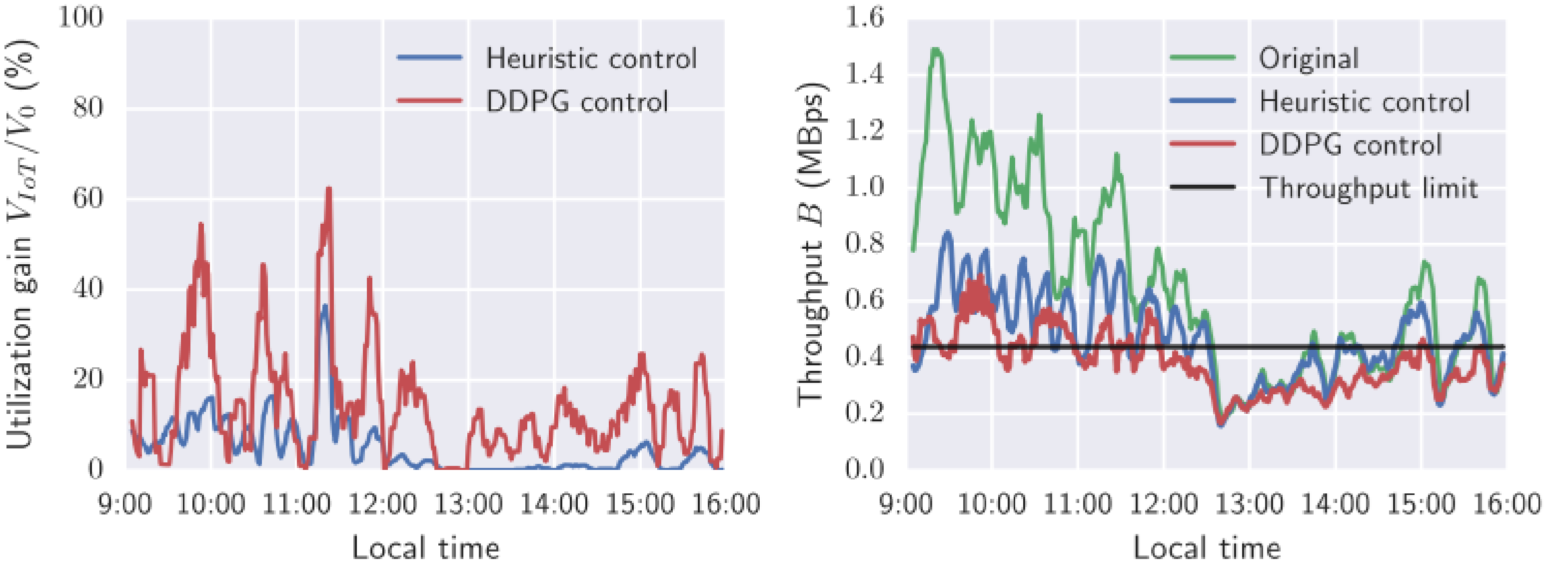
- Melbourne cell-day pairs (19 train, 8 test days)
  - IoT traffic favoring policy (α = 2, β = 1, κ = 1)
  - Conservative policy (α = 1, β = 1, κ = 1)



- Utilization gain: 14.7%

- The 10 MHz of radio spectrum, costing roughly $4.5B,
  - Utilization gain of 14.7% means the operator is saved about $661 million

RL vs. Benchmark Controllers



(a) IoT traffic (b) Throughput

Thank you very much for your attention.

Questions & Discussion…?