DeepNap: Data-Driven Base Station Sleeping Operations through Deep Reinforcement Learning

Jingchu Liu, Bhaskar Krishnamachari, *Senior Member, IEEE,* Sheng Zhou, *Member, IEEE,* and Zhisheng Niu, *Fellow, IEEE*

***Abstract*—Base station sleeping is an effective way to reduce the energy consumption of mobile networks. Previous efforts to design sleeping control algorithms mainly rely on stochastic traffic models and analytical derivation. However the tractability of models often conflicts with the complexity of real-world traffic, making it difficult to apply in reality. In this paper, we propose a data-driven algorithm for dynamic sleeping control called DeepNap. This algorithm uses a Deep Q-network (DQN) to learn effective sleeping policies from high-dimensional raw observations or un-quantized systems state vectors. We propose to enhance the original DQN algorithm with action-wise experience replay and adaptive reward scaling to deal with the challenges in non-stationary traffic. We also provide a model-assisted variant of DeepNap through the Dyna framework for inferring and simulating system dynamics. Periodical traffic modeling makes it possible to capture the non-stationarity in real-world traffic and the incorporation with DQN allows for feature learning and generalization from model outputs. Experiments show that both the end-to-end and the model-assisted version of DeepNap outperform table-based Q-learning algorithm and the non-stationarity enhancements improve the stability of vanilla DQN.**

***Index Terms*—Base station sleeping, deep reinforcement learning, deep Q-network, non-stationary traffic.**

I. Introduction

THE explosive growth of mobile data usage has triggered an accelerating expansion of the mobile network infrastructure over the past few years. As a consequence, the network energy consumption has surged dramatically, raising both economic and environmental concerns. This necessitates the adoption of more energy-efficient network architectures and operations. Previous studies reveal that base stations (BSs) are responsible for $60\% - 80\%$ of the total network energy consumption [1] and BS traffic load is less than $\frac{1}{10}$ of the peak value for 30% time of weekdays [2]. This motivates dynamic BS sleeping operations, in which BSs are automatically turned into a low-power sleep mode when the traffic volume is low.

This work is sponsored in part by the Nature Science Foundation of China (No. 91638204, No. 61571265, No. 61621091), and Hitachi Ltd. (*Corresponding author: Sheng Zhou.)*

J. Liu was with the Tsinghua National Laboratory for Information Science and Technology, Department of Electronic Engineering, Tsinghua University, Beijing 100084, China. He is now with Horizon Robotics (email: liujingchu@gmail.com).

B. Krishnamachari is with the Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089 USA (email: bkrishna@usc.edu).

S. Zhou and Z. Niu are with the Tsinghua National Laboratory for Information Science and Technology, Department of Electronic Engineering, Tsinghua University, Beijing 100084, China. (email: {sheng.zhou, niuzhs}@tsinghua.edu.cn).

BS sleeping can operate on various time-scales: *slow* sleep cycles are usually in minutes to hours while *fast* ones in seconds to minutes [1], [2], [3] . Fast sleeping operations can better leverage fine-grained traffic variations but need to carefully balance the tradeoff between energy savings and QoS degradations such as delay and outage. Queueing theory is a widely used analytical framework for studying such tradeoffs [4], [5], [6], [7]. To guarantee the tractability of analysis, previous work relies heavily on idealistic assumptions such as Poisson traffic models and threshold-based scheduling. However, real-world traffic are often more complex than Poisson models due to phenomenons such as self-similarity [8] and non-stationarity [9], and more delicate and model-free scheduling method may bring better sleeping gain. In fact, BS sleeping control problem manifests a sequential decision-making process: the sleeping controller observes past and current states of the system, e.g. traffic load and queue length, and decides the sleeping state of the BS. The control decisions are then carried out on the system and influences the uses network usage behavior and traffic pattern, driving transitions of system states, and invoke further sleeping control. Reinforcement learning (RL) is a widely-used framework to tackle complex sequential decision-making problems, and thus it is suitable for BS sleeping control, especially under real-world traffic that simple model cannot precisely describe.

Recently, deep learning has achieved significant breakthroughs in computer vision, speech recognition, and natural language processing [10]. It is also demonstrated in [11], [12], [13] that deep learning can help RL algorithms learn sequential control tasks in an end-to-end fashion. Deep learning and other data-driven methods have also been proposed for various challenging communications problems [14], [15], [16], [17], [18]. These advancements motivate us to use deep reinforcement learning to learn effective BS sleeping schedules.

In this paper, we propose DeepNap, a dynamic BS sleeping control algorithm based on deep reinforcement learning. DeepNap leverages a Deep Q-network (DQN) [11], [12] to learn energy efficient sleeping control policy from high-dimensional raw observations or system belief vectors. To address the issues of applying the original DQN with the non-stationarity of network traffic, we propose the following enhancements for DQN training: 1) Under non-stationary traffic, the replay memory may be dominated with experiences from the current traffic phase and produce biased samples. This can cause coupling among network outputs and trap DQN in sub-optimal policies. To avoid this, we employ **action-wise experience replay** to balance the amount of samples from different

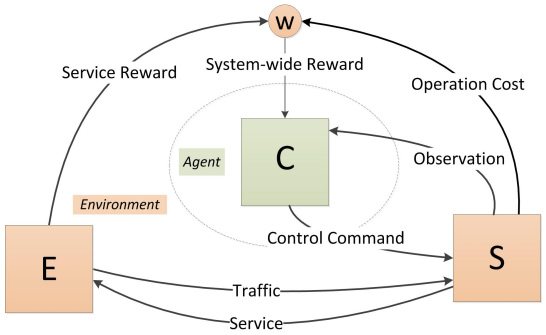


Fig. 1. RL formulation of BS sleeping control. The controller (C) servers as the agent, while the traffic emulator (E), traffic server (S), and reward combiner (W) together serve as the environment.

traffic phases. 2) The rewards received from mobile networks often have large and unknown dynamic ranges. Naive reward clipping may lose much information encoded in the magnitude of rewards. We apply **adaptive reward scaling** to adaptively rescale the received rewards to match with the response range of network outputs.

To facilitate the incorporation of domain knowledge, DeepNap can also work in a model-assisted manner. A traffic modeling module iteratively fits a Interrupted Poisson Process (IPP) [19] using the Baum-Welch algorithm [20] and then predicts the next *traffic belief state*. The DQN can subsequently uses the belief state vector to make decisions or learn better sleeping policies. Since the traffic model is learned in an online fashion, it can capture more complex traffic dynamics (e.g. non-stationarity) than a model with static parameters; Compared with table-based Q-learning and analytical derivation, DQN also makes it possible to use unquantized belief state and avoids the information loss due to quantization; We also incorporate planning architecture into the learning process through the Dyna framwork [21], in which pseudo state transitions generated from the learned traffic model are used alongside real transitions during DQN training.

We evaluate DeepNap using an data-driven network emulation framework called DragonEye [22]. Results show that DeepNap outperforms the baseline table-based Q-learning algorithm with a large margin. We also observe improved stability compared with DQNs without action-wise experience replay and adaptive reward scaling. The usage of a learned traffic model and pseudo experience also brings slight improvements over end-to-end DQN learning.

The rest of the paper is organized as follows. Section II provides the system model and some preliminaries of the deep Q-network. The proposed deep learning-based BS sleeping algorithm is introduced in Section III. Extensive experiment results are presented in Section IV. Related work and conclusions are summarized in Section V and VI respectively.

II. System Model and Preliminaries

A. BS Sleeping as a Reinforcement Learning Problem

To formulate short-term BS sleeping control as an RL task, we abstract the system as a collection of four components as illustrated in Fig. 1: a traffic emulator (E) which encapsulates the behavior of traffic sources, a traffic server (S) which serves as the data plane of the BS, a sleeping controller (C) which is the control plane of the BS and directs the operation of the data plane, and a reward combiner (W). The correspondence between an RL agent and the controller C is quite obvious. Meanwhile, the RL environment is the collection of E and S as well as the reward combiner W. In each round of interaction, E first generates traffic for S, which then prepares observations for C and receive control commands. Afterwards, S wakes up and serves the traffic or sleeps based on control commands. Depending on the service, E emits a scalar reward to measure the service quality and S also emits a scalar to quantify the operation cost. These rewards are summed up by W and passed back to C. Note this formulation is intended for the short-term BS sleeping control.

B. Deep Q-Network (DQN)

Q-learning [23] is a model-free RL algorithm. It leverages the Bellman iteration to estimate the action-value function

$$Q^{(i+1)}(s,a) = r + \gamma \max_{s'} Q^{(i)}(s',a'))$$

where s is the state, a is the action taken, r is the reward, s' is the next state, and $\max$ is the maximum operator. Once the optimal action-value function $Q^*(s,a)$ is estimated, the optimal policy can be explicitly expressed as a greedy procedure over the optimal action-value function $\pi^*(s) = \arg\max_a Q^*(s,a)$, where $\arg\max$ gives the argument that results in the maximum value in the given function.

Q-learning requires *every* state-action pairs be visited sufficiently often and the values separately stored. However, this point-wise estimation and representation of the action-value function is unpractical because the state space of most interesting problems are high-dimensional. As a solution, one can learn a parametrized approximate of the action-value function $Q(s,a;\boldsymbol{\theta}) \approx Q^*(s,a)$ instead. A deep Q-network (DQN) [11], [12] uses a deep neural network as the action-value approximator. DQN can be trained by minimizing the mean-squared temporal difference

$$L(\boldsymbol{\theta}^{(i)}) = \mathbb{E}\left[(y^{(i)} - Q(s,a;\boldsymbol{\theta}^{(i)}))^2\right] \tag{1}$$

with gradient-based optimization methods, where

$$y^{(i)} = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a';\boldsymbol{\theta}^{(i-1)})|s,a\right] \tag{2}$$

is the target.

Three techniques are proposed to stabilize the training procedure. 1) a replay memory stores certain amount of past transitions. A random batch of experiences is sampled from the replay memory in each training step as a bootstrapped estimation of the true distribution. 2) a separate network with stale parameters is used to generate the *target* Q values, in order to avoid unwanted oscillations and divergence. 3) the reward is clipped to $[-1,+1]$ range.

C. Prediction and Learning of IPP Models

An IPP model [19] is a Poisson-emission hidden Markov model with two hidden states $\{s_1, s_0\}$, one of which (say s_0) has zero emission rate. IPP models can be used to model bursty traffic and is commonly used to model network traffic in existing studies on the BS sleeping problem [7]. The formal definition of an IPP model is as follows: assume transition probabilities P_{10} and P_{01}, and emission rate λ, the belief of next state can be derived with one-step forward prediction following the Markov property

$$\begin{aligned}\Pr\{s^{(t)} = s_1\} =& (1 - P_{10}) \Pr\{s^{(t-1)} = s_1\} + \\ & P_{01} \Pr\{s^{(t-1)} = s_0\}.\end{aligned} \tag{3}$$

The parameters of an IPP model can be learned from a sequence of traffic $O = \{o_1, \cdots, o_T\}$ using the Baum-Welch algorithm [20]. This algorithm first estimates sufficient statistics

$$\xi_{ij}(t) = \Pr\{s^{(t)} = i, s^{(t+1)} = j \mid O\}, \tag{4}$$

$$\gamma_i(t) = \Pr\{s^{(t)} = i \mid O\}, \tag{5}$$

$$\zeta(t) = o_t \cdot \Pr\{s^{(t)} = s_1 \mid O\}, \tag{6}$$

based on old model parameters using the forward-backward inference algorithm. These sufficient statistics are then used to calculate the new maximum-likelihood estimate of model parameters:

$$\Pr\{s^{(1)} = i\} = \gamma_i(1), \tag{7}$$

$$P_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}, \tag{8}$$

$$\lambda = \frac{\sum_{t=1}^{T} \zeta(t)}{\sum_{t=1}^{T} \gamma_1(t)}. \tag{9}$$

III. Proposed Algorithm

In this section, we elaborate the design of DeepNap. We first explain the problems with naive experience replay and reward clipping to motivate the action-wise experience replay and adaptive reward scaling enhancements for DQN. Then we describe the environment model and how it can be incorporated with DQN through the Dyna framework. After that, we present the overall DeepNap algorithm.

If the environment is non-stationary, naive experience replay can become problematic. Shown in Fig. 2 is an example taken from our experiment with the original DQN algorithm. The top curve shows the average number of requests per time step smoothed over one-minute time windows. The traffic pattern is clearly non-stationary. Driven by the traffic variation, the experience distribution in the replay memory (middle curve) oscillates between waking- and sleeping-dominating regimes. Since the loss in (1) is related with one action (thus one network output) per sample, only a single network output can be updated in these dominating regimes, and the other "free-running" output may drift away from its expected value. For example, observe in the beginning of the experiment, the memory is dominated by waking experiences, thus the Q value for the "free-running" sleeping action drifts to around

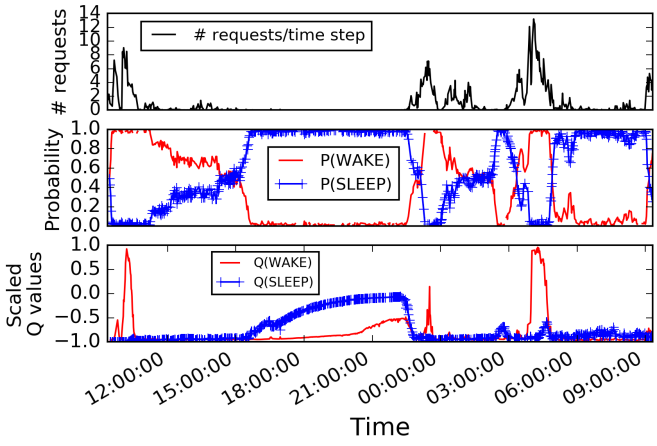


Fig. 2. Experimental results for the original DQN algorithm. The figures show the number of requests per time step smoothed over a one-minute time window (top), the percentage of experiences with waking and sleeping action in the replay memory (middle), and the average Q values for waking and sleeping actions over one-minute time window (bottom). The data used is taken from Sep 25 to Sep 26, 2014. For more details about the dataset, please see Section IV.

-1. It is only until $16:00$ that the dominance of waking experience got broken by random exploration and the Q value for sleeping action starts to amble towards the expected 0 value. The balance is once more broken by the traffic peak at around $23:00$, with sleeping Q values again pushed to around -1.

A. Action-wise Experience Replay

We propose to use an action-wise memory to address the above problem. To balance the sample action distribution, we store experiences with different actions into different memory buffers, and sample experiences of different actions with equal probability during training phase. This technique is motivated by the observation that non-stationary environments tend to drive the experiences in memory to switch between the dominating regimes of a single action. By storing and sampling experiences action-wisely, this action imbalance can be alleviated. In essence, action-wise experience replay tries to separately store experiences from different phases of a non-stationary environment, so that the the agent is less prone to overfitting to one particular phase.

B. Adaptive Reward Scaling

In mobile networks, the traffic volume of peak and off-peak periods may differ by one or two degrees-of-magnitude, and traffic-related reward components may consequently have extremely large dynamic range. Meanwhile, the reward magnitude is clipped to $[-1, +1]$ in the original DQN algorithm. In this way, the information encoded in value ranges outside of this clipping threshold is totally thrown away.

Reward rescaling is a straightforward solution for this issue. Appropriate scaling can bound the re-scaled action values to within the saturated range of network outputs. Even if the action-value function for some state-action pairs have extremely large magnitudes and cause saturation in network

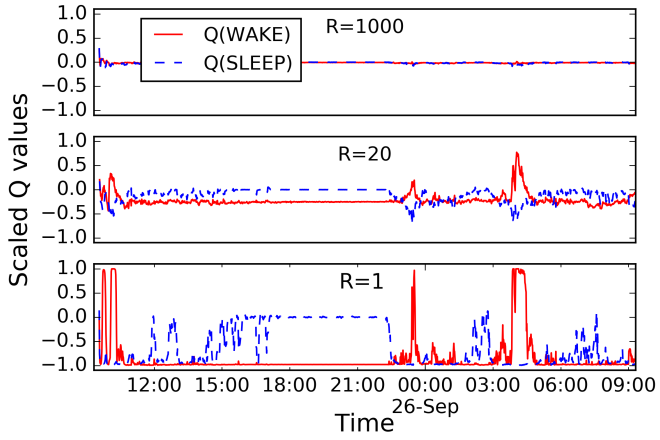


Fig. 3. Q values of waking and sleeping actions smoothed over one-minuted time window with fixed reward scaling factors of 1, 20, and 1000.

outputs, the policy will not be affected if the maximum action value is still maximum. However, choosing an appropriate scaling factor in unknown environments is no trivial task. On one hand, small scaling factors cause saturation on multiple outputs and the value of corresponding actions are confused (bottom chart in Fig. 3). It can also cause vanishing gradients and in turn hamper the training process. On the other hand, large scaling factors may create a fake local minima by squeezing action values to zero (top chart in Fig. 3).

To automate the search for an appropriate scaling factor, we propose the adaptive reward scaling procedure. Specifically, we revise the loss function as

$$L'(\boldsymbol{\theta}^{(i)}) = \mathbb{E}\left[\left(y'^{(i)} - Q(s,a;\boldsymbol{\theta}^{(i)})\right)^2 + U(Q(s,a;\boldsymbol{\theta}^{(i)}))\right], \tag{10}$$

where $y'^{(i)} = \frac{r}{R^{(i)}} + \gamma \max_{a'} Q(s',a';\boldsymbol{\theta}^{(i)-})$ is the rescaled target at iteration i by a scaling factor $R^{(i)} > 0$ and

$$U(Q) = \frac{\kappa}{(Q-1-\delta)^2} + \frac{\kappa}{(Q+1+\delta)^2} \tag{11}$$

is a U-shape saturation penalty, κ is a small constant to keep the penalty small in non-saturated regions, and δ is a small positive offset to bound the penalty value in saturated regions. The adaption process starts with a small scaling factor, e.g. $R^{(0)} = 1$. After every S updates for $\boldsymbol{\theta}$, we update the value of $R^{(i)}$ using gradient-based method towards reducing the loss in (10).During this process, the U-shaped penalty serves the purpose of pushing the output of saturating samples slightly in-wards (Fig. 3(c)), guaranteeing a non-diminishing gradients.

Intuitively, the proposed method guides a scheduled search for an appropriate $R^{(i)}$ starting from a small value. In each search cycle, if the true action-value function is larger than the scaled output, the loss will swell due to output saturation. In such a case, the loss can be reduced by increasing the scaling factor $R^{(i)}$. This increase of $R^{(i)}$ will gradually slow down when the output range catches up with the true action-value.

C. Environment Models and Dyna Integration

The core of our environment model is a learned IPP. Since the traffic is non-stationary, we make use of an online Baum-Welch algorithm to adapt model parameters to changing traffic patterns. Specifically, we keep the most recent W traffic observation during training and fit new model parameters with these observations every M time steps. We only apply three EM iterations each time to avoid overfitting to the short traffic window. In addition to the traffic belief state, the system state also contains the queue length in current and the latest time step as well as the latest sleeping action. The transitions of these additional state components is deterministic: new traffic arrivals are all enqueued if last action is to sleep while the queue is served and cleared if the last action is to wake up.

We apply the following rewarding scheme to favor immediate service and sleeping operations in low-traffic periods. In each time step, E emits $+R_s$ reward for each served request, $-R_q$ for each queued or retransmitted request, and $-R_f$ for each failed requests. At the same time, S emits $-C_o$ if the action is wake up and a $-C_w$ switching cost if the current action differs from the last one. The overall reward is derived by a weighted average of these rewards:

$$\begin{aligned} r^i =& w(R_s - R_q - R_f) \\ & - (1-w)(C_o + C_w \cdot \mathbf{1}\{a^{i-1} != a^i\}). \end{aligned} \tag{12}$$

Dyna [21] is an effective framework to integrate model-free learning and model-based planning in RL systems. We employ the DynaQ variant [24] to intergrate the environment model with DQN. Concretely, in each time step, we generate multiple pseudo experiences with the environment model. These simulated experiences are stored into the replay memory along with real experiences and used indiscriminately for training. Note all the above reward components can be calculated determinsitically excepte the traffic failure penalty $-R_f$, which requires finer traffic information than what the IPP model can provide. Thus we omit this component when generating pseudo experiences.

D. Putting Them All Together

The overall procedure of DeepNap is summarized in Algorithm 1. We periodically fit an IPP-base environment model with lateset traffic observations and use the filtered systems state as input for DQN. We use a feed-forward neural network as the DQN. Network weights are initialized with the Glorot initialization method [25]. We train the network periodically with mini-batches of samples taken from the action-wise replay memory and synchronize the parameters of the target Q network with a longer cycle. Besides, we also periodically update the scaling factor $R^{(i)}$. In the original DQN training procedure, experiences in the latest sliding window of length P are stacked as a extented observation vector

$$\boldsymbol{\phi}(s_t) = (a_{t-P}, s_{t-P+1}, a_{t-P+1}, \cdots, s_t).$$

DeepNap supports such a function for raw observations but keeps $P = 1$ when observations are filtered as system belief states.

IV. Experiments

In this section, we present the experiment results. The code and data used in experiments are publicly available at https://github.com/zaxliu/deepnap.

Algorithm 1: Pseudo-code of the DeepNap learning algorithm.

Initialize the network emulator as the environment.;
Initialize a traffic buffer $\mathcal{B}$ and a traffic model. ;
Initialize a value network and its target network with random weights. ;
Initialize an action-wise replay memory $\mathcal{M}_a$. ;
Repeat
  Push observation o into traffic buffer $\mathcal{B}$;
  Infer system state s ;
  Select a following ϵ-greedy policy;
  Feed a into the network emulator and observe o' and r;
  Every M **steps do** Fit new traffic model with current $\mathcal{B}$;
  $\phi' \leftarrow$`PHI(`s'`)` ;
  Simulate N pseudo experience $(\hat{\phi}, \hat{a}, \hat{r}, \hat{\phi}')$;
  Update memory $\mathcal{M}_a$ with both real and simulated experience (ϕ, a, r, ϕ');
  For 1 *to* $N+1$ **do**
    Every U **steps do**
      Sample a mini-batch uniform randomly from all action-wise memories $\{\mathcal{M}_a\}$;
      Update network parameters θ;
      Every F **steps do** Sync. target network. ;
      Every S **steps do** Update scaling factor R ;
    End
  End
  $s \leftarrow s'$, $\phi \leftarrow \phi'$;
Until `finished();`

TABLE I
FORMAT OF THE DATASET USED IN EXPERIMENTS.

Field	Content	Example
uid	user ID	41117355
bldn	building name	Dining Hall 1
start	session start time (Unix)	1409500812697
dur	session duration (ms)	295551
dmns	domain list	[a.com, b.net]
prvdr	domain providers	[Tencent, Apple]
types	domain categories	[Portal, Shopping]
bytes	bytes/domain	[8500, 341]
reqs	HTTP requests/domain	[5, 1]

A. Network Emulator and Dataset

We use the trace-driven network emulation framework DragonEye [22] as the test environment for the DeepNap agent. The trace we use is captured from a campus WLAN from September 2014 to January 2015. It contains the session-level HTTP traffic summaries of around $20,000$ users[1].Each record summarizes the per-domain HTTP activity of each user during each session with the following fields: session ID, user ID, building name[2], start time (UNIX time), duration (milliseconds), a list of the requested domain names as well as the corresponding provider, domain type, the total number of HTTP requests and bytes requested at each domain. Explanations and examples for record fields are summarized in Table I.

TABLE II
PARAMETERS AND DEFAULT VALUES

Params	Value	Description
P	15	length of $\phi(\cdot)$
$\lvert\mathcal{M}_a\rvert$	200	size of each action-wise memory
W	50	traffic window size
R	1.0	initial reward scaling value
N	5	# simulated experience per step
M	2	model fitting period
U	4	SGD update period
F	16	target network synchronization period
S	32	reward scaling update period
R_s	1	service reward
R_q	1	queueing cost
R_f	10	timeout penalty
C_o	5	BS active mode cost
C_w	0.5	switching cost
w	0.5	reward combining weight
T	7	# days of the testing period
Δ_T	2s	duration of each time step

DragonEye requires us to make the following assumptions to mitigate the imperfection in the dataset:

- **Virtual large cell**: the network trace we use only contains building-level location information. To cope with the uncertainty of user location at sub-building level, we assume all the users in a physical building are served by a large-coverage BS instead of multiple small-coverage APs. This assumption is inspired by the fact that the actual geographical layout of the buildings involved are rather far apart and the size of a single building matches the coverage size of a typical cell in cellular communication systems.
- **Byte and epoch allocation**: we assume a uniform[3] byte and epoch allocation model to translate coarse session-level summary to fine-grained request description. According to this model, each byte in a session is assigned to each request of that session with equal probability, and each request is assigned to an epoch within that session with equal probability too. This essentially results in a multinomial byte-per-request and request-per-epoch probability distribution.
- **User and network state machines:** we model the reaction of users and the network using state machines. At the user side, the transmission of each request follows the state machine. All requests are initialized as "pending" and remain there before being transmitted. If a request should be sent out in a particular epoch, its state is modified to "waiting" and starts transition according to the service received: if this request is successfully served,

[1]Here "session" is defined as the period in which a user generates HTTP requests and pauses for less than 5 minutes each time. If the user pauses longer, the subsequent requests will be summarized into the following session.

[2]More detailed knowledge of user location is hidden to preserve user privacy.

[3]Note that any other distribution could also be used.

its state is then changed to "served" and transition is terminated; if otherwise it is queued, its state remains at "waiting". All "pending" and "waiting" requests are also tagged as "failed" after the last epoch of a session. The network-side state machine is relatively simple. In each epoch, S first puts all incoming requests in a request queue. If the control command received is "serve", then all queued requests will be served and queue cleared; otherwise if the command is "queue", the queued requests remain in the queue for the next epoch.

B. Parameters

We use a 0.9 discount factor and 0.02 exploration probability for the DeepNap agent. The DQN has a input dimensions of 3 (system states) or $(3+2)\times P$ (raw observations) and output dimensions of 2. The network has two hidden layer, each with 500 units with ReLU activation, and a output layer uses with tanh non-linearity. The weight and offset of saturation penalties are respectively $\kappa = 10^{-5}$ and $\delta = 10^{-2}$. Network parameters are optimized using mini-batch Nestrov-momentum updates with 0.9 momentum, 10^{-2} step size, and 100 batch size. The default value for other hyper-parameters used in our experiments are listed in Table. II.

C. Learned Sleeping Patterns

Fig. 4 shows the basic characteristics of the sleeping patterns learned by a DeepNap agent. The top chart (a) shows the traffic variations during this period. The traffic volume peaks and valleys with regular temporal pattern, but local random variations is also present. Chart (b) shows the percentage of waking time steps in each one-minute interval. This waking pattern is the result of the learned action-value function as shown chart in (c). Following this sleep pattern, the BS gains rewards much faster than with the baseline always-on policy as shown in chart (d). Here sleeping gain is defined as the time averaged per-step reward.

As shown, the BS is almost always turned on during peak periods and seldom turned on during off-peak periods. In the transition regions between peaks and valleys, the agent issues mixed waking and sleeping operations. As the third case is more complex, we mainly focus on this case. We take two 2-minute time intervals and visualize the policy in finer detail in Fig. 5. For interval (a), the policy is roughly to gather more than two requests through queueing and serve them all together. This policy can help reduce the operational cost per request in intermittent traffic conditions as in (a). In interval (b), the traffic becomes more intensive (but still light compared to peak hours) and the agent adapts its policy to wake up the BS more aggressively. This new policy fits the current traffic condition better because the waiting cost may over-weigh the operating cost when the traffic becomes less intermittent, and therefore it is more beneficial to provide immediate service.

D. Algorithm Comparisons

We compare the performance of different algorithm configurations and the results are summarized in Table III. The investigated algorithms include:

TABLE III
TIME-AVERAGED PER-STEP REWARDS OF THE BASELINE ALWAYS-ON POLICY AND RELATIVE SLEEPING GAIN OF DIFFERENT ALGORITHM CONFIGURATIONS IN SIX DIFFERENT LOCATIONS.

Algorithms	**Locations**					
	dh3	dsy	dmW	mhC	mdB	gym
Per-step reward						
Baseline	-3.96	+2.49	-4.71	-4.28	-2.91	-4.42
Gain above baseline						
QL-d	3.280	1.384	3.678	3.014	2.695	3.420
DQN-m	3.443	1.616	3.898	3.228	2.673	3.578
DQN-d	**3.490**	1.879	**3.912**	**3.242**	**2.875**	**3.617**
DQN	3.481	**1.903**	3.880	3.238	2.863	3.600

- **Baseline:** the always-on policy agent.
- **DQN:** enhanced DQN using stacked raw observations.
- **DQN-m:** enhanced model-assisted DQN using continuous belief states.
- **DQN-d:** enhanced model-assisted DQN using continuous belief states and Dyna simulation.
- **QL-d:** table-based Q-learning agent using quantized belief states and Dyna simulation.

We calculated the time-averge per-step reward of each algorithm over the last 6 days of the experiment. Reward values are further averaged across at least 10 emulation runs. The improvement over the baseline performance is used as the metric for comparison.

As can be seen, deep reinforcement learning based configurations (DQN, DQN-m, DQN-d) consistently outperform table-based ones (QL-d) across all test cases. Note the only difference between DQN-d and QL-d is the presence of DQN. This demonstrate the advantage of deep reinforcement learning over table-based Q-learning. Due to the curse of dimensionality, table-based methods generally learn faster and better with small spaces. Fig. 6 corroborates with this argument, showing that QL-d performs better with more aggressive state quantization. In contrast, DQN-m and DQN-d can learn with un-quantized or lightly quantized state representations, and therefore avoids the information loss.

Interestingly, DQN-d performs slightly better than DQN in 5 out of 6 test cases. The performance gain may be contributed by two factors. Firstly, DQN-d leverages the domain knowledge through the IPP traffic model to extract more state from observations. In comparison, DQN has to learn such features end-to-end from raw observations, which should be much harder and slower. Second, DQN-d can use the learned environment model to generate pseudo experiences and train the DQN with more data. As shown in Fig. 7 the simulated experience can indeed help with the learning process.

Of course, the effectiveness of simulated data relies on the correctness of the learned model. We investigate the fitness of IPP model under real network trace in Fig. 8. See in the top chart, the learned traffic rate matches closely with the varying traffic volume. And also observe in the middle chart, IPP model can seamlessly fall back into a Poisson process in peak (with $P_{11} = 1$) and silent periods (with 0 emission rate). We also compare the observed and expected per-step data likelihood of the learned model as a quantitative goodness-of-

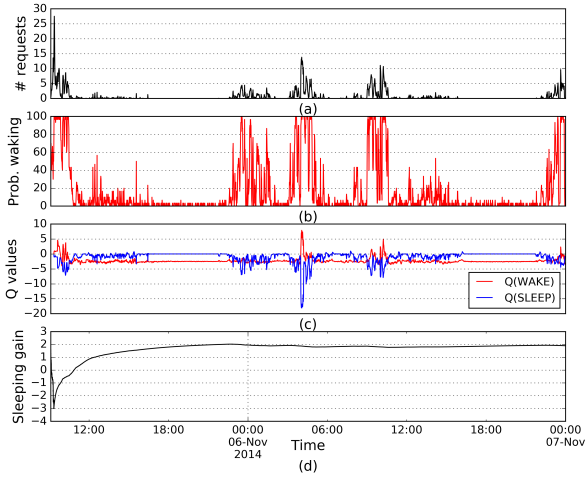


Fig. 4. Sleeping patterns of the DeepNap agent. (a) number of requests generated in each time step, (b) percentage of waking time steps, (c) Q values for waking and sleeping actions, (d) sleeping gain above the baseline always-on agent. All values are smoothed over one-minute time windows.

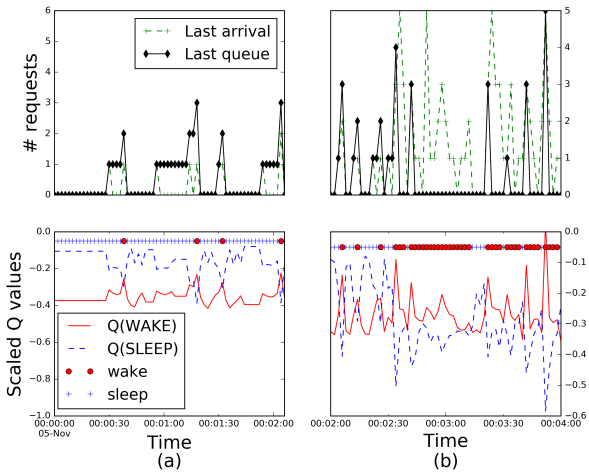


Fig. 5. Policy visualization for two time intervals (a) and (b). Top charts show the number of request arrivals and the queue length in last time step, which constitute the observations. Bottom charts show the Q values for waking and sleeping operations and the corresponding action.

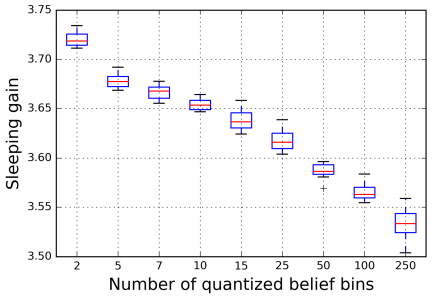


Fig. 6. Sleeping gain verses number of quantized belief state bins for QL-d at location dmW.

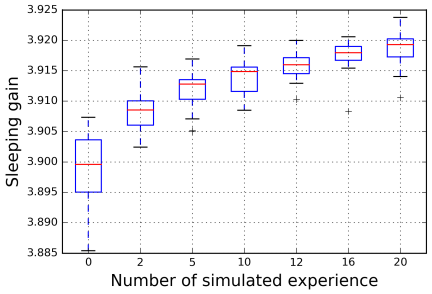


Fig. 7. Sleeping gain verses the number of simulated experiences for DQN-d at location dmW.

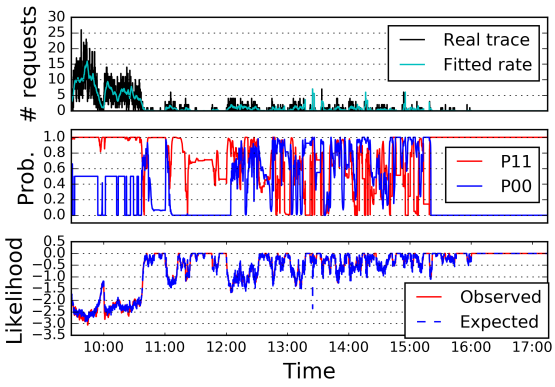


Fig. 8. Fitting results of the IPP model. Top: traffic and learned emission rates.; Middle: estimated transition probabilities; Bottom: Per-step likelihood values.

fit measure. As can be seen in the bottom chart, these two values roughly overlaps for most of the time, which proves that IPP is an appropriate model for the trace we use.

On the flip side, the slight performance improvement of DQN-d over DQN comes at a price. The model fitting and pseudo experience generation process incurs additional computational complexity for the algorithm. In resource constrained settings, it is reasonable to question whether these additional expenditures can be justified with such small performance gain. Also, the IPP model can also be wrong in other settings, calling for case-by-case model verification. In contrast, end-to-end DQN is model-free and can be straightforwardly applied to diffrent scenarios.

E. Action-Wise Experience Replay

Fig. 9(a) shows the sleeping gain of agents with and without action-wise replay memory. As shown, the agent with action-wise replay memory achieves a higher sleeping gain in the experiment. To better understand why action-wise replay memory is helpful, we also show the traffic, action distribution, and Q values during a short experiment period in Fig. 9(b). Because we sample $1/4$ of the memory each time, this distribution should be close to the actual action distribution in replay memory. Observe that the action distribution of the uniform-memory agent depends highly on the traffic regime. In traffic peaks and valleys, the distribution becomes highly biased towards waking or sleeping actions, respectively. The Q values for the minority action cannot be properly estimated during these highly biased periods. For

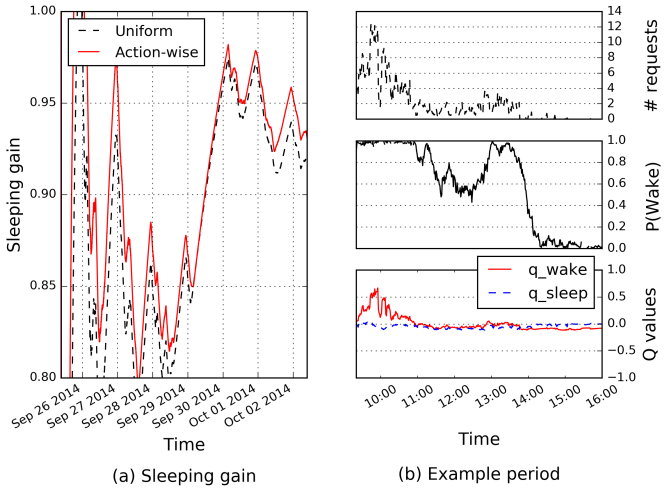


Fig. 9. Effectiveness of action-wise experience replay. (a) shows the time-averaged sleeping gain using both uniform and action-wise experience replay. (b) shows the number of request arrivals, batch distribution, and Q values for uniform experience replay for $w = 0.7$.

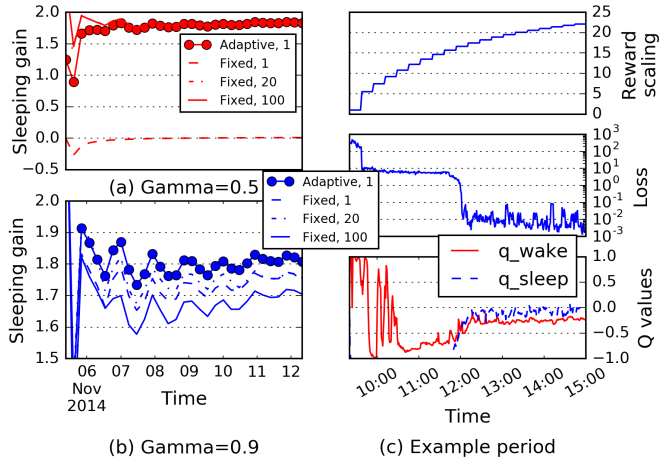


Fig. 10. Effectiveness of adaptive reward scaling. (a) and (b) show the sleeping gain with adaptive and fixed reward scaling. The discount factors are 0.5 and 0.9. (c) shows the variations of the reward scaling factor, training loss, and Q values during a short adaptation period.

instance before 11 : 00am, the traffic is high and the memory is dominated by waking experiences. The training process only tunes the waking Q value and leaves the sleeping Q value unchanged, despite the fact that the action value for sleeping should be small negative values in such high traffic. In comparison, the agent with action-wise replay memory is trained with balanced experiences and is less prone to such problems.

F. Adaptive Reward Scaling

Fig. 10(a) and Fig. 10(b) show the sleeping gain of DeepNap agents with different reward-scaling configurations and discount factor values. The fixed scaling factor needs to be carefully tuned to match with the different action value ranges of different γ: small scaling factors cause network saturation while large ones slow down the learning speed. In comparison,

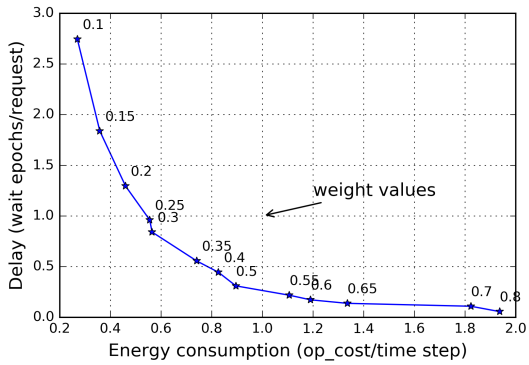


Fig. 11. Energy-delay tradeoff with different reward combining weights w. Smaller w values emphasize operational cost more, resulting in smaller energy cost and larger delay.

the adaptive reward scaling scheme with an initial factor of 1 can achieve a sleeping gain that is close to or better than the optimal fixed scaling factor regardless of the γ values. Shown in Fig. 10(c) are the variations of the reward scaling factor, training loss, and the estimated Q values during the process of adaptation. Initially, the scaling factor is relatively small and the rescaled reward is well beyond $(-1, +1)$. In consequence, the Q values are near saturation and the loss is dominated by the large saturation penalty. In response to this situation, the algorithm gradually increases the scaling factor such that the rescaled rewards fall back to the $(-1, +1)$ range. The network output is driven away from the saturated region and the penalty starts to diminish. The resulting loss is now starting to be dominated by estimation error. The loss then continues to drop as the action-value approximation becomes more accurate, and the reward scaling adaptation process also slows down.

G. Energy-Delay Tradeoff

A fundamental tradeoff in BS sleeping operations is the energy-delay tradeoff: more aggressive sleeping operations can save more network energy but at the expense of extra latency to the requests due to queuing In practice, it is very important flexibly balance these two conflicting interests to match the different preference of network operators and mobile users. The DeepNap agent can achieve such tradeoff by choosing different combining weights w for the traffic rewards and operational costs. Fig. 11 shows the average request delay and energy cost with different w. The figure shows that the policies learned by DeepNap trace out a monotonic tradeoff curve as the weight parameter w is increased.

V. Related Work

BS sleeping has long been identified as an effective approach to save system energy. Previous efforts to derive the optimal sleeping control policy have mainly adopted a model-based approach. For the simplistic setting of Poisson traffic arrival and exponential service time, the double-threshold hysteretic policy has been proven to be optimal [26], [27]. For more general traffic and service patterns, the double-threshold policy is not guaranteed to be optimal, still it is

often used as a baseline [6], [7]. When the optimal policy cannot be derived closed-form, it can be calculated numerically using value iteration methods under the Markov Decision Process (MDP) framework [28]. In contrast to these model-based previous works, DeepNap does not assert a fixed traffic model and learns an appropriate model in process, allowing it to adapt to varying traffic patterns. Besides, the end-to-end DQN configuration also provides a model-free method for BS sleeping operations.

In the broader context, RL has been widely applied to cognitive radio systems. Medium access control, resource management, and routing has also been formulated as RL problems in addition to node sleeping operations [29], [30], [31]. Among the common algorithms used to solve these problems, Q-learning is especially favorable due to its model-free nature. However, most previous work uses the canonical table-based Q-learning algorithm. But due to the curse of dimensionality, the possibly high-dimensional and continuous state space has to be reduced to a few discrete heuristic values. In comparison, DeepNap allows for high-dimensional and continuous states through value function approximation. Perhaps the most similar prior work to DeepNap is [32], in which a neural network is used to approximate the Q function for an interference control task. However, many of the important components, e.g. action-wise experience replay, and adaptive reward scaling, is not adopted [32].

DQN has been successfully applied to tackle problems in multiple domains, e.g. playing video games [11], [12], playing chess [13], and understanding human language [33]. DeepNap provides an additional application for this powerful framework. On top of the original DQN proposal, we also propose two new techniques, i.e. action-wise replay memory and adaptive reward scaling, to cope with non-stationary environments. Recent advances in this line is Prioritized Experience Replay [34], which samples experiences with non-uniform probabilities related to update losses. But this method is still not fit for non-stationary traffic, due to the same "memory dominace" phenomenon as uniform experience replay. In contrast, action-wise experience replay is specifically designed to cope with non-stationary environments. Moreover, the original DQN applies reward clipping to deal with the output range problem. Instead of clipping, we propose to adaptively find an appropriate scaling factor for reward values.

VI. Conclusions

In this paper, we propose a data-driven algorithm for dynamic base station sleeping control using deep reinforcement learning. In face of the non-stationarity in mobile network traffic, DeepNap enhances the original DQN algorithm with two modifications: action-wise experience replay for re-balancing the action distribution in the replay memory, and adaptive reward scaling for automatically searching an appropriate output scale in unknown environment. The proposed algorithm can also leverage the Dyna framework to incorporate an IPP traffic model into the DQN algorithm. We test DeepNap with real network traces and a data-driven network emulation framework. Experimental results show that action-wise experience replay and adaptive reward scaling improves the stability and adaptivity of vanilla DQN. The presence of DQN brings significant gain compared with table-based Q-learning algorithm. Moreover, the integration of IPP traffic model and the use of simulated experience also gives slight improvement over end-to-end DQN at the cost of more computation. For future work, more theoretical results from existing literature can be incorporated into the proposed data-driven framework to attain better performance.

References

[1] M. A. Marsan, L. Chiaraviglio, D. Ciullo, and M. Meo, "Optimal energy savings in cellular access networks," in *2009 IEEE International Conference on communications Workshops*, 2009, pp. 1–5.

[2] E. Oh, B. Krishnamachari, X. Liu, and Z. Niu, "Toward dynamic energy-efficient operation of cellular network infrastructure," *IEEE Communications Magazine*, vol. 49, no. 6, pp. 56–61, June 2011.

[3] K. Son, H. Kim, Y. Yi, and B. Krishnamachari, "Base station operation and user association mechanisms for energy-delay tradeoffs in green cellular networks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 8, pp. 1525–1536, 2011.

[4] Z. Niu, J. Zhang, X. Guo, and S. Zhou, "On energy-delay tradeoff in base station sleep mode operation," in *2012 IEEE International Conference on Communication Systems (ICCS)*, Nov 2012, pp. 235–239.

[5] X. Guo, S. Zhou, Z. Niu, and P. R. Kumar, "Optimal wake-up mechanism for single base station with sleep mode," in *2013 25th International Teletraffic Congress (ITC)*, Sept 2013, pp. 1–8.

[6] J. Wu, S. Zhou, and Z. Niu, "Traffic-aware base station sleeping control and power matching for energy-delay tradeoffs in green cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 8, pp. 4196–4209, 2013.

[7] J. Wu, Y. Bao, G. Miao, S. Zhou, and Z. Niu, "Base-station sleeping control and power matching for energy-delay tradeoffs with bursty ttraffic," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3657–3675, 2016.

[8] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, 1994.

[9] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary poisson view of internet traffic," in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, March 2004, pp. 1558–1569 vol.3.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, Dec. 2013, arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[14] J. Chen, K. Hu, Q. Wang, Y. Sun, Z. Shi, and S. He, "Narrowband internet of things: Implementations and applications," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2309–2314, Dec 2017.

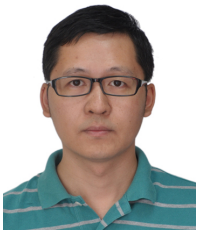
[15] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan 2018.

[16] L. Li, K. Ota, and M. Dong, "When weather matters: Iot-based electrical load forecasting for smart grid," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 46–51, Oct 2017.

[17] X. Cheng, L. Fang, L. Yang, and S. Cui, "Mobile big data: The fuel for data-driven wireless," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1489–1516, Oct 2017.
[18] X. Cheng, L. Fang, X. Hong, and L. Yang, "Exploiting mobile big data: Sources, features, and applications," *IEEE Network*, vol. 31, no. 1, pp. 72–79, January 2017.
[19] W. Fischer and K. Meier-Hellstern, "The markov-modulated poisson process (mmpp) cookbook," *Performance evaluation*, vol. 18, no. 2, pp. 149–171, 1993.
[20] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164–171, 1970.
[21] R. S. Sutton, "Integrated modeling and control based on reinforcement learning and dynamic programming," *Advances in neural information processing systems*, vol. 3, pp. 471–478, 1991.
[22] J. Liu, B. Krishnamachari, S. Zhou, and Z. Niu. (2017, Feb) Painting the DragonEye: From inanimate data to interactive control emulation of cellular networks. [Online]. Available: http://anrg.usc.edu/www/papers/DragonEye_ANRG_TechReport.pdf
[23] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[24] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135.
[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.
[26] D. P. Heyman, "Optimal operating policies for m/g/1 queuing systems," *Operations Research*, vol. 16, no. 2, pp. 362–382, 1968.
[27] I. Kamitsos, L. Andrew, H. Kim, and M. Chiang, "Optimal sleep patterns for serving delay-tolerant jobs," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. ACM, 2010, pp. 31–40.
[28] B. Leng, B. Krishnamachari, X. Guo, and Z. Niu, "Optimal operation of a green server with bursty traffic," in *2016 IEEE Global Communications Conference (Globecom)*, Dec 2016.
[29] K. A. Yau, P. Komisarczuk, and P. D. Teal, "Reinforcement learning for context awareness and intelligence in wireless networks: review, new features and open issues," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 253–267, 2012.
[30] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1136–1159, 2013.
[31] W. Wang, A. Kwasinski, D. Niyato, and Z. Han, "A survey on applications of model-free strategy learning in cognitive wireless networks," *arXiv preprint arXiv:1504.03976*, 2015.
[32] A. Galindo-Serrano and L. Giupponi, "Distributed q-learning for aggregated interference control in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1823–1834, 2010.
[33] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *arXiv preprint arXiv:1506.08941*, 2015.
[34] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

Jingchu Liu received his B.S. and Ph.D degrees from Department of Electronic Engineering of Tsinghua University, China, in 2012 and 2017, respectively. He is currently an algorithm engineer at Horizon Robotics, Inc. From October 2015 to April 2016, he visited the Autonomous Networks Research Group, Ming Hsieh Department of Electrical Engineering, University of Southern California, CA, USA. His research interests include cloud-based wireless networking, data-driven network management, network data analytics, and green wireless communications.

Bhaskar Krishnamachari is Ming Hsieh Faculty Fellow and Professor of Electrical Engineering and Computer Science at the Viterbi School of Engineering at the University of Southern California. He received his B.E. in Electrical Engineering from the Cooper Union in New York City in 1998, and his M.S. and Ph.D. in Electrical Engineering from Cornell University in 1999 and 2002, respectively. He is Director of the USC Viterbi Center for Cyber-Physical Systems and the Internet of Things (CCI). His research interests are in the design, analysis, implementation and empirical evaluation of algorithms and protocols for next-generation wireless networks, including for the Internet of Things. His work has received several best paper awards including at IPSN and Mobicom. He has received the NSF CAREER Award and the ASEE Terman Award. He is the author of a textbook titled Networking Wireless Sensors published by Cambridge University Press, and serves as an editor for the IEEE/ACM Transactions on Networking.



Sheng Zhou received the B.E. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2005 and 2011, respectively. From January to June 2010, he was a visiting student at the Wireless System Lab, Department of Electrical Engineering, Stanford University, Stanford, CA, USA. From November 2014 to January 2015, he was a visiting researcher in Central Research Lab of Hitachi Ltd., Japan. He is currently an Associate Professor with the Department of Electronic Engineering, Tsinghua University. His research interests include cross-layer design for multiple antenna systems, mobile edge computing, and green wireless communications.

Zhisheng Niu graduated from Northern Jiaotong University (currently Beijing Jiaotong University), China, in 1985, and got his M.E. and D.E. degrees from Toyohashi University of Technology, Japan, in 1989 and 1992, respectively. During 1992-94, he worked for Fujitsu Laboratories Ltd., Japan, and in 1994 joined with Tsinghua University, Beijing, China, where he is now a professor at the Department of Electronic Engineering. He was a Visiting Researcher at National Institute of Information and Communication Technologies (NICT), Japan (10/1995 C 02/1996), Hitachi Central Research Laboratory, Japan (02/1997 C 02/1998), Saga University, Japan (01/2001 C 02/2001), Polytechnic University of New York, USA (01/2002 C 02/2002), University of Hamburg, Germany (09/2014 C 10/2014), and University of Southern California, USA (11/2014-12/2014). His major research interests include queueing theory, traffic engineering, mobile Internet, radio resource management of wireless networks, and green communication and networks.

Dr. Niu has served as Chair of Emerging Technologies Committee (2014-15), Director for Conference Publications (2010-11), and Director for Asia-Pacific Board (2008-09) of IEEE Communication Society, Councilor of IEICE-Japan (2009-11), and a member of the IEEE Teaching Award Committee (2014-15) and IEICE Communication Society Fellow Evaluation Committee (2013-14). He has also served as associate editor-in-chief of IEEE/CIC joint publication China Communications (2012-16), editor of IEEE Wireless Communication (2009-13), editor of Wireless Networks (2005-09), and currently serving as an area editor of IEEE Trans. Green Commun. & Networks and Director for Online Content of IEEE ComSoc (2018-19).

Dr. Niu has published 100+ journal and 200+ conference papers in IEEE and IEICE publications and co-received the Best Paper Awards from the 13th, 15th and 19th Asia-Pacific Conference on Communication (APCC) in 2007, 2009, and 2013, respectively, International Conference on Wireless Communications and Signal Processing (WCSP13), and the Best Student Paper Award from the 25th International Teletraffic Congress (ITC25). He received the Outstanding Young Researcher Award from Natural Science Foundation of China in 2009 and the Best Paper Award from IEEE Communication Society Asia-Pacific Board in 2013. He was also selected as a distinguished lecturer of IEEE Communication Society (2012-15) as well as IEEE Vehicular Technologies Society (2014-16). He is a fellow of both IEEE and IEICE.