Experience-driven Networking: A Deep Reinforcement Learning based Approach

Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu and Dejun Yang

***Abstract*—Modern communication networks have become very complicated and highly dynamic, which makes them hard to model, predict and control. In this paper, we develop a novel experience-driven approach that can learn to well control a communication network from its own experience rather than an accurate mathematical model, just as a human learns a new skill (such as driving, swimming, etc). Specifically, we, for the first time, propose to leverage emerging Deep Reinforcement Learning (DRL) for enabling model-free control in communication networks; and present a novel and highly effective DRL-based control framework, DRL-TE, for a fundamental networking problem: Traffic Engineering (TE). The proposed framework maximizes a widely-used utility function by jointly learning network environment and its dynamics, and making decisions under the guidance of powerful Deep Neural Networks (DNNs). We propose two new techniques, TE-aware exploration and actor-critic-based prioritized experience replay, to optimize the general DRL framework particularly for TE. To validate and evaluate the proposed framework, we implemented it in ns-3, and tested it comprehensively with both representative and randomly generated network topologies. Extensive packet-level simulation results show that 1) compared to several widely-used baseline methods, DRL-TE significantly reduces end-to-end delay and consistently improves the network utility, while offering better or comparable throughput; 2) DRL-TE is robust to network changes; and 3) DRL-TE consistently outperforms a state-of-the-art DRL method (for continuous control), Deep Deterministic Policy Gradient (DDPG), which, however, does not offer satisfying performance.**

***Index Terms*—Experience-driven Networking, Deep Reinforcement Learning, Traffic Engineering**

I. Introduction

Extensive research efforts have been made to develop algorithms and protocols for communication networks to utilize their resources efficiently and effectively. Traditional network resource allocation methods are mostly model-based, which assume network environment and user demand can be well modeled. However, communication networks have become more complicated and highly dynamic, which makes them hard to model, predict and control. Hence, we aim to develop a novel experience-driven model-free approach that can learn to well control a communication network from its experience rather than an accurate mathematical model, just as a human learns a skill (such as driving, swimming, etc). We believe that some emerging networking technologies, such as Software Defined Networks (SDNs) [18], can well support such an experience/data driven approach. For example, the Openflow controller in an SDN can serve as the central control unit for collecting data, making decisions and deploying solutions.

A fundamental networking problem is the Traffic Engineering (TE): given a set of network flows with source and destination nodes, find a solution to forward the data traffic with the objective of maximizing a utility function. Simple and widely-used solutions include always routing traffic via shortest paths (e.g., Open Shortest Path First (OSPF) [24]); or evenly distributing traffic via multiple available paths (e.g., Valiant Load Balancing (VLB) [38]). Obviously, neither of them are optimal. Better solutions could be developed if there exist accurate and mathematically solvable models for network environment, user demands and their dynamics. Queueing theory has been employed to model communication networks and assist resource allocation [15], [25], [26], [37]. However, it may not work well for those networking problems involving multi-hop routing and end-to-end performance (such as delay) due to the following reasons: 1) In the queueing theory, many problems in a queueing network (rather than a single queue) remain open problems, while a communication network with a mesh-like topology represents a fairly complicated multi-point to multi-point queueing network where data packets from a queue may be distributed to multiple downstream queues, and a queue may receive packets from multiple different upstream queues. 2) The queueing theory can only provide accurate estimations for queueing delay under a few strong assumptions (e.g, tuple arrivals follow a Poisson distribution, etc), which, however, may not hold in a complex communication network. Note that even if the packet arrival at every source node follows a Poisson distribution, packet arrivals at intermediate nodes may not.

In addition, Network Utility Maximization (NUM) [17] has been well studied, which usually provides a resource allocation solution by formulating and solving an optimization problem. However, these methods may suffer from the following issues: 1) They usually assume that some key factors (such as user demands, link usages, etc) are given as input, which, however, are hard to estimate or predict. 2) It is hard to directly minimize end-to-end delay by explicitly including it in the utility function since given decision variables for resource allocation (such as TE), it is hard to express the corresponding end-to-end delay in a closed form with them since an accurate mathematical model is needed to achieve this (while queueing theory may not work here as described above). 3) Network dynamics have not been well addressed by these works. Most

Zhiyuan Xu, Jian Tang, Jingsong Meng and Yanzhi Wang are with Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244, USA. Email: {zxu105, jtang02, jmeng02, ywang393}@syr.edu. Weiyi Zhang is with AT&T Labs Research, Middletown, NJ 07748 USA. Chi Harold Liu is with Beijing Institute of Technology, Beijing, China, 100081. Dejun Yang is with Department of Electrical Engineering and Computer Science, Colorado School of Mines, Golden, CO 80401, USA. This research was supported in part by NSF grants 1704662, 1525920 and 1443966. The information reported here does not reflect the position or the policy of the federal government.

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

of them claimed to provide a "good" resource allocation solution, which is optimal or close-to-optimal but only for a snapshot of the network. However, most communication networks are highly time-varying. How resource allocation should be adjusted or re-computed to accommodate such dynamics has not been well addressed by these NUM methods.

Recent breakthrough of *Deep Reinforcement Learning (DRL)* [20] provides a promising technique for enabling effective experience-driven model-free control. *DRL* (originally developed by DeepMind) enables computers to learn to play games, including Atari 2600 video games and one of the most complicated games, Go (AlphaGo [29]), and beat the best human players. Even though DRL has made tremendous successes on game-playing that usually has a limited action space (e.g., moving up/down/left/right), it has not yet been investigated how DRL can be leveraged for resource allocation problems (such as TE) in complex communication networks, which usually have sophisticated states and huge or continuous action spaces.

We believe DRL is especially promising for control in communication networks because: 1) It has advantages over other dynamic system control techniques such as model-based predictive control in that the former is model-free and does not rely on accurate and mathematically solvable system models (such as queueing models), thereby enhancing its applicability in complex networks with random and unpredictable behaviors. 2) It is able to deal with highly dynamic time-variant environments such as time-varying system states and user demands. 3) It is capable of handling a sophisticated state space (such as AlphaGo [29]), which is more advantageous over traditional Reinforcement Learning (RL) [32]. However, direct application of the basic DRL technique, such as Deep Q-Network (DQN) based DRL (proposed in the pioneering work [20]), does not work for the TE problem since it is a continuous control problem (See Section IV); while DQN-based DRL is only capable of handling control problems with a limited action space. Although DRL methods have been proposed for continuous control very recently [8], [16], we show a state-of-the-art method, Deep Deterministic Policy Gradient (DDPG) [16], does not work well for our TE problem.

In this paper, we develop a novel and highly effective DRL-based model-free control framework for TE in a communication network to jointly learn network dynamics and making decisions under the guidance of powerful Deep Neural Networks (DNNs). We summarize our contributions in the following:

- We are the first to present a highly effective and practical DRL-based experience-driven control framework, DRL-TE, for TE.
- We discuss and show that direct application of a state-of-the-art DRL solution for continuous control, namely Deep Deterministic Policy Gradient (DDPG) [16], does not work well for the TE problem.
- We propose two new techniques, TE-aware exploration and actor-critic-based prioritized experience replay to optimize the general DRL framework particularly for TE.
- We show via extensive packet-level simulation using ns-3 [22] with both representative and random network topologies that DRL-TE significantly outperforms several widely-used baseline methods.

To the best of our knowledge, we are the first to leverage the emerging DRL for enabling model-free control in communication networks. We aim to promote a simple and practical experience-driven approach based on DRL, which, we believe, can be easily extended to solve many other resource allocation problems in communication networks.

II. Deep Reinforcement Learning (DRL)

We provide necessary background about DRL in this section. We consider a standard RL setup consisting of an agent interacting with an environment in discrete decision epochs. At each decision epoch t, the agent observes state $\mathbf{s}_t$, takes an action $\mathbf{a}_t$ and receives a reward r_t. The objective is to find a policy $\pi(\mathbf{s})$ mapping a state to an action (deterministic) or a probability distribution over actions (stochastic) with the objective of maximizing the discounted cumulative reward $R_0 = \sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)$, where $r(\cdot)$ is the reward function and $\gamma \in [0, 1]$ is the discount factor.

In the seminal work [20], DeepMind introduced DRL, which extends the well-known Q-learning to enable end-to-end system control based on high-dimensional sensory inputs (such as raw images). The training phase adopts a DNN called Deep Q-Network (DQN) to derive the correlation between each state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ of the system under control and its value function $Q(\mathbf{s}_t, \mathbf{a}_t)$, which is the expected discounted cumulative reward. If the system in state $\mathbf{s}_t$ and follows action $\mathbf{a}_t$ at decision epoch t (and a certain policy π thereafter):

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}\Big[R_t | \mathbf{s}_t, \mathbf{a}_t\Big], \tag{1}$$

where $R_t = \sum_{k=t}^{T} \gamma^k r(\mathbf{s}_t, \mathbf{a}_t)$. A commonly-used off-policy algorithm takes the greedy policy: $\pi(\mathbf{s}_t) = \mathrm{argmax}_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t)$. The DQN can be trained by minimizing the loss:

$$L(\boldsymbol{\theta}^Q) = \mathbb{E}\Big[y_t - Q(\mathbf{s}_t, \mathbf{a}_t | \boldsymbol{\theta}^Q)\Big], \tag{2}$$

where $\boldsymbol{\theta}^Q$ is the weight vector of the DQN and y_t is the target value, which can be estimated by:

$$y_t = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}|\boldsymbol{\theta}^\pi)|\boldsymbol{\theta}^Q). \tag{3}$$

It is not new to use a neural network (or even DNN) as the function approximator in RL. But a non-linear function approximator (such as neural network) is known to be unstable or even to diverge. Two effective techniques were introduced in [20] to improve stability: experience relay and target network. Unlike traditional RL, a DRL agent updates the DNN with a mini-batch from an experience replay buffer [20], which stores state transition samples collected during learning. Compared to using only immediately collected samples (such as original Q-learning), randomly sampling from the experience replay buffer allows the DRL agent to break the correlation between sequentially generated samples, and learn

from a more independently and identically distributed past experiences, which is required by most of training algorithms, such as Stochastic Gradient Descent (SGD). So experience replay can smooth out learning and avoid oscillations or divergence. In addition, a DRL agent uses a separate target network (which has the same structure as the DQN) to estimate target values $< y_t >$ for training the DQN, whose parameters, however, are slowly updated with the DQN weights every $C > 1$ epochs and are held fixed between individual updates.

The traffic engineering problem (described next) is a continuous control problem. Unfortunately, the DQN-based DRL only works for control problems with a low-dimensional discrete action space. It cannot be easily applied to continuous control since it needs to find the action that maximizes the action-value function, which, however, requires an iterative process to solve a non-trivial non-linear optimization problem at each epoch. A straightforward solution to adapting DQN-based approach to continuous cases is to simply discretize the action space, which, however, may likely leads to a huge number of actions, which are very hard to deal with too.

Continuous control has often been tackled by the actor-critic approach [14], which usually employs the policy gradient method to search for the optimal policy. The traditional actor-critic approach can also be extended to embrace DNN (such as DQN) to guide decision making [16]. For example, a recent work [16] from DeepMind introduced an actor-critic method, called Deep Deterministic Policy Gradient (DDPG), for continuous control. The basic idea is to maintain a parameterized actor function $\pi(\mathbf{s}_t|\boldsymbol{\theta}^\pi)$ and a parameterized critic function $Q(\mathbf{s}_t, \mathbf{a}_t|\boldsymbol{\theta}^Q)$. The critic function can be implemented using the above DQN, which returns Q value for a given state-action pair. The actor function can also be implemented using a DNN, which specifies the current policy by mapping a state to a specific action. According to [28], the actor network can be updated by applying the chain rule to the expected cumulative reward J with respect to the actor parameters $\boldsymbol{\theta}^\pi$:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}^\pi} J &\approx \mathbb{E}\Big[\nabla_{\boldsymbol{\theta}^\pi} Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^Q)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi(\mathbf{s}_t|\boldsymbol{\theta}^\pi)}\Big] \\ &= \mathbb{E}\Big[\nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^Q)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi(\mathbf{s}_t)} \cdot \nabla_{\boldsymbol{\theta}^\pi} \pi(\mathbf{s}|\boldsymbol{\theta}^\pi)|_{\mathbf{s}=\mathbf{s}_t}\Big].\end{aligned} \tag{4}$$

Note that the experience replay and target network introduced above can also be integrated to this approach to ensure stability.

III. Problem Statement

We describe the Traffic Engineering (TE) problem in this section. First, we summarize the major notations below for quick reference.

We consider a general communication network with K end-to-end communication sessions. We use a directed graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ to model the network, where each vertex corresponds to a node (router or switch) and each edge corresponds to a directed communication link connecting a pair of nodes. Each communication session k has a source node s_k, destination d_k and a set of candidate paths $\mathbf{P}_k$ (connecting s_k with d_k) that can carry its traffic load. As mentioned above, we aim to study a TE problem seeking a rate allocation solution, which specifies the amount of traffic load $f_{k,j}$ going through the jth path of $\mathbf{P}_k$. Note that once we have such a solution, then when a packet of session k arrives at s_k, path j is chosen to transmit the packet with a probability of $w_{k,j}$, where $w_{k,j} = f_{k,j}/(\sum_{j=1}^{|P_k|} f_{k,j})$, which is known as the split ratio.

TABLE I: Notation Definition

Variable	Definition
K	The number of communication sessions
$\mathbf{P}_k$	The set of candidate paths of session k
$\mathbf{E}$	The set of links of the network
B_k	Traffic demand of session k
C_e	Capacity of link e
$f_{k,j}$	The amount of traffic of the jth path of session k
$w_{k,j}$	Split ratio for the jth path of session k
x_k, z_k	Throughput and delay of session k
$\mathbf{s}$, $\mathbf{a}$, r	State, action and reward
$p_i, P(i)$	Priority and probability (being selected) of transition sample i
$\boldsymbol{\theta}^\pi, \boldsymbol{\theta}^Q$	Weights of actor and critic networks $\pi(\cdot)$ and $Q(\cdot)$

The α-fairness [31], [36] model has been widely used for NUM. According to this model, the utility of a communication session with a steady-state throughput of x is $U_\alpha(x) = (\frac{x^{1-\alpha}}{1-\alpha})$. Particularly, as $\alpha \to 1$, in the limit $U_1(x)$ becomes $\log x$ [36]. For $\alpha > 0$, $U_\alpha(x)$ is monotonically increasing with x. The objective of the TE problem is usually set to maximizing the total utility of all the communication sessions, i.e., $\sum_{k=1}^{K} U_\alpha(x)$. α can be used to tradeoff fairness and efficiency. If $\alpha = 1$, the objective is to achieve the proportional fairness, which is widely used for resource allocation.

In order to address both throughput and delay, similar as in [36], we define a utility function $U(\cdot)$ for session k:

$$U(x_k, z_k) = U_{\alpha_1}(x_k) - \sigma \cdot U_{\alpha_2}(z_k), \tag{5}$$

where x_k and z_k are the end-to-end throughput and delay of session k respectively; and σ expresses the relative importance of delay vs. throughput. Similarly, the objective of the TE problem is to maximize the total utility of all the communication sessions in the network, i.e., $\sum_{k=1}^{K} U(x_k, z_k)$.

Note that we aim to consider a general communication network and show how DRL can enable experience-driven networking rather than targeting at a specific physical network (such as SDN, multihop wireless network) or a specific scenario (such as WAN, MAN, LAN, etc). So we try to make system model and problem statement as general as possible. However, the proposed control framework (Section IV) is so flexible that it can be easily extended for a specific network or scenario with additional constraints.

IV. Proposed DRL-based Control Framework

In this section, we present the proposed DRL-based control framework, DRL-TE, for the TE problem described above.

In order to utilize the DRL techniques (no matter which method/model to use), we first need to design the state space, action space and reward function.

- *State Space*: The state consists of two components: throughput and delay of each communication session. Formally, the state vector $\mathbf{s} = [(x_1, z_1), \cdots, (x_k, z_k), \cdots, (x_K, z_K)]$.
- *Action Space*: An action is defined as the solution to the TE problem, i.e., the set of split ratios for the communication sessions. Formally, the action vector $\mathbf{a} = [w_{1,1}, \cdots, w_{kj}, \cdots, w_{K,|P_k|}]$, where $\sum_{j=1}^{|P_k|} w_{k,j} = 1$.
- *Reward*: The reward is the objective of the TE problem, which is the total utility of all the communication sessions. Formally, $r = \sum_{k=1}^{K} U(x_k, z_k)$.

Note that the design of state space, action space and reward is critical to the success of a DRL method. Our design well captures network states and the key components of the TE problem without including useless/redudant information. The core of the proposed control framework is an agent, which runs a DRL algorithm (Algorithm 1) to find the best action at each decision epoch, takes the action to the network (e.g., through a network controller) observes the network state, and collects a transition sample.

The TE problem is obviously a continuous control problem. As explained above, the DQN-based DRL proposed in the well-known work [20] does not work here; so we choose the state-of-the-art DRL-based solution for continuous control, DDPG [16], as the starting point for our design, whose basic idea has been introduced in Section II.

Even though DDPG has been demonstrated to work well on quite a few continuous control tasks [16], our experimental results, however, show that direct application of DDPG to the TE problem does not lead to satisfying performance (Section V). We suspect this is due to the following two reasons: 1) The DDPG framework in [16] does not clearly specify how to explore. A simple random noise based method or the exploration methods proposed for physical control problems (mentioned in [16]) do not work well for the TE problem here. 2) DDPG utilizes a simple uniform sampling method for experience replay, which ignores the significance of transition samples in the replay buffer. To address these two issues, we propose two new techniques to optimize DDPG particularly for TE, including TE-aware exploration which leverages a good TE solution as the baseline during exploration; and actor-critic-based prioritized experience replay which can employs a new method for specifying significance of samples with careful consideration for both the actor and critic networks. Exploration is an essential and important process for training a DRL agent because an inexperienced agent needs to see sufficient transition samples to gain experience and eventually learn a good (hopefully optimal) policy. For continuous control problems, exploration is quite challenging because there are infinite number of actions that can be chosen in each decision epoch and the commonly-used ϵ-greedy method [20] only works for tasks with a limited discrete action space, which obviously does not work here. DDPG generates an action for exploration by adding a random noise to the action returned by the current actor network.

For exploration, we propose a new randomized algorithm that guides the exploration process with a base TE solution. Specifically, with ϵ probability, the DRL agent derives action as $\mathbf{a}_{\text{base}} + \epsilon \cdot \mathcal{N}$; and with $(1-\epsilon)$ probability, it derives action as $\mathbf{a} + \epsilon \cdot \mathcal{N}$; where $\mathbf{a}_{\text{base}}$ is a base TE solution, $\mathbf{a}$ is the output of actor network $\pi(\cdot)$ and ϵ is an adjustable parameter. ϵ can tradeoff exploration and exploitation by determining the probability of adding a random noise to the action rather than taking the derived action from the actor network. ϵ decays with decision epoch t, which means with more learning, more derived (rather than random) actions will be taken. The parameter $\mathcal{N}$ is a uniformly distributed random noise.

The proposed control framework is not restricted to any specific base TE solution for $\mathbf{a}_{\text{base}}$, which can be obtained in many different ways. For example, a simple solution is to use the shortest path to deliver all the packets for each communication session, which is not optimal in most cases but is good enough to sever as a baseline for exploration. Another solution is to evenly distribute traffic load of each communication session to all candidate paths. NUM-based methods can also be used to find base solutions. For example, we can obtain a TE solution by solving the following mathematical programming:
NUM-TE:

$$\max_{<x_k, f_{k,j}>} \sum_k U_\alpha(x_k) \tag{6a}$$

subject to:

$$\sum_{k=1}^{K} \sum_{\mathbf{p}_j \in \mathbf{P}_k : e \in \mathbf{p}} f_{k,j} \le C_e, \forall\ e \in \mathbf{E}; \tag{6b}$$

$$x_k \le B_k,\ k \in \{1, \cdots, K\}; \tag{6c}$$

$$\sum_{j=1}^{|\mathbf{P}_k|} f_{k,j} = x_k,\ k \in \{1, \cdots, K\}. \tag{6d}$$

In this formulation, the objective is to maximize the total utility in terms of throughput. Note that it is hard to include the end-to-end delay term in the utility function since there does not exists a mathematical model that can accurately establish a connection between end-to-end delay and the other decision variables $< x_k, f_{k,j} >$. This is why end-to-end delay has not been well addressed by most existing works on NUM. Constraints (6b) ensure the aggregated traffic load on each link does not exceed its capacity C_e, where $\mathbf{p}_j$ is the jth path in $\mathbf{P}_k$. Constraints (6c) ensure the total throughput of each session k does not exceed its demand B_k, which can be estimated. Constraints (6d) establish the connections between two set of decision variables $< x_k >$ and $< f_{k,j} >$. If $\alpha = 1$, $U_\alpha(x_k) = \log x_k$, then this problem becomes a convex programming problem, which can be efficiently solved by the Gurobi Optimizer [10] that were used in our implementation.

DDPG simply uniformly samples transition data from the experience replay. It has been shown by [30] that an DRL

agent can learn more effectively from some transitions than others. A method called prioritized experience replay has also been introduced in [30], which has been shown to lead to better performance on game-playing tasks when being combined with DQN. It assigns a priority for each transition sample. Based on this priority, transition data in the replay buffer are sampled in each epoch. However, this method was proposed only for DQN-based DRL and has never been used with the actor-critic method for continuous control. We extend this method to enable prioritized experience replay under the actor-critic framework. Specifically, since an actor-critic method uses two networks (actor and critic) to guide decision making, the priority should consist of two parts. The first part is the Temporal-Difference (TD) error, which corresponds to training of the critic network:

$$\delta = y - Q(\mathbf{s}, \mathbf{a}), \tag{7}$$

where y is the target value for training the critic network, which is defined in Equation (3). Note that to help understand the basic idea better, we omit the subscripts/superscripts here for clean presentation; the exact forms of these equations can be found at the formal algorithm presentation. The actor and critic network are jointly trained by transition samples in the replay buffer. The second part is related to training of the actor network, i.e., the Q gradient $\nabla_{\mathbf{a}} Q = \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\pi(\mathbf{s}_i)}$ (Equation (4)). Combining them together, the priority of a transition sample is given as:

$$p = \varphi \cdot (|\delta| + \xi) + (1 - \varphi) \cdot \overline{|\nabla_{\mathbf{a}} Q|}, \tag{8}$$

where φ is a parameter controlling the relative importance of TD error vs. Q gradient. $\overline{|\nabla_{\mathbf{a}} Q|}$ is the average of absolute values of the Q gradient (which is a vector). A small positive constant ξ is used to prevent the edge-cases of transitions not being revisited once their error is zero. The probability of sampling transition i is:

$$P(i) = \frac{p_i^{\beta_0}}{\sum_j^{|\mathbf{B}|} p_j^{\beta_0}}, \tag{9}$$

where the exponent β_0 determines how much prioritization is used; if $\beta_0 = 0$, then it becomes uniform sampling.

We formally present the proposed DRL-based control framework for TE, DRL-TE, as Algorithm 1. First the algorithm randomly initializes all the weights $\boldsymbol{\theta}^{\pi}$ of actor network $\pi(\cdot)$; and $\boldsymbol{\theta}^{Q}$ of the critic networks $Q(\cdot)$(line 1). As mentioned above, we employ target networks $\pi'(\cdot)$ and $Q'(\cdot)$ to improve learning stability. The target networks are clones of the original actor or critic networks, whose weights $\boldsymbol{\theta}^{\pi'}$ and $\boldsymbol{\theta}^{Q'}$ are initialized in the same way as their original networks (line 2) but are slowly following updated (line 23). The update rate is controlled by a parameter τ. In each decision epoch, the algorithm applies the TE-aware exploration method to obtain an action first (line 6), which is explained above.

We use a prioritized replay buffer for storing transition samples. We first store the sample into the replay buffer with maximal priority (line 8), and then sample a mini-batch of transition samples from $\mathbf{B}$ (lines 10-19) to train the actor and critic networks. The priority of transition is then updated using the method described right above (lines 16-18). Note that for every transition sample $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})$ in the mini-batch, we first obtain its important-sampling weight ω (line 12), which is used to correct the bias introduced by prioritized replay [30]. The weight is integrated into the critic network updating in the form of $\omega \cdot \delta$ (rather than δ only) (line 17). Priorities ensure high-error transitions are seen more frequently. Those large steps (with large priority) can be very disruptive because of large updating values. As suggested by [30], we import

Algorithm 1: DRL-TE

1: Randomly initialize critic network $Q(\cdot)$ and actor network $\pi(\cdot)$ with weights $\boldsymbol{\theta}^{Q}$ and $\boldsymbol{\theta}^{\pi}$ respectively;
2: Initialize target networks $Q'(\cdot)$ and $\pi'(\cdot)$ with weights $\boldsymbol{\theta}^{Q'} := \boldsymbol{\theta}^{Q}$, $\boldsymbol{\theta}^{\pi'} := \boldsymbol{\theta}^{\pi}$;
3: Initialize prioritized replay buffer $\mathbf{B}$ and $p_1 := 1$;
/**Online Learning**/
4: Receive the initial observed state $\mathbf{s}_1$;
/**Decision Epoch**/
5: **for** t = 1 **to** T **do**
6: Apply the TE-aware exploration method to obtain $\mathbf{a}_t$;
7: Execute action $\mathbf{a}_t$ and observe the reward r_t;
8: Store transition sample $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ into $\mathbf{B}$ with maximal priority $p_t = \max_{j<t} p_j$;
9: /**Prioritized Transition Sampling**/
10: **for** i = 1 **to** N **do**
11: Sample a transition $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})$ from $\mathbf{B}$ where $i \sim P(i) := p_i^{\beta_0} / \sum_j p_j^{\beta_0}$;
12: Compute important-sampling weight: $\omega_i := (|\mathbf{B}| \cdot P(i))^{-\beta_1} / \max_j \omega_j$;
13: Compute target value for critic network: $Q(\cdot)$ $y_i := r_i + \gamma \cdot Q'(\mathbf{s}_{i+1}, \pi'(\mathbf{s}_{i+1}))$;
14: Compute TD-error: $\delta_i := y_i - Q(\mathbf{s}_i, \mathbf{a}_i)$;
15: Compute gradient: $\nabla_{\boldsymbol{\theta}^{\pi}} J_i := \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\pi(\mathbf{s}_i)} \cdot \nabla_{\boldsymbol{\theta}^{\pi}} \pi(\mathbf{s})|_{\mathbf{s}=\mathbf{s}_i}$;
16: Update the transition priority: $p_i := \varphi \cdot (|\delta_i| + \xi) + (1 - \varphi) \cdot \overline{|\nabla_{\mathbf{a}} Q|}$;
17: Accumulate weight-change for critic network: $Q(\cdot)$ $\Delta_{\boldsymbol{\theta}^{Q}} := \Delta_{\boldsymbol{\theta}^{Q}} + \omega_i \cdot \delta_i \cdot \nabla_{\boldsymbol{\theta}^{Q}} Q(\mathbf{s}_i, \mathbf{a}_i)$;
18: Accumulate weight-change for actor network: $\pi(\cdot)$ $\Delta_{\boldsymbol{\theta}^{\pi}} := \Delta_{\boldsymbol{\theta}^{\pi}} + \omega_i \cdot \nabla_{\boldsymbol{\theta}^{\pi}} J_i$;
19: **end for**
20: /**Network Updating**/
21: Update the weights of critic network: $Q(\cdot)$ $\boldsymbol{\theta}^{Q} := \boldsymbol{\theta}^{Q} + \eta^{Q} \cdot \Delta_{\boldsymbol{\theta}^{Q}}$, reset $\Delta_{\boldsymbol{\theta}^{Q}} := 0$;
22: Update the weights of actor network: $\pi(\cdot)$ $\boldsymbol{\theta}^{\pi} := \boldsymbol{\theta}^{\pi} + \eta^{\pi} \cdot \Delta_{\boldsymbol{\theta}^{\pi}}$, reset $\Delta_{\boldsymbol{\theta}^{\pi}} := 0$;
23: Update the weights of the corresponding target networks: $\boldsymbol{\theta}^{Q'} := \tau \boldsymbol{\theta}^{Q} + (1 - \tau) \boldsymbol{\theta}^{Q'}$; $\boldsymbol{\theta}^{\pi'} := \tau \boldsymbol{\theta}^{\pi} + (1 - \tau) \boldsymbol{\theta}^{\pi'}$;
24: **end for**

annealing weight β_1 to correct this bias, by linearly annealing it from its initial value to 1 (line 12). For learning stability, we always normalize ω by $1/\max_j \omega_j$, so they only scale the weight update downward. We obtain the action for the next state from target actor network $\pi'(\mathbf{s}_{i+1})$, and the target value y_i (line 13) for training the critic network; In addition, we compute the policy gradient by the chain rule, as described in Equation (4) (line 15). The weight-changes are accumulated (lines 17-18) and used to update the actor and critic networks (lines 21-22).

There are quite a few hyper-parameters in the proposed control framework. To maximize its performance, we conducted a comprehensive empirical study to find the best settings for them and the best structures of the actor and critical networks. In our design and implementation, we used a 2-layer fully-connected feedforward neural network to serve as the actor network, which includes 64 and 32 neurons in the first and second layer respectively and utilized the Leaky Rectifier [7] for activation. In the final output layer, we, however, employed the softmax [7] as activation function to ensure the sum of output values equals one. For the critic network, we also used a 2-layer fully-connected feedforward neural network, with 64 and 32 neurons in the first and second layer respectively and with the Leaky Rectifier for activation. In order to sample N transitions with probabilities given by Equation (9), the range [0, p_{total}] is divided into N sub-ranges, and a transition is uniformly sampled from each sub-range, where p_{total} is the sum of priorities of all transitions in replay buffer. As suggested by [30], we used a sum-tree to implement the priority probability, which is similar to a binary heap. The differences are 1) leaf nodes store the priorities of transitions; and 2) internal nodes store the sum of its children. In this way, the value of root is p_{total}, and the time complexity for updating and sampling is $O(\log N_{\text{tree}})$, where N_{tree} is the number of nodes in the sum-tree. During the empirical study, we also found good settings for the other important hyper-parameters: $\xi := 0.01$, $\beta_0 := 0.6$, $\beta_1 := 0.4$, $\gamma := 0.99$, $\varphi := 0.6$, $\eta^{\pi} := 0.001$, $\eta^{Q} := 0.01$, $\tau := 0.01$ and $N = 64$.

V. Performance Evaluation

We conducted extensive simulation to evaluate the performance of the proposed DRL-based framework. We present and analyze the simulation results in this section. We implemented the proposed framework and set up the environment in ns-3 [22] for packet-level simulation. The DNNs included in the framework (i.e., the actor and critic networks) were implemented using Tensorflow [33]. Due to the light wight of our design, we found that we could easily run and train the proposed framework (along with the corresponding DNNs) on a regular desktop with an Intel Quad-Core 2.6Ghz CPU with 8GB memory.

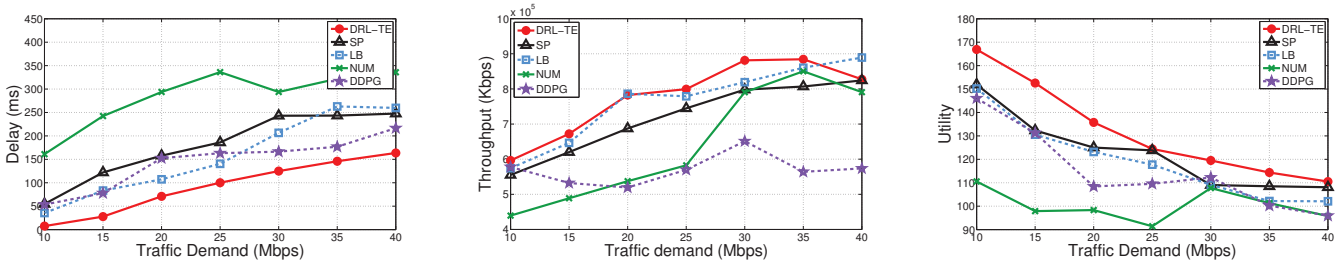
The simulation runs were performed on two well-known network topologies, NSF Network (NSFNET [23]) and Advanced Research Projects Agency Network (ARPANET [1]). Besides, we randomly generated a network topology with 20 nodes and 80 links, using the widely-used network topology generator, BRITE [19]. For each network topology, we assigned $K = 20$ communication sessions, each with randomly selected source and destination nodes. For each communication session, we selected 3-shortest paths (in terms of hop-count) as its candidate paths. The capacity of each link was set to 100Mbps. The packet arrival at the source node of each communication session (i.e., traffic demand) follows a Poisson process (note that the packet arrivals at intermediate nodes may not follow a Poisson process), with its mean value uniformly distributed within a window with a size of 20Mbps. In our experiments, we set the window to $[0, 20]$Mbps initially, and we increased the traffic demand by sliding the window with a step size of 5Mbps for each run. We set $\alpha := 1$ and $\sigma := 1$ for the utility function to balance throughput, delay and fairness, i.e. the objective/utility function became $\sum_{k=1}^{K}(\log x_k - \log z_k)$.

We compared our DRL-based control framework with three widely used baseline solutions as well as DDPG [16]:

- Shortest Path (SP): every communication session uses a shortest path to deliver all its packets.
- Load Balance (LB): every communication session evenly distributes its traffic load to all candidate paths.
- Network Utility Maximization (NUM): it obtains TE solutions by solving the convex programming problem, NUM-TE given in Section IV.
- DDPG: For fair comparison, we replaced the DRL-TE algorithm (Algorithm 1) with the DDPG algorithm [16], while keeping the other settings (such as state, action, reward and the DNNs) the same.

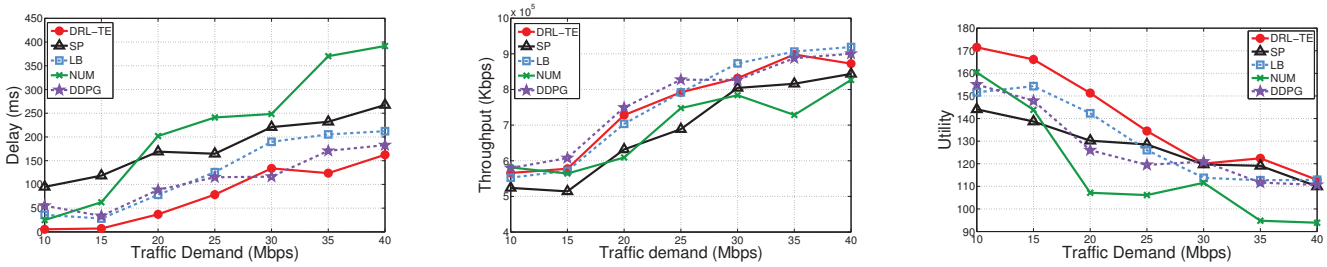
We used the total end-to-end throughput, the end-to-end average packet delay, and the network (i.e., total) utility value as the performance metrics for comparisons. We present the corresponding simulation results in Figs. 1-3, each of which corresponds to a network topology. Note that the numbers on the x-axis are the central values of the corresponding traffic demand windows (mentioned above). In addition, we show the performance of two DRL methods (DDPG and DRL-TE) over the three network topologies during the online learning procedure in terms of the reward. For illustration and comparison purposes, we normalized and smoothed the reward values using a commonly-used method $(r - r_{\min})/(r_{\max} - r_{\min})$ (where r is the actual reward, $r_{\min}$ and $r_{\max}$ are the minimum and maximum rewards during online learning respectively) and the well-known forward-backward filtering algorithm [11] respectively. We present the corresponding simulation results in Fig. 4. Note that for these results, the corresponding traffic demand was generated using window $[10, 30]$Mbps. We can make the following observations from these results.

1) From Figs. 1a, 2a and 3a, we can see that compared to all the four baseline methods, DRL-TE significantly reduces end-to-end delay on all the three topologies. For example, on the NSF topology, when the traffic load is medium (i.e., traffic demand window is $[10, 30]$Mbps), DRL-TE significantly reduces the end-to-end delay by 51.6%, 28.6%, 74.6% and 50.0% respectively, compared to SP, LB, NUM and DDPG.



(a) End-to-end delay (b) End-to-end throughput (c) Network utility

Fig. 1: Performance of all the methods over the NSFNET topology



(a) End-to-end delay (b) End-to-end throughput (c) network utility

Fig. 2: Performance of all the methods over the ARPANET topology

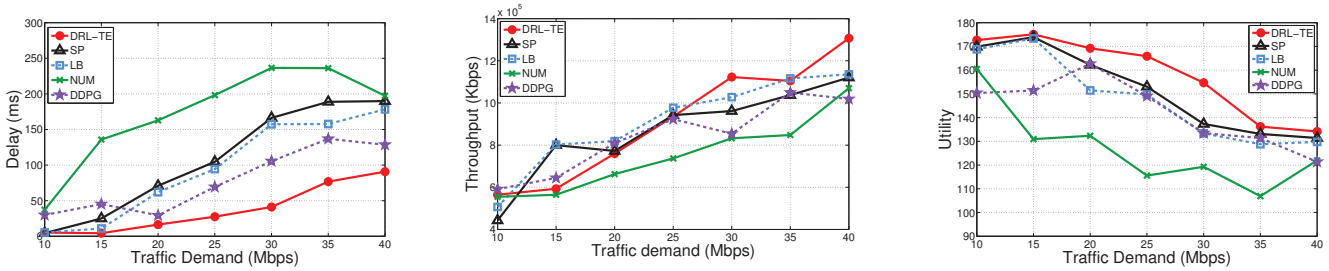
Overall, DRL-TE achieves an average reduction of 55.4%, 47.1%, 70.5% and 44.2% respectively. Compared to throughput, end-to-end delay is harder to deal with since as discussed above, it lacks accurate mathematical models that can well capture its characteristics and runtime dynamics. It is not surprising to see NUM leads to fairly poor performance since it fails to explicitly address end-to-end delay and its design is based on the assumption that network state is fairly stable or slowly changes, which may not be true; while simple solutions such as SP and LB offers expected performance since intuitively, the shortest paths and load balancing (which can avoid congestions) can help reduce delay. DRL-TE unarguably delivers superior performance with regards to end-to-end delay because it keeps learning runtime dynamics and making wise decisions to move to the optimal with the help of DNNs.

2) Even though the objective (reward function) of DRL-TE is not to simply maximize end-to-end throughput, it still delivers satisfying performance, as shown in Figs. 1b, 2b and 3b. Compared to all the other methods, DRL-TE leads to consistently higher throughput on the NSFNST topology. On both the ARPANET and random topologies, the throughput values given by DRL-TE are comparable to those given by LB (load balancing is supposed to yield high throughout), but still higher than those offered by SP and NUM.

3) As expected, we can see from Figs. 1c, 2c and 3c that DRL-TE outperforms all the other methods in terms of the total utility because its reward function is set to maximizing it. On average, DRL-TE outperforms SP, LB, NUM and DDPG by 7.7%, 9.1%, 26.4% and 12.6% respectively.

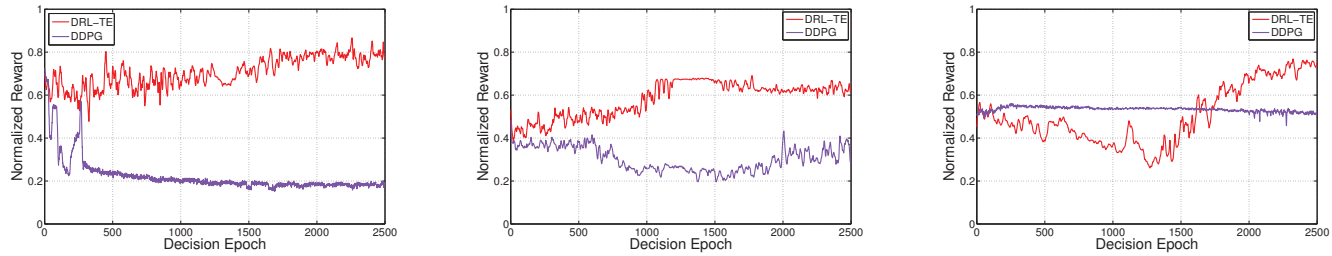
4) From Figs. 1-3, we can observe no matter which method is used and no matter which network topology is chosen, the throughput and delay basically go up with the traffic demand; while the total utility generally go down. This is easy to understand because the higher the traffic load, usually the higher the throughput, but the higher the delay due to longer waiting time or even congestion, which brings down the total utility. Moreover, the throughput does not increase monotonically, when the network becomes saturated, higher traffic demands may even lead to poorer throughput due to congestion and packet losses. We also notice that DRL-TE is robust to changes of traffic load and network topology since it performs consistently better than all the other methods across all the traffic demand settings and all the topologies.

5) In addition, we can also observe from Figs. 1-3 that DDPG does not work very well on these topologies. For example, compared to SP and LB, it performs generally worse in terms of the total utility, even though it provides slightly better end-to-end delay. To further explain why DRL-TE works better than DDPG, we also show how the reward value changes during online learning over the three network topologies in Fig. 4. Clearly, over all these network topologies, DRL-TE quickly (within just a couple of thousands of decision epoches) reaches a good solution (that gives a high reward); while DDPG seems to be stuck at local optimal solutions with lower reward values. Particularly, on the random topology, we can only see minor improvement on the first few hundred epoches, then it fails to find better solutions (actions) to improve the reward. These results clearly justify the effectiveness of the proposed new techniques including TE-aware exploration and the actor-critic-based prioritized experience replay.



(a) End-to-end delay (b) End-to-end throughput (c) Network utility

Fig. 3: Performance of all the methods over the random topology



(a) NSFNET topology (b) ARPANET topology (c) Random topology

Fig. 4: Reward over the three network topologies during online learning

VI. Related Work

Traffic Engineering (TE) and Network Utility Maximization (NUM): TE and NUM have been well studied in the literature. In a seminal work [17], Low and Lapsley proposed asynchronous distributed algorithms to solve a flow control problem whose objective is to maximize the aggregate source utility over their transmission rates. In [26], Palomar and Chiang, introduced primal, dual, indirect, partial, and hierarchical decompositions, focusing on NUM problems and the meanings of primal and dual decompositions in terms of network architectures. In [25], the authors designed a congestion control system that scales gracefully with multiple objectives, which was built on decentralized control laws at end-systems. Xu *et al.* [37] proposed a new link-state routing protocol PEFT, which splits traffic over multiple paths with an exponential penalty on longer paths, with hop-by-hop forwarding, with the objective of achieving optimal TE. The authors of [15] proposed algorithms to solve a NUM problem in a network with delay sensitive/insensitive traffic, which is modelled by adding explicit delay terms to the utility function measuring QoS. Einhorn *et al.* [5] proposed a RL-based decentralized approach for QoS routing and TE in MPLS networks. Recently, TE has been studied in the context of SDN. For example, Jain *et al.* [13] presented design and implementation of Google's SDN-based WAN, B4, and proposed a TE algorithm based on a bandwidth function for data transmissions among its data centers. The authors of [2] proposed approximation algorithms for TE problems with with partial deployment of SDN. NUM, TE and/or related problems have also been studied by quite a few works [4], [27], [34], [39] in the context of wireless networks, which were mainly focused on wireless-specific issues such as interference, time-varying link states, etc. We summarize the differences from these works as follows: 1) Unlike [25], [26], [37], [15] guided by queueing models, we develop an expereience/data-driven model-free approach based on DRL. 2) Related works [2], [13], [17] have not explicitly addressed end-to-end delay, which, however, is one of the major concerns of this paper. 3) This paper considers a TE problem in general networks, which is mathematically different from those problems in specific networks/scenarios [2], [4], [5], [13], [27], [34], [39]. 4) We are the first to leverage the emerging DRL for TE, which has been shown to be very effective.

Deep Reinforcement Learning (DRL): DRL has recently attracted extensive attention from both industry and academia. In a pioneering work [20], Mnih *et al.* proposed DQN, which can learn successful policies directly from high dimensional sensory inputs. Particularly, they introduced two new techniques, experience replay and target network, to improve learning stability. The authors of [12] proposed Double Q-learning as a specific adaptation to the DQN. The authors of [30] proposed to use prioritized experience replay in DQN, so as to replay important transitions more frequently, and therefore learn more efficiently. In [35], Wang *et al.* presented a new dueling neural network architecture, which includes two separate estimators: one for the state value function and one for the state-dependent action advantage function. The above works were focused on discrete control with a limited action space. Research efforts have also been made to extend DRL to address continuous control. Lillicrap *et al.* [16] proposed

an actor-critic-based and model-free algorithm, DDPG, based on the deterministic policy gradient that can operate over continuous action spaces. Gu *et al.* [8] proposed normalized advantage functions for reducing sample complexity. The authors of [21] proposed asynchronous gradient descent for optimizing learning with DNNs, and showed the successes of asynchronous the actor-critic method on a wide variety of continuous motor control tasks. In [9], the authors proposed a policy gradient method Q-Prop, which uses a Taylor expansion of the off-policy critic as a control variant. We aim to answer the questions if and how the emerging DRL can be applied to solving complicated control and resource allocation problems, such as TE, in communication networks. Our work represents the first effort along this line. Moreover, we introduce new techniques on exploration and experience replay to optimize the general DRL framework particularly for TE.

VII. Conclusions

In this paper, we proposed to use a novel experience-driven approach for resource allocation in communication networks, which can learn to well control a communication network from its experience rather than an accurate mathematical model. Specifically, we presented a novel and highly effective DRL-based control framework, DRL-TE, to solve the TE problem. The proposed framework enables experience-driven control by jointly learning network dynamics, and make decisions under the guidance of two DNNs, actor and critic networks. Moreover, we proposed two new techniques, TE-aware exploration and actor-critic-based prioritized experience replay, to optimize the general DRL framework particularly for TE. We implemented DRL-TE in ns-3, and conducted a comprehensive simulation study to evaluate its performance on two well-known network topologies, NSFNET and APRANET, and a random topology. Extensive simulation results have shown that 1) compared to several widely-used baseline methods, DRL-TE significantly reduces end-to-end delay and consistently improves the total utility, while offering better or comparable throughput; 2) DRL-TE is robust to network changes; and 3) DRL-TE consistently outperforms DDPG, which, however, does not offer satisfying performance.

References

[1] ARPANET, *https://en.wikipedia.org/wiki/ARPANET*

[2] S. Agarwal, M. Kodialam and TV. Lakshman, Traffic engineering in software defined networks, *IEEE INFOCOM'2013*, pp. 2211–2219.

[3] G. D. Arnold, *et al.* , Deep reinforcement learning in large discrete action spaces, *arXiv: 1512.07679*, 2016.

[4] S. Bai, W. Zhang, G. Xue, J. Tang and C. Wang, DEAR: Delay-bounded energy-constrained adaptive routing in wireless sensor networks, *IEEE INFOCOM'2012*, pp. 1593–1601.

[5] E. Einhorn and A. Mitschele-Thiel, RLTE: reinforcement learning for traffic-engineering, *IFIP AIMS'2008*, pp. 120–133.

[6] J. N. Foerster, Y. M. Assael, N. d. Freitas and S. Whiteson, Learning to communicate with deep multi-agent reinforcement learning, *NIPS'2016*, pp. 2137–2145.

[7] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016, *http://www.deeplearningbook.org*.

[8] S. Gu, T. Lillicrap, I. Sutskever and S. Levine, Continuous deep Q-Learning with model-based acceleration, *ICML'2016*, pp. 2829–2838.

[9] S. Gu, T. Lillicrap, Z. Ghahramani, R. Turner and S. Levine, Q-prop: Sample-efficient policy gradient with an off-policy critic, *arXiv: 1611.02247*, 2016.

[10] Gurobi Optimizer, *http://www.gurobi.com/*

[11] F. Gustafsson, Determining the initial states in forward-backward filtering, *IEEE Transactions on Signal Processing*, Vol. 44, No. 4, 1996, pp. 988–992.

[12] H. v. Hasselt, A. Guez, and D. Silver, Deep reinforcement learning with double Q-learning, *AAAI'2016*, pp. 2094–2100.

[13] S. Jain, *et al.* , B4: Experience with a globally-deployed software defined WAN, *ACM SIGCOMM'2013*, pp. 3–14.

[14] V. Konda and J. Tsitsiklis, Actor-critic algorithms, *NIPS'2000*. pp. 1008–1014.

[15] Y. Li, A. Papachristodoulou, M. Chiang and A R. Calderbank, Congestion control and its stability in networks with delay sensitive traffic, *Computer Networks*, Vol. 55, No. 1, 2011, pp. 20–32.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, Continuous control with deep reinforcement learning, *ICLR'2016*.

[17] S. H Low and D. E. Lapsley, Optimization flow control. I. Basic algorithm and convergence, *IEEE/ACM Transactions on networking*, Vol. 518, No. 6, 1999, pp. 861–874.

[18] N. McKeown, *et al.* , OpenFlow: enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 2, 2008 pp. 69–74.

[19] A. Medina, *et al.* , BRITE: An approach to universal topology generation, *IEEE MASCOTS'2011*, pp 346–353.

[20] V. Mnih, *et al.* , Human-level control through deep reinforcement learning, *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.

[21] V. Mnih, *et al.* , Asynchronous methods for deep reinforcement learning, *ICML'2016*, pp. 1928–1937.

[22] ns-3, *https://www.nsnam.org/*

[23] NSFNET, *https://en.wikipedia.org/wiki/National_Science_Foundation_Network*

[24] Open Shortest Path First (OSPF), *https://en.wikipedia.org/wiki/Open_Shortest_Path_First*

[25] F. Paganini, Z. Wang, J. C Doyle and S. H Low, Congestion control for high performance, stability, and fairness in general networks, *IEEE/ACM Transactions on Networking (ToN)*, Vol. 13, No. 1, 2005, pp. 43–56.

[26] D. P. Palomar and M. Chiang, Member, A tutorial on decomposition methods for network utility maximization, *IEEE Journal on Selected Areas in Communications*, Vol. 24, No. 8, 2006, pp. 1439–1451.

[27] L. Rao, X. Liu, K. Kang, W. Liu, L. Liu and Y. Chen, Optimal joint multi-path routing and sampling rates assignment for real-time wireless sensor networks, *IEEE ICC'2011*, pp. 1–5.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, Deterministic policy gradient algorithms, *ICML'2014*, pp. 387–395.

[29] D. Silver, *et al.* , Mastering the game of Go with deep neural networks and tree search *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489.

[30] T. Schaul, J. Quan, I. Antonoglou and D. Silver, Prioritized experience replay, *arXiv: 1511.05952*, 2015.

[31] R. Srikant, The mathematics of Internet congestion control, *Springer Science & Business Media*, 2012.

[32] R. Sutton and A. Barto, Reinforcement learning: an introduction, *MIT press Cambridge*, 1998.

[33] TensorFlow, *https://www.tensorflow.org/*

[34] P. Thulasiraman, J. Chen and X. Shen, Multipath routing and max-min fair QoS provisioning under interference constraints in wireless multihop networks, *IEEE Transactions on Parallel and Distributed systems*, Vol. 22, No. 5, 2011, pp. 716–728.

[35] Z. Wang, T. Schaul, M. Hessel, H. Van, M. Lanctot and N. De Freitas, Dueling network architectures for deep reinforcement learning, *ICML'2016*, pp. 1995–2003.

[36] K. Winstein and H. Balakrishnan, Tcp ex machina: Computer-generated congestion control, *ACM SIGCOMM'2013*, pp. 123–134.

[37] D. Xu, M. Chiang and J. Rexford, Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering, *IEEE/ACM Transactions on networking*, Vol. 19, No. 6, 2011, pp. 1717–1730.

[38] R. Zhang-Shen, Valiant Load-Balancing: building networks that can support all traffic matrices, *Algorithms for Next Generation Networks*, 2010.

[39] P. Zhou, L. Cheng and D. O. Wu, Shortest path routing in unknown environments: is the adaptive optimal strategy available? *IEEE SECON'2016*.