Delay-Optimal Traffic Engineering through Multi-agent Reinforcement Learning

Pinyarash Pinyoanuntapong, Minwoo Lee, Pu Wang
Department of Computer Science
University of North Carolina Charlotte
Charlotte, USA
{ppinyoan, minwoo.lee, pu.wang}@uncc.edu

***Abstract*— Traffic engineering is one of the most important methods of optimizing network performance by designing optimal forwarding and routing rules to meet the quality of service (QoS) requirements for a large volume of traffic flows. End-to-end (E2E) delay is one of the key TE metrics. Optimizing E2E delay, however, is very challenging in large-scale multi-hop networks due to the profound network uncertainties and dynamics. This paper proposes a model-free TE framework that adopts multi-agent reinforcement learning for distributed control to minimize the E2E delay. In particular, distributed TE is formulated as a multi-agent extension of Markov decision process (MA-MDP). To solve this problem, a modular and composable learning framework is proposed, which consists of three interleaving modules including policy evaluation, policy improvement, and policy execution. Each of component can be implemented using different algorithms along with their extensions. Simulation results show that the combination of several extensions, such as double learning, expected policy evaluation, and on-policy learning, can provide superior E2E delay performance under high traffic load cases.**

I. Introduction

Over the last few years data traffic has drastically increased due to the changes in the way today's society creates, shares, and consumes information. Thus, the wired and wireless network expansions are necessary to accommodate demand growth and require the efficient and intelligent utilization of limited network resources to optimize network performance. Traffic engineering (TE) [1] is one of the most important methods of optimizing network performance by dynamically measuring and analyzing real-time network traffic, and designing optimal forwarding and routing rules to meet the quality of service (QoS) requirements for a large volume of traffic flows. End-to-end (E2E) delay is one of the key QoS metrics TE aims to optimize. Optimizing E2E delay, however, is very challenging in large-scale wired networks and wireless multi-hop networks due to the profound uncertainties and dynamics in traffic flow patterns, working conditions of routers, network topology, and wireless link status.

The commercially-available TE solutions, such as OSPF, IEEE 802.11s [2], and B.A.T.M.A.N., [3], are generally variants of shortest path routing protocol. They are simple and easy to implement. But, they cannot guarantee optimal E2E TE performance. To provide provably optimal TE performance, there exists a large body of theoretical research work on stochastic network utility maximization (NUM) [4], [5], where multi-hop TE problem can be formulated as constrained maximization problems of the utility function under stochastic dynamics in user traffic and time-varying wireless channels. However, they cannot be used to minimize E2E delay because E2E delay cannot be explicitly and mathematically related to the TE control parameters, such as traffic splitting ratio over each output link, which, however, have to be included in the utility function in the NUM formulation.

Recent advances in reinforcement learning (RL) have provided diverse solutions to complex problems such as robotics [6], [7], cloud computing [8], [9], advertisement [10], and finance [11]. Adoption of reinforcement learning has been enabling experience-based model-free TE [12], [13], [14], [15], which has several key advantages: (1) it needs neither strong assumptions nor accurate modeling of the network, thus allowing it to *achieve robust and resilient performance* in complex networking systems with high-level uncertainties and randomness, (2) it is designed to handle non-stationarity, and thus *it is able to automatically adapt* to the time-varying network dynamics, (3) it can deal with large and sophisticated state/action spaces when it is combined with the recent advances in linear and non-linear function approximation (i.e. deep learning).

However, these TE solutions are mainly designed for small-scale computer networks, where a centralized network controller acts as a single agent to solve the TE problems using deep reinforcement learning. The centralized TE is not applicable in large-scale multi-hop networks. Moreover, the computational complexity of the centralized control grows exponentially as the scale of wired/wireless networks grows. Thus, it is reasonable move to distributed TE to counter the fast-changing dynamics of networks and real-time adaptation to develop local TE policies when real-time traffic measurements are only available locally.

So far, the research on distributed model-free TE mainly focuses on applying Q-learning and its variants in a multi-agent setting [16], [17], [18], [19], [20]. However, Q-learning based TE has fundamental limitations. Q-learning is an off-policy learning method, which is of greater variance and

This work is supported by NSF 1763182

converges slowly. This leads to a suboptimal TE performance when there is not sufficient experience to learn from. Q-learning is also an action-value based method, which can only learn deterministic TE policy. This means that each traffic flow can only take a single routing path to reach the destination. There exist a variety of RL algorithms and their extensions that can address the limitations of Q-learning in theory and in the general sense. However, it is still unclear (1) How these algorithms can be generalized to a multi-agent setting to enable distributed TE? (2) What is the actual performance of these algorithms when applied for TE problems? (3) Which of the algorithm extensions are complementary and can be combined to yield substantial performance improvements?

To answer three questions above, the overall objective of this paper is to develop a modular and composable multi-agent learning system, which provides modules and module extensions that can be selected and assembled in various combinations to generate a specific multi-agent reinforcement learning algorithm that can automate the E2E TE in multi-hop computer networks. Towards this goal, (1) we formulate distributed TE as a multi-agent extension of Markov decision process (MA-MDP); (2) to solve this MA-MDP problem, we propose a modular and composable learning framework consisting of three interleaving modules, each of which can be implemented using different algorithms along with different algorithm extensions. These implementations can be selected and assembled in various combinations to generate a specific reinforcement learning algorithm; (3) we propose a distributed multi-agent actor-critic-executor (MA-ACE) architecture to simplify the interleaving operations between framework modules, thus facilitating fast learning algorithm prototyping and instantiation; (4) we present preliminary results through simulations in a discrete-time network environment.

II. Multiagent Markov Decision Processes (MA-MDP) for Traffic Engineering

We formulate the traffic engineering problem as a multi-agent extension of Markov decision process [21], [22], [23], where a set $\mathscr{N} = (1,...,N)$ of agents (i.e., routers) interacts in an environment with an objective to learn the award-maximizing behavior. An MA-MDP learns which next-hop each router should send its packets to in order to move the packets to their destinations with the optimal end-to-end traffic engineering (E2E-TE) performance metrics including E2E delay, E2E throughput, and hybrid E2E TE metric that jointly considers delay and throughput. An MA-MDP for N routers is defined by a tuple $< \mathscr{S}, \mathscr{O}_1,...,\mathscr{O}_N, \mathscr{A}_1,...,\mathscr{A}_N, \mathscr{P}, r_1,...r_N >$, where

- $\mathscr{S}$ is a set of environment states, which include the network topology, the source and destination (i.e., source and destination IP addresses) of each packet in each router, the number of packets (queue size) of each router, and the status of links of each router, e.g., signal-to-interference-plus-noise ratio (SINR).
- $\mathscr{O}_i, i = 1,...,N$ is a set of observations for each router i, which include local network states available at router i. For example, the observation can be simply the destination of the incoming packet.
- $\mathscr{A}_i, i = 1,...,N$ is a set of actions for each router i, which include the next-hop routers the current router can forward the packets to.
- $\mathscr{P} : \mathscr{S} \times \mathscr{A}_1 \times ... \times \mathscr{A}_N \times \mathscr{S} \mapsto [0,1]$ is the state transition probability function, which models the environment dynamics. The environment dynamics are driven by the unknown stochastic packet arrival processes of traffic sources, and the packet departure processes that are determined by the action and link status of each router.
- $r_i, i = 1,...,N : \mathscr{A}_i \times \mathscr{S} \mapsto \mathbb{R}$ is the reward function of each router i, which is defined based on the E2E-TE metrics we want to optimize. In this paper, we aim to optimize E2E delay. Therefore, the reward $r_i = d_i$ is defined as the (negative-signed) 1-hop delay d_i a packet experiences when it is forwarded from current router i to next-hop router $i+1$. d_i includes processing delay, queueing delay, transmission delay, and propagation delay.

When a packet enters a router i, the router obtains its local observation $o \in \mathscr{O}_i$ of the network states and takes an action $a \in \mathscr{A}_i$ to determine where to send this packet to. As a result, the router receives a reward r_i (e.g., (negative) one-hop delay) when the packet arrives at its next-hop router $i+1$, which has its own local observation $o' \in \mathscr{O}_{i+1}$. Each router selects actions based on a policy π_i, which specifies how the router chooses its action given the observation. The policy can be stochastic $\pi_i(a|o) : \mathscr{O}_i \times \mathscr{A}_i \mapsto [0,1]$, where given current observation $o \in \mathscr{O}_i$, the router sends a packet to the next-hop router $a \in \mathscr{A}_i$ according to the probability $\pi_i(a|o)$ with $\sum_{a \in \mathscr{A}_i} \pi_i(a|o) = 1$. The policy can be also deterministic $\pi_i(o) : \mathscr{O}_i \mapsto \mathscr{A}_i$, where given current observation $o \in \mathscr{O}_i$, the router sends a packet to a fixed next-hop router $a \in \mathscr{A}_i$. The return $G_i = \sum_{k=i}^{T} r_k$ is the total reward from intermediate state s_i to final state s_T, where s_i and s_T are the states when a packet arrives at the intermediate router i and destination router T, respectively. Let s_1 be the initial state when a packet enters the network from its source router. The goal is to find the optimal policy π_i for each router i so that the expected return $J(\boldsymbol{\pi})$ from the initial state (E2E TE metric) is maximized,

$$J(\boldsymbol{\pi}) = E[G_i|\boldsymbol{\pi}] = E[\sum_{i=1}^{T} r_i|\boldsymbol{\pi}] \quad (1)$$

where $\boldsymbol{\pi} = \pi_1,...,\pi_N$. Using different reward function (e.g., 1-hop delay d_i), $J(\boldsymbol{\pi})$ can characterize different individual E2E-TE metric (e.g., expected E2E delay $E[G_i^{(d)}|\boldsymbol{\pi}] = E[\sum_{i=1}^{T} d_i|\boldsymbol{\pi}]$).

In TE problems, the states are fully observable. That is, the state is uniquely defined by the observations of all routers (i.e., $P(o|s,a) > 0 \implies P(s'|o) = 1$). Thus, in the following sections, for simplicity of notation, we represent an observation o as a state s. In the scope of this paper, we examine only the E2E delay as a reward for delay-optimal traffic

engineering. Moreover, we assume that the environmental dynamics, which are characterized by the state transition probability and the distribution of the rewards, are unknown for practical real-world applications. This leads us to propose the following model-free multi-agent reinforcement learning framework.

III. Distributed TE Learning Framework

The proposed modular framework architecture includes a generic composing procedure, which assembles a variety of different algorithm and extension options to enable fast prototyping and evaluation. It consists of three key modules, each of which can be implemented using different algorithms along with using different algorithm extensions. These implementations can be selected and assembled in various combinations to generate specific reinforcement learning algorithms. This framework will be developed based on the generalized policy iteration (GPI) strategy [24]. GPI was initially developed to generalize single-agent value-based reinforcement learning algorithms. We will extend it for solving generic TE problems, which needs to exploit emerging policy gradient based learning along with function approximation in a multi-agent setting.

In particular, our framework consists of three interleaving modules for each router/agent: policy evaluation, policy improvement, and policy execution. Let us consider a particular router i and its policy π_i to be learned. Policy evaluation estimates the action-value functions,

$$q_i^{\pi_i}(s,a) = E^{\pi_i}\left[G_i = \sum_{k=i}^{T} r_k | s_i = s, a_i = a\right] \quad (2)$$

that measure the expected return (expected E2E TE metric) if the router i performs a given action in a given state. Next, policy improvement utilizes the estimated action-values $q_i^{\pi_i}(s,a)$ to adjust current policy π_i in the direction of greater expected return. After that, the agent executes a behavior policy b_i to generate new action-reward experiences for next-round policy evaluation and improvement. As illustrated in Fig. 1, we adopt a distributed actor-critic-executor architecture similar to asynchronous advantage actor-critic (A3C) [25] to simplify and implement the interleaving operations between policy evaluation, improvement, and execution. Each router has its own actor, critic and executor running locally and in parallel. The local critic uses a variety of methods to estimate the action-value functions $Q_i^{\pi_i}(s,a)$, which criticize the action selections. Based on critic's inputs, the actor improves the target policy that we want to learn and optimize. Then, the executor executes the actions according to the behavior policy, which is either equal to the target policy or similar to the target policy but more exploratory.

A. Local Critic for Policy Evaluation

1) 1-hop Action-value Estimation: As shown in eq. (2), the performance of the policy π is measured by the action-value $q_i^{\pi}(s,a)$, which is a E2E TE metric. Thus, there will be no direct training sample for policy evaluation until a packet forwarded by this router arrives at its destination. Inspired

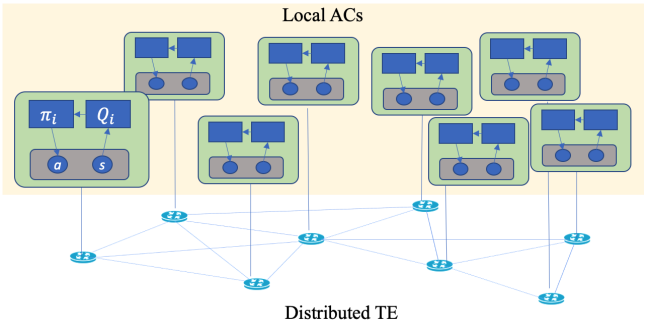


Fig. 1. Distributed TE Framework which adopts multi-agent, asynchronous actor-critic (AC) architecture. Each router's actor updates its policy function while critic updates the function approximation of state-action Q values.

by temporal-difference prediction [24], we can apply *spatial-difference (SD) prediction* to quickly update the estimation of $q_i^{\pi}(s,a)$ only using local information exchanged between adjacent routers. In particular, the action-value $q_i^{\pi}(s,a)$ of router i can be recursively rewritten as the sum of 1-hop reward of router i and the action-value of the next-hop router $i+1$, i.e.,

$$q_i^{\pi_i}(s,a) = E\left[r_i + q_{i+1}^{\pi_{i+1}}(s',a')\right]. \quad (3)$$

This equation (3) indicates $q_i^{\pi_i}(s,a)$ can be estimated by averaging the samples of $r_i + q_{i+1}^{\pi_{i+1}}(s',a')$. This leads to a simple SD predication method based on *exponential weighted average (EWA)*, which iteratively updates the estimate of $q_i^{\pi_i}(s,a)$, denoted by $Q_i^{\pi_i}(s,a)$, based on 1-hop experience tuples (s,a,r_i,s',a') and the estimate of $q_{i+1}^{\pi_{i+1}}(s',a')$ of next-hop router, denoted by $Q_{i+1}^{\pi_{i+1}}(s',a')$, i.e.,

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i + Q_{i+1}^{\pi_{i+1}}(s',a') - Q_i^{\pi_i}(s,a)] \quad (4)$$

where $\alpha \in (0,1]$ is the learning rate.

2) n-hop Action-value Estimation: 1-hop action-value estimation can be generalized to n-hop one by using n-hop return r_i^n instead of 1-hop return r_i to update the action-value estimate. The n-hop return r_i^n is the accumulated reward when a packet arrives at the n-hop router, i.e., $r_i^n = \sum_{k=i}^{i+n-1} r_k$. For example, if the reward is 1-hop delay, r_i^n represents n-hop delay. With n-hop return, the action-value estimation process in eq. (4) can be generalized to

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i^n + Q_{i+n}^{\pi_{i+n}}(s',a') - Q_i^{\pi_i}(s,a)]$$

for all $n \geq 1$ where $Q_{i+n}^{\pi_{i+n}}(s',a')$ is the action-value of router $i+n$. It is shown in previous research [26], [27], [28] (e.g., in video gaming settings) that n-hop/n-step estimation may lead to better policy with higher expected return (e.g., better E2E TE metric in our case) because the n-hop estimate has smaller bias compared with the 1-hop one. However, n-hop estimate may not work well in non-stationary cases and may slow down policy learning process because of the higher variance in the n-step estimation and the longer waiting time to obtain n-hop return from the router n-hop away.

3) Expected n-hop Action-value Estimation: Variance in the action-value estimates is unavoidable because the environment can introduce stochasticity through stochastic rewards r_i^n and stochastic environment state transitions $\mathscr{P}$. There is little we can do to reduce the variance caused by environmental stochasticity, except using a suitably small learning rate α. Besides environmental stochasticity, the action change of next-hop or nth-hop router introduces additional variance. To mitigate such variance, instead of $Q_{i+n}^{\pi_{i+n}}(s',a')$, the expected value $E[Q_{i+n}^{\pi_{i+n}}(s',a')] = \sum_{a' \in A_{i+n}} \pi_{i+n}(s',a') Q_{i+n}^{\pi_{i+n}}(s',a')$ is used to update $Q_i^{\pi_i}(s,a)$, $\forall n \geq 1$ [29]:

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i^n + E[Q_{i+n}^{\pi_{i+n}}(s',a')] - Q_i^{\pi_i}(s,a)].$$

B. Local Actor for Policy Improvement

Policy evaluation process drives the action-value function to accurately predict the true returns (E2E TE metrics) for current policy. Policy improvement process improves the policy with respect to current action-value function. Policy improvement can be done through action-value methods or policy-gradient methods. In this paper, we focus on action-value methods.

1) Action-value methods: Action-value control algorithms aim to learn an deterministic target policy, which maximizes the performance objective $J(\boldsymbol{\pi})$ in eq. (1), i.e., the expected return from start state, by selecting a fixed greedy action with respect to the expected return from any state. This can be done by letting each router i greedily improve its current policy π_i, i.e., select the action with the maximum estimated action-value,

$$\pi_i(s) \leftarrow \arg\max_a Q_i^{\pi_i}(s,a).$$

Since value based methods can only learn deterministic policies, this naturally leads to single-path TE solutions, where a single routing path is learned between a source source-destination pair. The single-path TE solutions are simple and easy to implement. However, to improve E2E delay at high traffic load cases, multi-path TE solutions are generally preferred, where each source-destination pair is connected with multiple routing paths to better distribute traffic load and reduce E2E delay. To address this problem, two generic near-greedy action selections can be exploited: (1) $\varepsilon-$greedy policy, where with probability $1-\varepsilon$, select the best action and with probability ε, other actions are selected, and (2) softmax-greedy policy, where each action a is selected with a probability $P(a)$ according to the exponential Boltzmann distribution

$$P(a) = \frac{\exp(Q_i^{\pi_i}(s,a)/\tau)}{\sum_{b \in \mathscr{A}_i} \exp(Q_i^{\pi_i}(s,b)/\tau)}$$

where τ is a positive parameter called the temperature. High temperatures (as $\tau \to \infty$) cause the actions to be almost equally probable (close to random action selection). Low temperatures (as $\tau \to 0$) get close to the deterministic action selection. The near-greedy methods force each router to select next-hop forwarding nodes stochastically, which create multiple routing paths connecting source and destination nodes.

2) Double learning: All action-value algorithms above involve maximization during the construction procedure for the target policies. More specifically, the target policies follow the greedy or near-greedy action selection methods given the current action values, which are defined with a maximization operation. In this case, the maximum of the estimated action values is used as an estimate of the maximum of the true action value. This can lead to a significant overestimation bias and thus degrade the gain of the learning algorithms. To address this overestimation bias, double learning can be adopted [30], which decouples action selection from its evaluation using a pair of estimators. In particular, we divide the time steps (or equivalently the experiences) into two sets and use them to learn two independent estimates, namely $Q_{i,1}^{\pi_i}(s,a)$ and $Q_{i,2}^{\pi_i}(s,a)$. One estimate, e.g., $Q_{i,1}^{\pi_i}(s,a)$, can be used to determine the improved action denoted by $a^*(Q_{i,1}^{\pi_i})$, according to greedy or near-greedy strategies $\pi_i \in \{\text{greedy},\ \text{softmax},\ \varepsilon-\text{greedy}\}$. Then, we use the other estimate $Q_{i,2}^{\pi_i}(s,a)$ to provide the estimation of the value of the improved action, i.e., $Q_{i,2}^{\pi_i}(s,a^*) = Q_{i,2}^{\pi_i}(s,a^*(Q_{i,1}^{\pi_i}))$, Then, the role of the two estimates can be reversed to yield a second unbiased estimate $Q_{i,1}^{\pi_i}(s,a^*(Q_{i,2}^{\pi_i}))$.

Double learning strategy can be combined with any policy evaluation and policy improvement methods to create new learning algorithms. For example, we can design the double expected softmax learning by combining double learning with expected action-value estimation and softmax policy improvement. In this case, the action value update rule for the first estimate $Q_{i,1}^{\pi_i}(s,a)$ is given as follows

$$Q_{i,1}^{\pi_i}(s,a) \leftarrow Q_{i,1}^{\pi_i}(s,a) + \alpha[r_i + \Psi_{i+1,2}(s',a') - Q_{i,1}^{\pi_i}(s,a)].$$

Here, $\Psi_{i+1,2}(s',a') = \sum_{a \in \mathscr{A}} P(a,1) Q_{i+1,2}^{\pi_{i+1}}(s',a')$ where $P(a,1) = \frac{\exp(Q_{i,1}^{\pi_i}(s,a)/\tau)}{\sum_{b \in \mathscr{A}_i} \exp(Q_{i,1}^{\pi_i}(s,b)/\tau)}$. By the same way, we can obtain the update rule for the second estimate $Q_{i,2}^{\pi_i}(s,a)$. Similarly, we can also combine double learning with greedy policy improvement, where $\Psi_{i+1,2}(s',a') = Q_{i+1,2}^{\pi_{i+1}}(s', \arg\max_a Q_{i+1,1}^{\pi_i}(s',a))$.

C. Local Executor for Policy Execution

In reinforcement learning, the behavior policy b_i is the policy that generates the actual actions of the learning agent, which yields the actual experiences for improving target policy π_i. For on-policy learning, the behavior policy b_i is same as the policy being improved (target policy π_i). For off-policy learning, the behavior policy b_i is different from the target policy, where the target policy is generally the greedy policy and the behavior policy is near-greedy to encourage explorations. The policy π_i will be learned when the evaluation process and the improvement process stabilize, that is, no longer produce changes.

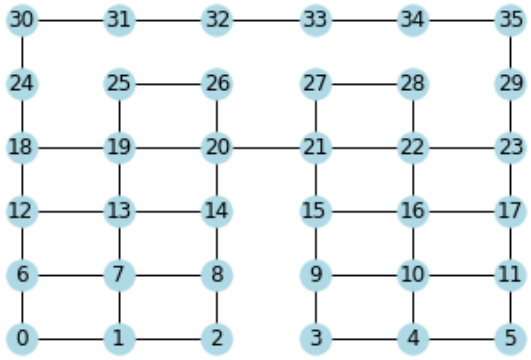


Fig. 2. Network topology in the simulations [16]

TABLE I

HYPERPARAMETERS

Parameter	value
Initial Q-values	0
Learning rate (α)	0.9
ε-greedy (ε)	0.1
Softmax Temperature (τ)	1
Multi-hop returns n	2

IV. EXPERIMENTS

A. Experiment Setup

We evaluate the performance of the learning-based TE algorithms in a discrete-time network simulator [16], which is widely used to investigate the performance of Q-learning based routing algorithm and its variants [16], [17], [18], [20], [19]. The network topology is shown in Fig. 2, where the nodes represent routers and edges represent network links. In this discrete-time simulation, the whole network is driven by three major factors: the packets arrival pattern, the average packet arrival rate, and the queuing and link delay. First, packets are periodically generated with a randomly selected source router and destination router for each packet. Second, the network loads are based on the packet arrival rate and it is driven by Poisson distribution with parameter λ, the average number of packets injected into network per time step. Third, when a packet arrives at a router, it has to wait in a FIFO queue (queue length = 1000 packets) and thus experiences queuing delay. The packet will be transmitted over the communication link if it becomes the head-of-line packet. The transmission delay of the communication link needs to be constant in this simulator and accordingly, we use the unit transmission delay for the simulations (i. e. , the link delay is set to 1.0).

Local states at each router include the source and destination IPs of the incoming packet and local queue length, i.e., $s = (srcIP, dstIP, queue)$. Action is the next-hop forwarding node, i.e., *nexthop ID*. Critic sets 1-hop value estimation ($n = 1$). The 2-hop value estimation ($n = 2$) is included to examine the comparative efficacy of n-hop learning. In this experiment, we investigated deterministic and near-deterministic policies with ε-greedy ($\varepsilon = 0.1$) and softmax ($\tau = 1$).

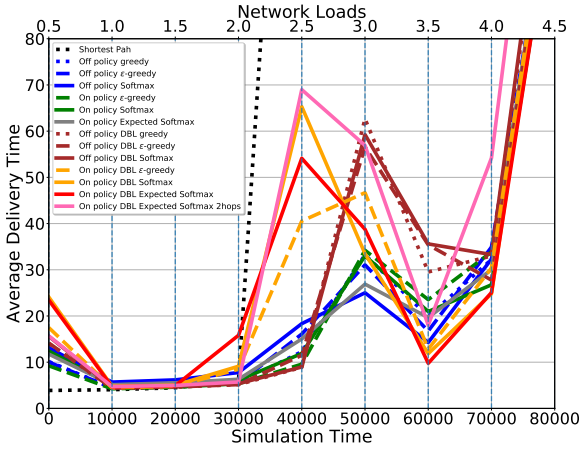


Fig. 3. Average of 50 runs of E2E delay patterns on low to high network loads

TABLE II

AVERAGE E2E OF EACH NETWORK LOAD IN NON-STATIONARY CASE

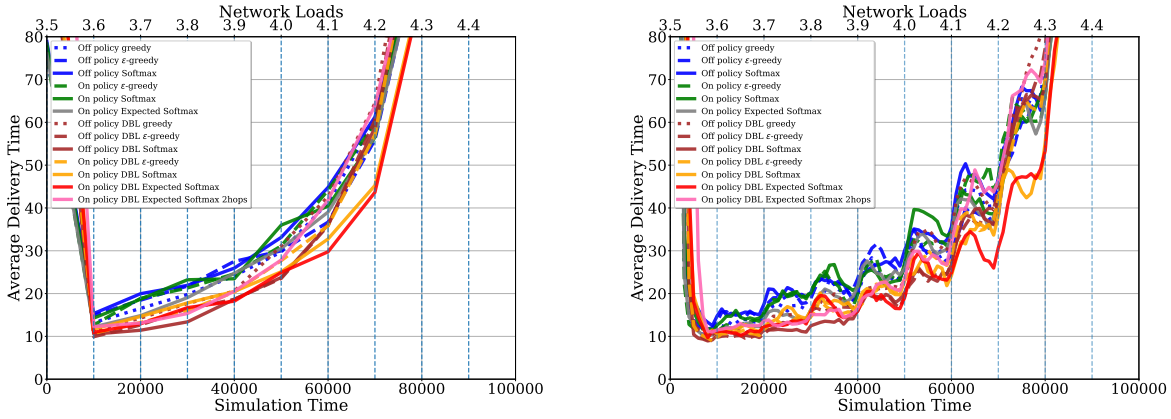
Methods	Network Loads (λ) 0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
Shortest Pah (baseline)	**3.88**	**4.05**	**4.46**	6.15	215.01	437.8	441.85	441.36	441.95
Off policy greedy (baseline)	10.14	4.43	4.78	5.51	12.27	33.25	19.87	32.24	114.89
Off policy ε-greedy	10.02	4.44	4.8	5.45	16.09	31.06	16.74	34.91	117.9
Off policy Softmax	13.14	5.69	6.17	7.76	18.34	**25.02**	14.26	32.34	111.35
On policy ε-greedy	9.22	4.24	4.59	5.34	9.57	34.22	23.54	33.85	113.43
On policy Softmax	12.43	5.06	5.38	6.08	11.89	33.02	20.96	26.74	112.85
On policy Expected Softmax	11.74	5.12	5.55	6.3	15.03	**26.89**	19.66	30.05	121.32
Off policy DBL greedy	14.05	4.39	4.64	**5.16**	**8.88**	62.68	29.53	33.0	131.63
Off policy DBL ε-greedy	14.15	4.39	4.64	**5.14**	11.5	56.57	35.39	27.77	121.08
Off policy DBL Softmax	15.6	4.56	4.8	5.35	**8.97**	59.27	35.6	33.29	133.92
On policy DBL ε-greedy	17.48	4.46	4.79	8.47	40.55	46.68	12.82	30.38	119.9
On policy DBL Softmax	24.09	4.8	5.02	9.12	65.31	33.47	11.86	**24.79**	117.08
On policy DBL Expected Softmax	23.44	4.66	4.91	15.82	54.12	38.77	**9.76**	**25.07**	113.34
On policy DBL Expected Softmax 2hops	15.67	4.72	4.97	5.72	68.99	56.91	18.01	54.2	162.77

Off-policy, on-policy, expected value estimation (Expected), double learning (DBL), and n-hop learning (n = 2) are employed into the framework to examine the efficacy of learning-based adaptation. Two baseline algorithms are also included: the shortest-path routing and the off-policy greedy algorithm (i.e., Q-routing [16]). In particular, for Q-routing, both behavior and target policies are greedy and the explorations are implicitly driven by the changes in action value (Q-value) estimations. The hyperparameters of learning algorithms are summarized in table I.

B. Varying Traffic Loads: Low to High

Unlike previous research that has tested algorithms in single traffic load case, we consider the low to high load condition to study the overall performance of all adaptive learning algorithms in non-stationary network environment, where the state transition probability function keeps changing as traffic load increases. In this experiment, the following changes to the network parameters were made according to time. We initialized Q-table values with 0. The first 10k time steps were given for the initial exploration at a low load λ = 0.5. λ is increased by 0.5 for every 10k time steps. From pilot tests, we select the best performing learning rate $\alpha = 0.9$ for all algorithms except 2-hop learning algorithm, whose optimal learning rate is $\alpha = 0.5$.

The results in Fig. 3 and Table II show that all of reinforcement learning approaches learn efficient routing



(a) (b)

Fig. 4. Average of 50 runs for E2E delay under high network increasing load conditions. For each run, we measured the average delivery time for 10,000 time step or every network load changes in (a) and every 1,000 time steps in (b)

TABLE III

AVERAGE E2E DELAY PRESENTED IN FIG. 4

Methods	Network Loads (λ) 3.5	3.6	3.7	3.8	3.9	4.0	4.1	4.2	4.3	4.4
Off policy greedy (baseline)	**75.47**	13.21	16.54	19.78	24.14	29.72	41.25	60.67	109.97	194.45
Off policy ε-greedy	77.51	14.93	18.46	21.75	27.53	29.98	36.69	55.91	103.57	193.79
Off policy Softmax	78.88	15.33	19.97	21.92	25.95	33.13	44.8	61.25	105.06	189.23
On policy ε-greedy	**75.33**	12.66	18.89	21.3	24.62	31.13	44.14	59.76	107.58	190.17
On policy Softmax	78.29	14.27	18.68	23.23	23.46	35.99	40.41	56.51	106.54	193.22
On policy Expected Softmax	76.0	12.21	14.85	19.05	24.75	30.54	39.04	56.79	102.35	189.33
Off policy DBL greedy	115.44	11.31	14.28	17.67	20.57	31.16	42.65	64.47	127.45	211.62
Off policy DBL ε-greedy	114.92	**9.92**	12.67	16.48	18.65	**23.53**	35.85	60.62	122.09	207.46
Off policy DBL Softmax	113.82	**10.53**	**11.43**	**13.39**	18.66	**23.76**	36.11	58.76	112.81	200.47
On policy DBL ε-greedy	119.95	10.72	13.26	16.03	20.36	27.69	35.84	57.07	115.98	202.2
On policy DBL Softmax	136.85	11.68	14.65	17.81	20.49	25.09	32.76	45.25	92.22	**189.29**
On policy DBL Expected Softmax	136.71	10.95	12.94	16.71	**18.23**	24.93	**29.77**	**43.66**	**90.99**	192.62
On policy DBL Expected Softmax 2hops	200.28	12.24	13.13	15.34	20.53	27.83	41.85	63.91	109.42	201.08

policies while the shortest path routing scheme failed to tolerate the increased packet loads (when $\lambda > 2.0$). During the medium traffic loads ($2.0 \leq \lambda \leq 3.0$), off-policy learning algorithms outperform on-policy algorithms, where off-policy softmax with double learning is the best one during the load ($2.0 \leq \lambda \leq 2.5$) and off-policy softmax is the best during the load ($\lambda = 3.0$). During high traffic loads ($3.5 \leq \lambda \leq 4.5$), on-policy algorithms outperform off-policy approaches. Moreover, for on-policy algorithms, expected value evaluation helps to improve the delay performance due to the smaller variance in the return estimation. Double learning, for most of the cases except loads ($\lambda = 3.0$), is beneficial by reducing maximization bias. Softmax action-selection, in general, helps the agent to select second-best control links with a probability, and it helps exploration of other paths to balance the traffic in high network loads and to reduce average packet delivery time. 2-hop learning algorithm does not bring much performance gain.

C. Adaptivity under High Traffic Load Region

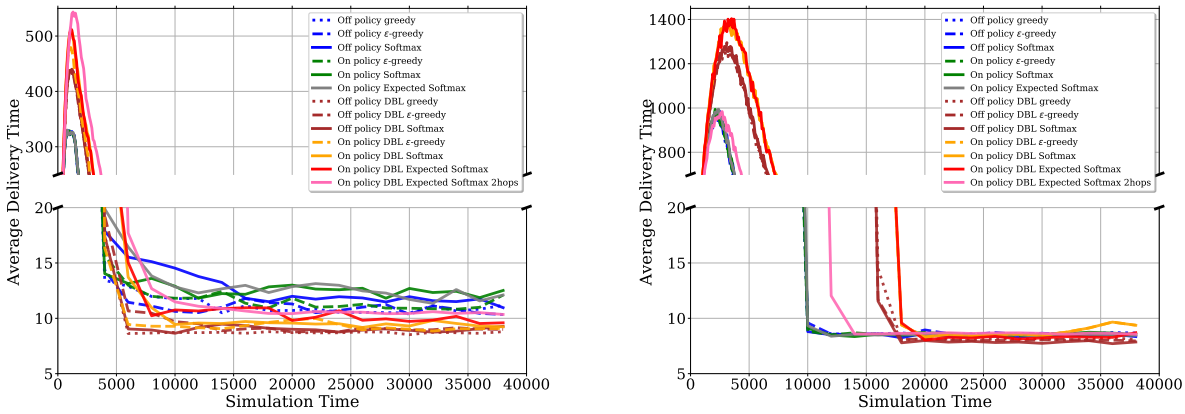
In this experiment, we further investigate the performance of the learning algorithms in high-traffic load cases. We increase the network loads from $\lambda = 3.5$ to $\lambda = 4.4$ by 0.1 on every 10k time steps. As illustrated in Fig. 4(a) and Table III, when network loads are high, double learning algorithms with softmax action selection (orange, red, and brown solid lines) shows highest adaptivity and best performance. As network loads further increases, off-policy double learning adapts poorly, comparing to on-policy double learning algorithms after $\lambda = 3.9$. These results show additional exploration, leaving from deterministic policy, helps balancing high network traffic loads. Both softmax and on-policy learning add efficient exploration for this, and avoiding overestimation with double-learning improves overall E2E delay performance. Fig. 4(b) shows the delay performance with higher time resolution, where the average packet delay is computed for every 1,000 time steps. We observe that all algorithms take some time to learn and eventually adapt to the traffic load changes well.

D. Convergence Analysis under a Stationary Case

In Fig. 5, with fixed high network loads (λ = 3.5) , we test the convergence of learning modules using different learning rates (0.1 and 0.9, respectively). The fixed traffic load case represents stationary network condition, where state-transition probability function converges as time proceeds. As shown in Fig. 5, all learning algorithms converge eventually. Table IV summarizes the convergence time and the delay performance of each algorithm. As expected, low learning rate ($\alpha = 0.1$) leads to better delay performance with less divergence for all algorithms by observing more samples for learning (thus, taking more learning time). High learning rate ($\alpha = 0.9$) shortens the convergence time at least by half to reach fairly good (through not optimal) performance. This also explains why high learning rate leads to the overall better performance in non-stationary environment with changing traffic loads. Moreover, we observe the double learning models stably converges the best results for both low and high learning rates. With high learning rate, double learning algorithms only need one third of convergence time, compared with low learning rate case.

E. Discussions

We have demonstrated that combining several techniques such as Double learning, Expected action value estimation, and Softmax on-policy into a single learning algorithm minimizes E2E delay specially in high-traffic load region.



(a) High learning rate ($\alpha = 0.9$)

(b) Low learning rate ($\alpha = 0.1$)

Fig. 5. Average of 50 runs for convergence of learning algorithms in fixed high network loads ($\lambda = 3.5$). For each run, we measured the average delivery time for every 100 time steps

TABLE IV

AVERAGE E2E DELAY PRESENTED IN FIG. 5

Methods	$\alpha = 0.9$	convergence time	$\alpha = 0.1$	convergence time
Off policy greedy (baseline)	10.89 ±0.35	4,000	8.61 ±0.06	10,000
Off policy ε-greedy	10.88 ±0.32	4,000	8.63 ±0.1	10,000
Off policy Softmax	12.11 ±1.1	4,000	8.43 ±0.07	10,000
On policy ε-greedy	11.34 ±0.47	5,000	8.53 ±0.06	10,000
On policy Softmax	12.44 ±0.32	5,000	8.57 ±0.06	10,000
On policy Expected Softmax	12.41 ±0.6	5,000	8.45 ±0.05	10,000
Off policy DBL greedy	**8.71 ±0.07**	6,000	8.12 ±0.06	17,000
Off policy DBL ε-greedy	9.06 ±0.12	6,000	8.02 ±0.04	17,000
Off policy DBL Softmax	9.02 ±0.13	6,000	**7.86 ±0.04**	17,000
On policy DBL ε-greedy	9.26 ±0.2	6,000	8.70 ±0.23	19,000
On policy DBL Softmax	9.48 ±0.12	7,000	8.78 ±0.27	19,000
On policy DBL Expected Softmax	10.23 ±0.6	7,000	8.29 ±0.08	19,000
On policy DBL Expected Softmax 2hops	10.63 ±0.19	8,000	8.64 ±0.07	19,000

2-hop action-value estimation did not bring so much benefit partially because 2-hop estimation leads to lower estimation bias at the cost of higher estimation variance and lower learning speed. The combined effect may not necessarily lead to improved performance specially under the non-stationary environment. In the non-stationary case (varying network loads), we observed that high learning rate (λ = 0.9) is the best for all algorithms since the agent needs to learn and adapt policy quickly when network load changes. High learning rate allows agents to take the most updated reward (per-hop delay) into the return (E2E) estimation, while quickly forgetting the previous outdated estimation. Double learning in overall can effectively improve delay performance specially in medium and high traffic load region due to the reduced maximization bias. Double learning does not perform well during low traffic load region because only half of the experience is used to train each Q table and low traffic load means less experience. Expected action-value estimation can lead to improved performance specially in high traffic load region because of the reduced variance in estimated action-value. In this paper, a constant high learning rate is exploited, which may not be the best choice. One possible solution is to use adaptive learning rate. We could use high learning for agent to adjust their policy when network load changes, but when it converges and stable, we can decrease the learning rate to reach the best optimal policy.

V. CONCLUSIONS

In this paper, we formulate distributed TE problems in reinforcement learning problems and suggest a composable framework for diverse learning algorithm application. Through extensive comparative experiments, we demonstrated high adaptiveness of learning-based distributed TE approaches, which lead robust load-balancing network systems that minimize the E2E delay. Empirically, we were also able to observe that reducing over-estimation bias with double-learning along with non-deterministic action selection with softwax improves adaptivity and sustainability of network systems.

REFERENCES

[1] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings of IEEE INFOCOM*. IEEE, 2013, pp. 2211–2219.

[2] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "Ieee 802.11 s: the wlan mesh standard," *IEEE Wireless Communications*, vol. 17, no. 1, 2010.

[3] The open mesh networks consortium. [Online]. Available: http://www.open-mesh.org

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

[5] S.-C. Lin, P. Wang, I. F. Akyildiz, and L. Min, "Utility-optimal wireless routing in the presence of heavy tails," *IEEE Transactions on Vehicular Technology*, 2018.

[6] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially Aware Motion Planning with Deep Reinforcement Learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1343–1350.

[7] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 5048–5058.

[8] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 372–382.

[9] D. Yang, W. Rang, and D. Cheng, "Joint optimization of mapreduce scheduling and network policy in hierarchical clouds," in *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 2018, p. 66.

[10] J. Zhao, G. Qiu, Z. Guan, W. Zhao, and X. He, "Deep Reinforcement Learning for Sponsored Search Real-time Bidding." *arXiv preprint arXiv:1803.00259*, 2018.

[11] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading." *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 653–664, 2017.

[12] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach," in *Services Computing (SCC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 25–33.

[13] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[14] Z. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrows intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[15] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. Liu, and D. Yang, "Experience-driven networking: a deep reinforcement learning based approach," in *Proc. of IEEE Infocom*, 2018.

[16] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in neural information processing systems (NIPS)*, 1994, pp. 671–678.

[17] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN'02)*, vol. 2. IEEE, 2002, pp. 1825–1830.

[18] Y. Shilova, M. Kavalerov, and I. Bezukladnikov, "Full echo q-routing with adaptive learning rates: a reinforcement learning approach to network routing," in *2016 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus),*. IEEE, 2016, pp. 341–344.

[19] M. Kavalerov, Y. Shilova, and Y. Likhacheva, "Adaptive q-routing with random echo and route memory," in *2017 20th Conference of Open Innovations Association (FRUCT)*. IEEE, 2017, pp. 138–145.

[20] M. V. Kavalerov, Y. A. Shilova, and I. I. Bezukladnikov, "Preventing instability in full echo q-routing with adaptive learning rates," in *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus),*. IEEE, 2017, pp. 155–159.

[21] P. Xuan, V. Lesser, and S. Zilberstein, "Communication decisions in multi-agent cooperation: Model and experiments," in *Proceedings of the fifth international conference on Autonomous agents*. ACM, 2001, pp. 616–623.

[22] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[23] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling, "Learning to cooperate via policy search," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2000, pp. 489–496.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[26] H. van Seijen, "Effective multi-step temporal-difference learning for non-linear function approximation," *arXiv preprint arXiv:1608.05151*, 2016.

[27] A. R. Mahmood, H. Yu, and R. S. Sutton, "Multi-step off-policy learning without importance sampling ratios," *arXiv preprint arXiv:1702.03006*, 2017.

[28] K. De Asis and R. S. Sutton, "Per-decision multi-step temporal difference learning with control variates," *Proceedings of the 2018 Conference on Uncertainty in Artificial Intelligence*, 2018.

[29] H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of expected sarsa," in *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*. IEEE, 2009, pp. 177–184.

[30] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2010, vol. 23, pp. 2613–2621.