Future Generation Computer Systems 99 (2019) 401–409

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

TIDE: Time-relevant deep reinforcement learning for routing optimization

Penghao Sun [a], Yuxiang Hu [a], Julong Lan [a], Le Tian [a], Min Chen [b,c,*]

[a] National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China
[b] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
[c] Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, 430074, China

HIGHLIGHTS

- A "collections-decision-adjustment" loop is proposed to generate an intelligent routing.
- An RNN-based deep reinforcement learning method is designed for traffic characteristics abstracting.
- The proposed architecture is implemented and validated on a real transmitting network topology.

ARTICLE INFO

Article history:
Received 27 February 2019
Received in revised form 27 March 2019
Accepted 3 April 2019
Available online 28 April 2019

Keywords:
Deep reinforcement learning
Recurrent neural network
Routing optimization
Software-defined networking

ABSTRACT

Routing optimization has been researched in network design for a long time, and plenty of optimization schemes have been proposed from both academia and industry. However, such schemes are either too complicated in applications or far from the optimal performance. In recent years, with the development of Software-defined Networking (SDN) and Artificial Intelligence (AI), AI-based methods of routing strategy are being considered. In this paper, we propose TIDE, an intelligent network control architecture based on deep reinforcement learning that can dynamically optimize routing strategies in an SDN network without human experience. TIDE is implemented and validated on a real network environment. Experiment result shows that TIDE can adjust the routing strategy dynamically according to the network condition and can improve the overall network transmitting delay by about 9% compared with traditional algorithms.

1. Introduction

In recent years, the communication network has witnessed a rapid growth of network scale and application variety. Correspondingly, the intensity, variety, and complexity of spatio-temporal distribution of traffic flow also increase largely, which have brought great challenges to the routing policy of underlying networks. Without a proper routing policy, the utilization of transmitting networks will be seriously undermined. For example, the imbalance of network traffic may lead to overload in some regions while idle in others, which causes a series problem such as low quality of service (QoS) and a waste of resources. To deal with the routing optimization problem under such pressure, a large amount of white box schemes is proposed [1,2]. However, since the traffic flows are not static, designing a dynamic routing strategy to reach a near real-time optimal performance is not easy, both for the complexity of traffic pattern analysis and the acquisition of real-time traffic distribution. Load balancing and QoS guarantee are two typical routing optimization targets. The primary goal of them is to change the routing path of traffic flows to reach a better network performance. Generally, QoS guarantee takes more parameters into consideration, thus is more complex than load balancing in modeling. In this paper, we use QoS guarantee as a representative of routing optimization application.

Traditional methods of routing optimization rely mainly on meticulously designed algorithms based on link prediction or prior experience, which are either not fine-grained or time-consuming. Apart from such traditional white box methods, Artificial Intelligence (AI) has broadened the horizon of this research field. Artificial Intelligence (AI) techniques are gaining momentum in recent years and is thought to have a greater ability in the analysis and processing of a large amount of data than traditional algorithms, thus having the potential to discover some new data patterns and make accurate decisions in more complicated environment. Therefore, many researchers in the field of network design are now paying attention to a usage of AI. In 2003, David D. Clark et al. proposed "knowledge plane" [3], which depicted a primitive view about the usage of AI in network design. Researches [4–7] also discussed the network architecture

* Corresponding author.
E-mail address: minchen2012@hust.edu.cn (M. Chen).

https://doi.org/10.1016/j.future.2019.04.014
0167-739X/

with AI application. R.W. Thomas et al. [4] proposed "cognitive network" in 2005, Hajer Derbel et al. [5] proposed an autonomic network management architecture named "ANEMA" in 2009, and Michele, et al. [6] designed an intelligent network architecture "COBANETS" in 2016. In 2017, Albert et al. [7]. published their work entitled "Knowledge-Defined Networking", which redefined knowledge plane in the network and triggered a hot discussion in the academia. In [8–10], M. Chen et al., also discussed the framework of machine learning technologies in edge network and user end. However, such schemes did not provide details to realize the automatic control on the network, and there are still many problems that stand in the way of the employment of AI in network design. For example, Agoulmine et al. [11] listed a series of problems that made the automatic management of network difficult.

The emerging of SDN (Software-defined networking) [12] also provides much help for routing optimization. In SDN, control plane and data plane are decoupled, thus the manager can place some centralized logic on the control plane while leaving the forwarding functions to the data plane. Such separation makes the utilization of network resources more efficient. On the one hand, the programmability on data plane also brings much convenience for the network designers and managers to manipulate and utilize the underlying network infrastructure, where new network functions and strategies can be flexibly deployed [13,14]. On the other hand, a decoupled control plane makes it easier to obtain a global view of the network and accordingly deploy heterogeneous function entities on data plane. However, to fully exploit the benefits of centralized network management, one should be able to tackle the heavy work of dynamic information analysis and policy generation, and then convert the policy to flow tables for the data plane. Traditional routing optimization technologies such as load balancing and QoS guarantee algorithms fail in such new condition, of which most are based on off-line analytical optimization or heuristic methods, as presented in [15]. Recently, with the help of artificial intelligence, researchers are trying to solve such problems in a new way called network AI.

In this paper, we build an intelligent network control architecture TIDE (TIme-relevant DEep reinforcement learning for routing optimization) to realize the automatic routing strategy in SDN. The proposed architecture to perform intelligent network control contains three logic planes: data plane, control plane, and AI plane. The intelligent decision loop also consists of three parts: state, reward and collection, policy generation and policy deployment. The smart agent in the AI plane can output a near-optimal routing solution for the underlying network without any prior knowledge about the network or human experience. The main contributions of this paper are summarized as follows:

1. An intact "collections-decision-adjustment" loop is proposed to perform an intelligent routing control of a transmitting network.

2. An RNN-based deep reinforcement learning method is carefully designed which is suitable for traffic characteristics abstracting and can dynamically output a near-optimal routing strategy according to the varying traffic distribution.

3. The proposed architecture is implemented and validated on a real transmitting network topology, while to our best knowledge, previous state-of-art works only validate their schemes on simulation tools such as NS3.

The rest of this paper is organized as follows. In Section 2, we review the related works. In Section 3, we introduce the overall architecture of TIDE and explain the function of each plane. In Section 4, the interaction mechanism between TIDE and the network environment is explained. In Section 5, we explain the intelligent algorithm of TIDE and elaborate the corresponding parameter setting. In Section 6, we explain the experiment setup of a real network that validates TIDE, and analyze the experiment result. In Section 7, we conclude this paper and discuss the challenge and our future work of AI in network usage.

2. Related works

There are many researches focusing on the routing optimization problem, especially QoS guarantee [15,16]. Among them, most recent schemes are based on SDN or machine learning. Melinda Barabas et al. [17] proposed a simple routing management scheme based on the QoS information to improve the overall network traffic capacity, which still needed a lot of further consideration in practical use. Wenhao Huang et al. [18] employed a deep neural network to characterize the node traffic, but the acquisition of correct labeled data was hard. There are also other supervised learning methods that are based on the characterization ability of deep neural network, such as [19]. Many other solutions to solve the QoS limitations of the current networking technologies such as [15] and [20] either failed or were not implemented because these approaches come with many challenges. For example, dynamic QoS routing can be seen as a Constrained Shortest Path (CSP) problem, which is an NP-complete problem [15]. On the other hand, the advantage of reinforcement learning (RL) is being considered. Justin et al. [21] started the research of RL in the context of routing optimization, and [22] applied RL to achieve QoS routing. J. Jianget al. [23] proposed a Q-learning based architecture to improve end-to-end HTTP media stream transmitting. However, a transmitting network is a complicated continuous-time system with almost countless state number, while the researches mentioned above all use a state–action table to find a certain routing strategy, which is hard to deal with too many states. Giorgio et al. [24] used deep reinforcement learning to deal with such problem, but the proposed algorithm did not perform well compared to traditional routing algorithms. Haipeng Y et al. proposed an intelligent network architecture named NetAI [25], which analyzed the advantage of SDN architecture in the implementation of an intelligent network, and run a simple test to show that NetAI performed better than traditional shortest path algorithm. However, NetAI was only validated with simulations, and NetAI did not have an explicit explanation of the intelligent algorithm in use. In TIDE, we implement the automatic control architecture on a real network, and furthermore, we explain in detail the appropriate intelligent algorithm that matches the characteristics of network traffic.

3. Architecture of TIDE

In this section, we introduce the overall architecture of our time-relevant deep reinforcement learning network control architecture (TIDE) and its corresponding operating mechanism. The framework of TIDE is shown in Fig. 1, which is composed of three planes: data plane, control plane and AI plane.

The control plane and data plane are built upon a typical SDN network. The data plane is mainly composed of forwarding devices such as Openflow enabled switches. Such programmable forwarding devices can be used to execute flexible forwarding policy, and are also convenient for the network status collection process. For example, in OpenFlow v1.3, 39 matching fields are supported, which provides great granularity for network managers to classify and schedule traffic flows in the network; each flow entry has a counter field to reflect the intensity of each flow, which can be used to reflect the traffic distribution in the network. Online update of forwarding rules is supported on such devices; thus, a controller can make real-time changes on the routing policy to dynamically adjust the network condition.

The control plane connects the data plane with southbound interface and the AI plane with northbound interface. With the southbound interface, the controller can collect the network status such as the flow table statistics, resource availability, etc.

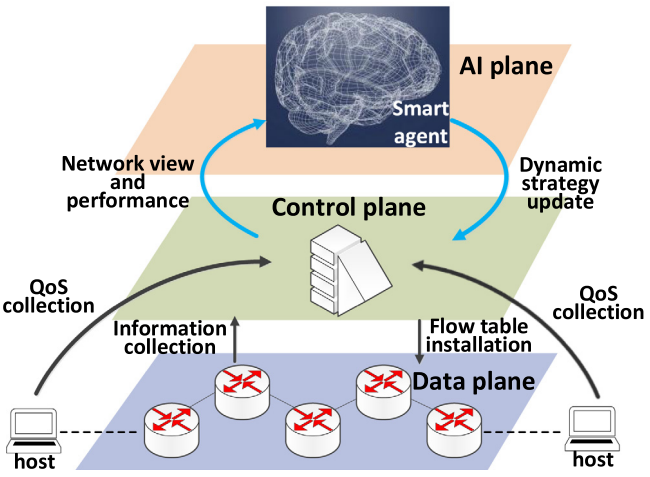


Fig. 1. System architecture of TIDE.

With such information, the control policies can be made based on a high abstraction of the network. Also, the controller collects the QoS measurement information which is used to evaluate the performance of network strategies. The controller can also generate fine-grained routing policies or convert them to flow tables and install the flow tables on corresponding switches using the southbound interface. With the northbound interface, the controller sends the global view of the network as state input to the AI plane, and receives the dynamic strategy from the AI plane.

As the core part of TIDE, the AI plane functions as the brain of TIDE. In the AI plane, a smart agent uses the global view of the network and network performance as the input to an intelligent algorithm, and then outputs a link weight for each link in the network topology, which is used for routing strategy update. In each interaction with the network, the smart agent generates a routing strategy and then evaluates the performance of such policy according to the reward (e.g., service quality measured by QoS parameters) collected from the network. Then, the smart agent adjusts the algorithm parameters trying to get a higher reward. After a period of training the smart agent can learn enough experience from the interaction with the network environment and produce a near-optimal strategy for the network.

4. Interaction between TIDE and the network environment

As mentioned in Section 1, one of our contributions is the design of a "collection-decision-adjustment" loop for real network routing. This loop defines the working logic of TIDE, and also reflects the interaction process between TIDE and the network environment. Among all parts of the logic, "decision" relies on the intelligent algorithm in the AI plane, which will be discussed in Section 5. "collection" and "adjustment" contain a lot of interaction with the underlying network, which will be illustrated in this section.

4.1. Network information collection

"collection" process includes the information collection of network status and routing strategy performance. As the input to AI plane, the granularity and timeliness of information collection can greatly affect the performance of the smart agent. Intuitively, according to sampling theorem, to discover the pattern of general network traffic, a status report with a frequency at least twice the main frequency of the traffic flow (here we use main frequency because there are always a few flows that cannot be predicted, which is illustrated in Section 5.2) should be guaranteed. Therefore, a proper information collection scheme should be selected.

There many network monitoring methods in SDN networks, usually categorized as pushing [26], polling [27] and INT [28] methods. Pushing methods have low computation overhead, but are coarse in granularity. Polling methods provide flexible monitoring of the network, but are less flexible in the information format. INT attaches the network information (e.g., link utilization, latency) to every packet at every switch that supports INT. In this way, a real-time and fine-grained network status can be sent to the controller, but the packet-level information reports lead to a high processing pressure on the controller. In practice, a proper network monitoring method should be chosen according to the capability of the hardware in use and the granularity required of the information. For pushing methods, the controller has no control over when to get required information, thus the efficiency of the smart agent may not be fully achieved in TIDE, while INT is not supported on the hardware of our testbed. Therefore, in the validation of TIDE, we use a polling method to acquire the network information with a controlled rate, as shown in Fig. 2. Generally, in a distributed control plane, there are multiple controllers to maintain the network status. In our experiment, we use a single controller to illustrate the effectiveness. In a distributed control plane, the consistency problem of multiple controllers has been researched in many studies. The format of information depends on the target of routing strategy, which consists of two parts: network status information and QoS measurement information. For example, in QoS guarantee, end-to-end QoS parameters should be considered in policy evaluation information, including end-to-end delay and packet loss, and the corresponding network status information should include link capacity and link utilization. In Section 6, we validate TIDE with QoS guarantee as a representative application.

4.2. Adjustment with intelligent action

The AI plane make decisions to adjust the routing policy based on the information collected. The adjustment is according to the neural network output in the smart agent, which has a certain meaning in the network, e.g., each output node of the neural network denotes a link weight value. When an adjustment is generated, it is then sent to the controller through the northbound interface. In the controller, the adjustment is converted to a routing policy, which takes on the form of a group of flow tables for each switch. Generally, consistency of flow table is one of the main considerations in flow table update, which are categorized into two types: per-packet consistency and per-flow consistency [29]. Without a proper flow table update scheme, the asynchronism of switches in the network will lead to wrong paths of some flows during the process of flow table update. In TIDE, we use a intermediate transfer method [30] to ensure the consistency.

5. Intelligent routing algorithm of TIDE

In TIDE, we use DDPG (Deep Deterministic Policy Gradient) [31], a deep reinforcement learning model for continuous control as the main algorithm. Different from a majority of reinforcement learning models such as DQN, the output of DDPG is not discretized as a small set of fixed actions. This character makes DDPG suitable for the generation of continuous actions. In routing

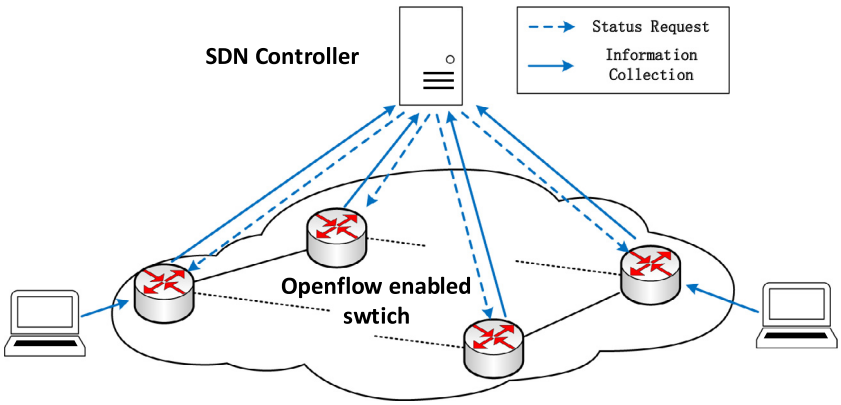


Fig. 2. Polling method of network monitoring.

optimization, we usually need to adjust the link weight for each link to delicately manipulate all traffic flows in the network (as illustrated in Section 5.2), thus the link weight value space should be large in a scaled network, which makes continuous control algorithms such as DDPG suitable for routing strategy design. Section 5.1 introduces the core idea behind DDPG and its suitability in the routing optimization. Section 5.2 introduces how DDPG is combined with routing optimization application in TIDE.

5.1. Mathematical model of DDPG

Typically, reinforcement learning regards the interaction process between the environment and agent as a Markov Decision Process (MDP). The basic element tuple of such MDP is $M = (S, A, R, P, \gamma)$, where S is the space of state, A the space of action a, R the space of reward r, P the transition probability function ($p(s_{t+1}, r|s_t, a_t) \in P$), and $\gamma \in [0, 1]$ the discount factor. An agent chooses action a under state s according to a policy, which is denoted as $\pi(a|s)$ in stochastic policies and $a = \mu(s)$ in deterministic policies, as illustrated in [32].

To evaluate whether a policy is good or not, value functions are introduced. One popular value function used in reinforcement learning is the Q value. The policy value Q under state s when choosing action a is defined as (1):

$$Q(s_t, a_t) = E[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k})] \tag{1}$$

The bootstrap form above can properly represent the value, but the acquisition of the future reward is only suitable for offline learning, which is improper in the network usage. Such Q value can also be expressed in an iterative mode as (2):

$$Q(s_t, a_t) = E[R_t + \gamma Q(s_{t+1}, a_{t+1})] \tag{2}$$

Eq. (2) is more convenient for us to introduce the policy gradient function as illustrated in the following.

For policy gradient in continuous environment, we need to represent the Q value with a function approximator instead of the simple cases where Q values are stored and searched in a table. In this paper, we use a neural network as the function approximator. Thus, $Q(s, a)$ is more accurately specified as $Q(s, a|\theta)$, where θ is the set of parameters in the neural network. For the evolution of the neural network, we need a loss function to carry out the back propagation. As defined in DDPG, the loss function is as (3):

$$L(\theta) = E[(Q(s_t, a_t|\theta) - y_t)^2] \tag{3}$$

of which y_t is defined as follows:

$$y_t = R(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta) \tag{4}$$

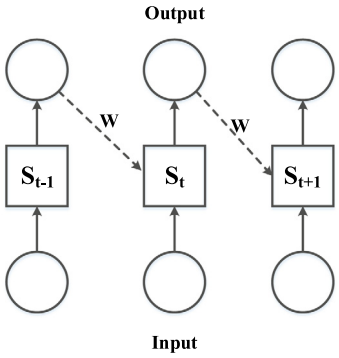


Fig. 3. RNN network logic.

Therefore, the main policy gradient method of DDPG is defined as (5), with $a = \mu(s|\theta^\mu)$ as the actor network to choose an action under a certain state and $Q(s, a|\theta^Q)$ as the critic network to evaluate the policy value.

$$\nabla_{\theta^Q} J \approx E[\nabla_\theta Q(s, \mu(s|\theta^\mu)|\theta^Q)] = E[\nabla_a Q(s, a|\theta^Q) \nabla_{\theta^\mu} \mu(s|\theta^\mu)] \tag{5}$$

To improve the stability of the learning, we use "soft" update method to update the parameters as illustrated by Mnih et al. [33]. The key idea is to create a copy of the actor and critic network, and the parametersθ' of such target network are updated slowly with $\theta' = \tau\theta + (1 - \tau)\theta'$, where $\tau \ll 1$.

5.2. Network application oriented combination with DDPG

Lakhina et al. [34] did a research on network traffic analysis and pointed out that Origin–Destination (OD) flows in network traffic had periodic flows as the main component. For example, in data-set Sprint-1 [35], more than 90% of its traffic content has a periodic feature. Therefore, time-relevance is one of the most indispensable traffic features in a transmitting network. In this paper, we assume that the network traffic is composed of two components: periodical flow PF and random flow RF. We use

$$RP = \frac{RF}{PF} \tag{6}$$

to denote the ratio of such two components. By selecting the traffic sequence instead of an instantaneous traffic distribution in the network, the agent can get a more accurate view of the network state.

In our design, we use RNN (Recurrent Neural Network) as the input neural network to exploit such feature in traffic. RNN is designed to process sequence data, and has now various popular

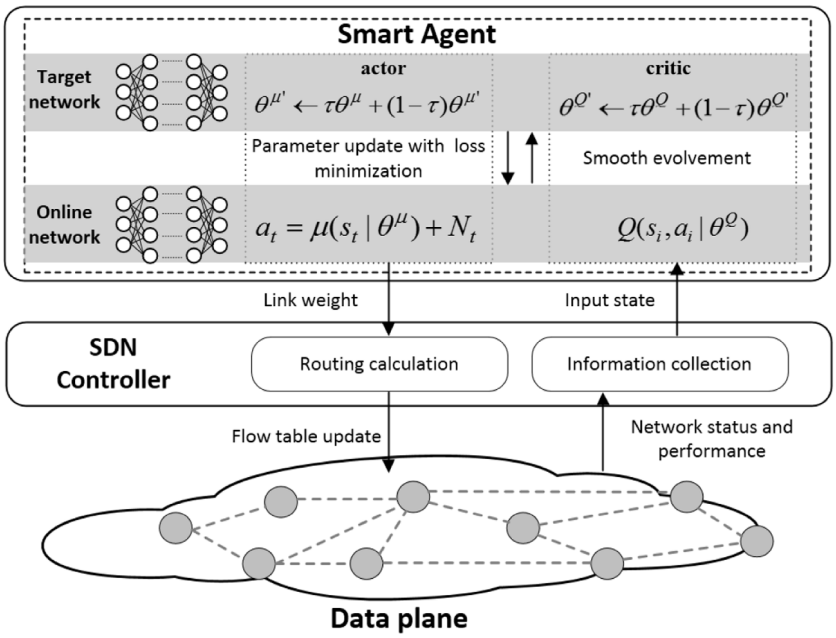


Fig. 4. Algorithm logic of TIDE.

Table 1
Different QoS sceneries.

	delay	jitter	throughput	loss
Video download			√	√
VoIP	√	√		
Virtual reality	√	√	√	√
Internet of things	√			√

versions such as LSTM (long short-term memory) , GRU (Gated Recurrent Unit) and BiRNN. The basic structure of RNN is shown in Fig. 3. After each time step, the output data also serve as the one of the inputs in the next time step. Then, the output of all RNN units is connected to a feed forward neural network, which produces the final outcome (which is the link weight in TIDE). In TIDE, we use the basic type of RNN as shown in Fig. 3, and the output neurons are connected to a feed forward neural network with one hidden layer.

The state of network is represented by the sequence of network status information, which is denoted by a state matrix$r$, where t is the length of time series and n is the number of switches in the network. The action r is a weight matrix that defines the link weight for each duplex link between any two routers, where l is the number of links.

As mentioned in Section 2, we take QoS guarantee as a representative application of routing optimization. Under the circumstance of QoS guarantee, the reward$r$ is calculated from the QoS feedback from the network. $r$could be flexibly calculated with different indices under different QoS strategy, for example,

$$r = w_1 \cdot delay + w_2 \cdot jitter + w_3 \cdot throughput + w_4 \cdot loss \tag{7}$$

of which w_i is the weight parameter. In practice, different applications have different performance indices, such as shown in Table 1.

The deep reinforcement learning agent takes the network state as neural network input data and then output a tuple of l link weight values corresponding to l links among switches. With such a link weight tuple, a Floyd–Warshall algorithm is run to calculate the routing path for each traffic flow, and such paths are then converted to routing tables in the controller for all the switching nodes.

Table 2
Deep reinforcement learning algorithm of TIDE.

Input: hyperparameters of neural network, discount factor γ, smooth factor τ;
Randomly initialize θ^{Q} and θ^{μ} of the neural network;
Create target network Q' and μ' with $\theta^{Q'} \leftarrow \theta^{Q}$ and $\theta^{\mu'} \leftarrow \theta^{\mu}$;
Initialize replay buffer B;
for episode = 1 to M:
  Initialize a random process N for action exploration;
  Initialize a fifo as the state sequence SS;
  Receive the initial state s_1 and update SS_1;
  for t = 1: STEP_NUM
    execute $a_t = \mu(s_t|\theta^{\mu}) + N_t$, observe r_t and s_{t+1};
    update SS_{t+1};
    store $(SS_t, a_t, r_t, SS_{t+1})$ in B;
    sample a minibatch of $(SS_t, a_t, r_t, SS_{t+1})$ from B;
    set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$;
    update θ^{Q} with $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2$;
    update θ^{μ} with $\nabla_{\theta^{Q}} J \approx \frac{1}{N}\sum_i [\nabla_a Q(SS_i, a_i|\theta^{Q}) \nabla_{\theta^{\mu}} \mu(SS_i|\theta^{\mu})]$;
    update target network :
    $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$;
    $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$;
  end for
end for
Output θ^{μ} as the routing strategy parameter

The overall AI frame work of TIDE is shown in Fig. 4, and the routing strategy generation algorithm is shown in Table 2.

6. Experiment and analysis

In this section, we present our experiment on a real network to validate TIDE. Section 6.1 introduces the experiment environment and parameter setting, and Section 6.2 shows and analyzes the experiment result.

6.1. Testbed setup

Our scheme is evaluated with the topology of OS3E [35], a typical topology for network experiment. To make our hardware devices compatible with the topology, we adjust the link capacity

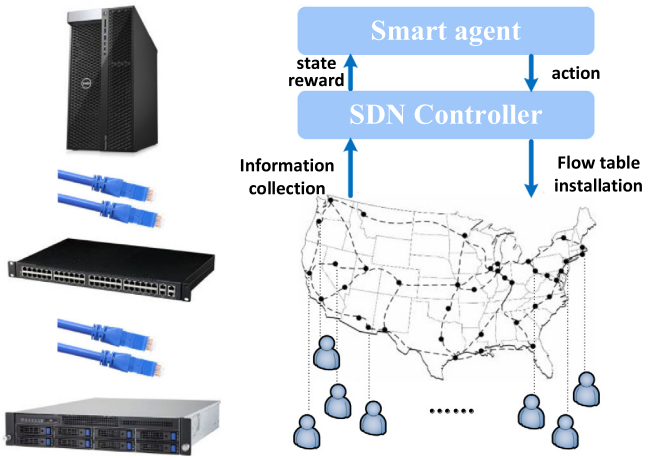


Fig. 5. Experiment network.

and characteristics accordingly. For each routing node, there is a host machine connected to it, and each host is a virtual machine run on a server. The routing nodes are implemented with PICA8 switches, and 38 hosts are run on 5 servers. In each host, several scripts are run to generate different types of traffic, and such network traffic can approximately imitate the traffic transmitted to and received from the transmitting network in a certain edge routing node. The POX controller is run on a DELL work station with a CPU of Xeon E5-2600 v4 and 128G memory. The smart agent is implemented with Keras based on python 2.7, which is run on the same system with the controller. The whole network is shown as Fig. 5.

In the test network, each host connected to a controller serves as a traffic originator. Actually, the 38 switches that are controlled by the smart agent can be regarded as the core transmitting network, whereas there can also be other routing nodes connected to the topology which are not shown in the topology. Therefore, the traffic generated by the scripts of one host in our experiment can imitate the traffic generated either by multiple users connected to this switch or other routing nodes connected to this switch, as long as they have the similar complexity of the composition in the traffic. In this way, we find that minor changes in the scripts will not change the performance of the trained smart agent in our experiment, thus proving that TIDE can tolerate small changes in the network.

We create traffic patterns for these traffic originators, of which each flow is composed of two components: periodical flow PF and random flow RF as discussed in Section 5.2. Each traffic originator communicates a fixed group of destination hosts that is predefined before the experiment starts. In the experiment, the RP of each host (as defined in (6)) is regulated to generated different traffic for the network. We use the shortest path (SP) algorithm in the popular protocol OSPF (Open Shortest Path First) for contrast. OSPF is now widely used in intra-domain routing. In OSPF, link state is collected, and the routing path of one source–destination pair is calculated with Dijkstra algorithm based on the link state, which is a fundamental shortest path algorithm. Dijkstra algorithm is also widely used in IS-IS(Intermediate system to

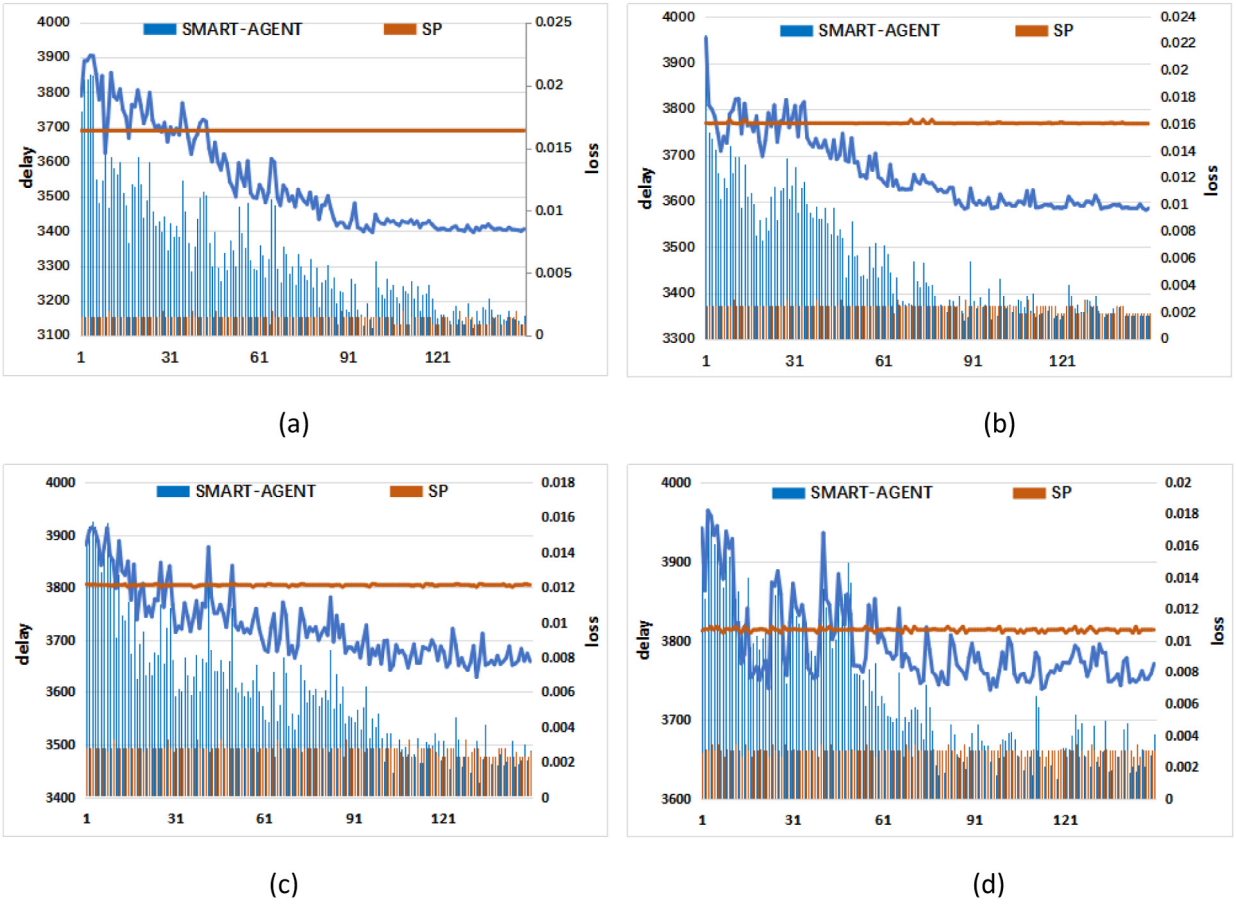


Fig. 6. Performance under different RP within 150 episodes of training. The RPs of a∼d are 0, 0.05, 0.10, 0.15 respectively, with the PF of each flow set to 0.5 Mbps. The *X*-axis shows the episode number of training, the line chart shows the amount of average transmission delay (ms) of each step within each episode, and the bar shows the average packet loss within each episode.

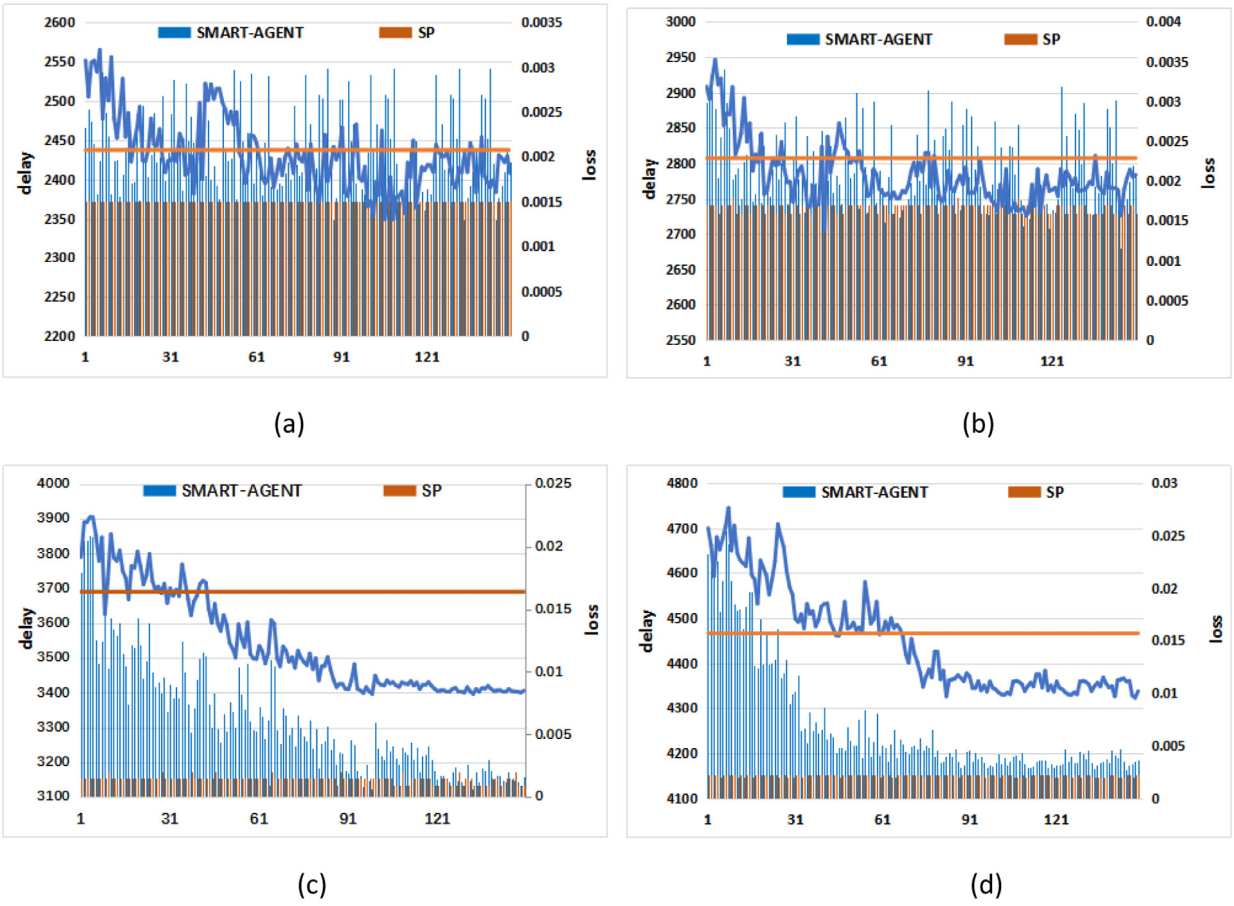


Fig. 7. Performance under different traffic intensity within 150 episodes of training. The average traffic intensity of a∼d are 0.2, 0.35, 0.5, 0.65 Mbps respectively, with the RP of each flow set to 0. The *X*-axis shows the episode number of training, the line chart shows the amount of average transmission delay (ms) of each step within each episode, and the bar shows the average packet loss within each episode.

intermediate system). The traffic intensity on each link is sent to the controller periodically as the network status, and the hosts also report the QoS indices to the controller periodically (set to 2 s in our experiment). Currently, we only use two indices: delay and loss.

In our experiment, each episode of training contains 1000 steps, and the average transmission delay of each step is added to the total delay of this episode to measure the performance.

6.2. Result analysis

The effectiveness of TIDE is validated in our test bed under different traffic intensities and different traffic composition. Figs. 6 and 7 shows the training process of the smart agent of TIDE. andAs shown in Fig. 6, we can find that in the first few dozens of training episodes, TIDE does not learn enough knowledge, therefore the agent uses stochastic strategy that performs worse than SP. As the training step increases, the total traffic delay and loss of TIDE is on the trend of decline. After about a hundred of training episodes, the over all delay of TIDE is obviously lower than SP when RPs are low, and the corresponding transmitting loss are basically equivalent. This means that the smart agent of TIDE has intelligently gained routing knowledge without human experience. After the training process, the smart agent is then ready to be used in routing. In the best case, TIDE reduces the total transmitting delay of all traffic by about 9%. Also, we can judge from Fig. 6(a) to Fig. 6(d) that with the increase of RP, TIDE shows decreasing advantage over SP, this is mainly because that the increasing randomness of noise traffic makes it harder for TIDE to characterize the network traffic, thus impairing TIDE's ability to make accurate decision.

Fig. 7 shows the performance comparison between the two schemes under different traffic intensity. Compared with the other three diagrams, in Fig. 7(a), TIDE shows just a small advantage over SP. This mainly results from the fact that when the traffic intensity is low, the transmission delay and loss caused by packet queueing in switches is not the dominating source, e.g., packet processing delay in the protocol stack of hosts also contributes to the overall delay. Therefore, in this condition TIDE cannot reach a significant advantage. When the traffic intensity rises from 0.2 Mbps to 0.5 Mbps, TIDE shows an obvious improvement in advantage, but the advantage decreases again in the traffic intensity of 0.65 Mbps. The reason may be that our PICA8 switch has only a port buffer of 10M bits, so a packet queue larger than this threshold in the ports will lead to much packet loss, which is non-linear to the traffic intensity. Since the overall QoS takes in both delay and loss into account, TIDE has to make a trade-off of overall transmitting delay to avoid too much packet loss.

Fig. 8 compares the running time of the smart agent of TIDE and SP when generating a routing path for each flow. As shown in the figure, the running time of TIDE is less than SP, and as the scale of the network grows, TIDE saves more running time. The calculation cost of TIDE mainly consists of two parts: link weight calculation in the neural network and routing path calculation based on the link weight. Once the training process of the smart agent is finished, the link weight calculation time is determined by the structure of the neural network, of which the calculation

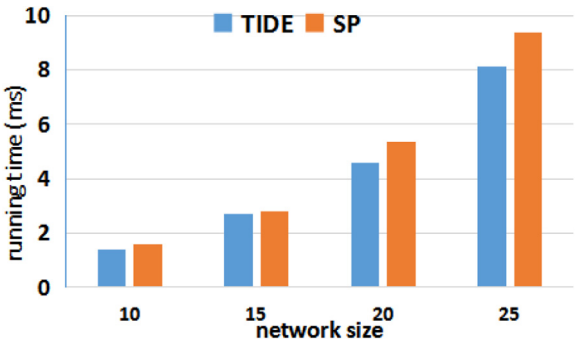


Fig. 8. Running time comparison.

is mainly a series of matrix multiplication, which is $O(n_1 \times n_2 + n_2 \times n_3 \ldots + n_{d-1} \times n_d)$, where $n_i (i = 1, 2, \ldots \mathrm{d})$ is the number of neurons in each layer of the neural network, and d is the number of layers. The routing path calculation uses Floyd–Warshall algorithm, whose time complexity is $O(N^3)$, where N is the number of switching nodes in the network. Therefore, the total time complexity of TIDE is $O(n_1 \times n_2 + n_2 \times n_3 \ldots + n_{d-1} \times n_d) + O(N^3)$. In practice, matrix calculation is usually much faster than iteration, and according to the experiment result, the computation time of neural calculation is negligible. On the other hand, OSPF uses Dijkstra algorithm, to give a path for all node pairs the time complexity is also$O(N^3)$, but it only does the calculation of routing path for each flow, without a global optimization for all flows. Actually, the global optimization problem is an NP-hard problem as mentioned in [15], and previous solutions are mostly heuristic algorithms such as ant colony problem, whose time complexity is$O(N \times (N-1) \times mt)$ [30], where N is the number of switching nodes, m is the ant number, and t is the iteration.

7. Conclusion and future work

In this paper, we demonstrate the advantage of deep reinforcement learning in routing optimization. As a model-free scheme, the proposed scheme can dynamically generate a near-optimal routing strategy for the network once trained. Compared to traditional routing algorithms, RL based method can quickly process a large amount of network status data and output a proper strategy following the change of traffic distribution in the network once deployed online. Therefore, such methods have a great potential in replacing traditional routing strategies.

Though inspiring result has been reached in this paper that proves the advantage of artificial intelligence in traffic engineering over traditional algorithms, there are still many challenges.

First, the time and contents granularity of network status influences the performance of the smart agent. To acquire a more detailed analysis of the network environment, the agent should take in more frequent reports of network status and performance, and finer-grained flow information as the input of the neural network. For example, elephant flows and mice flows can be separately treated in the smart agent to balance the QoS of different flows. However, this brings more communication overhead to the network, especially to the centralized controller. Furthermore, in this paper, the performance of TIDE is validated in networks with the scale of dozens of nodes, but in a much larger network (e.g., a network with thousands of nodes), the information collection and processing might also be a problem. To address such problem, a trade-off between the performance and communication overhead should be made for different scenarios.

Second, the smart agent uses a neural network which takes the network view as the input. However, when the topology changes, the learned knowledge may fail, thus a re-training process is required. Therefore, such architecture is temporarily more suitable for stable environments such as data center network. A knowledge-transfer mechanism should be researched in the near future to ensure that a trained agent can be transplanted to other networks with similar topology without much re-training cost.

Acknowledgments

The research of this paper is supported by the National Key Research and Development Plan (grant number 2017YFB0803204), the National Natural Science Fund (grant numbers 61521003, 61872382) and the Research and Development Program in Key Areas of Guangdong Province (grant number 2018B010113001). Prof. Min Chen's work is supported by the National Key R&D Program of China (2017YFE0123600, 2018YFC1314605), and the China National Natural Science Foundation for Innovative Research Groups (No. 61821003), and Director Fund of WNLO.

References

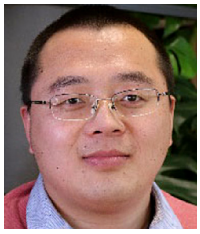
[1] H.E. Egilmez, S.T. Dane, K.T. Bagci, et al., Openqos: An openflow controller design for multimedia delivery with endto-end quality of service over software-defined networks, in: Signal & Information Processing Association Summit and Conference, 2012, pp. 1–8.
[2] J. Liu, N.B. Shroff, C.H. Xia, et al., Joint congestion control and routing optimization: an efficient second-order distributed approach, IEEE/ACM Trans. Netw. (2015) 1–17.
[3] D.D. Clark, C. Partridge, J.C. Ramming, et al., A knowledge plane for the internet, in: Proceedings of the SIGCOMM'03 Karlsruhe, Germany, 2003J. Clerk Maxwell, A Treatise on Electricity and Magnetism, vol. 2, third ed., Clarendon, Oxford, 1892, pp. 68–73.
[4] R.W. Thomas, L.A. Dasilva, A.B. Mackenzie, Cognitive networks, in: First IEEE International Sympo sium on New Frontiers in Dynamic Spectrum Access Networks, Baltimore, MD, USA, 2005.
[5] H. Derbel, N. Agoulmine, M. Salaün, Anema: autonomic network management architecture to support self-configuration and self-optimization in ip networks, Comput. Netw. 53 (3) (2009) 418–430.
[6] M. Zorzi, A. Zanella, A. Testolin, et al., Cobanets: a new paradigm for cognitive communications systems, in: International Conference on Computing, NETWORKING and Communications, IEEE, 2016, pp. 1–7.
[7] A. Mestres, A. Rodrigueznatal, J. Carner, et al., Knowledge-defined networking, ACM Sigcomm Comp. Commun. Rev. 47 (3) (2016) 2–10.
[8] M. Chen, Yixue Hao, Task offloading for mobile edge computing in software defined ultra-dense network, IEEE J. Sel. Areas Commun. 36 (3) (2018) 587–597.
[9] M. Chen, Y. Hao, K. Lin, L. Hu, Z. Yuan, 'Label-less learning for traffic control in an edge network', IEEE Netw. 32 (6) (2018) 8–14.
[10] M. Chen, X. Shi, Y. Zhang, et al., Deep features learning for medical image analysis with convolutional autoencoder neural network, IEEE Trans. Big Data 2017 (2017) http://dx.doi.org/10.1109/TBDATA.2017.2717439.
[11] N. Agoulmine, S. Balasubramaniam, D. Botvitch, et al., Challenges for autonomic network man agement, in: 1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE), 2006.
[12] O.N. Foundation, Software-defined networking: The new norm for networks, 2012.
[13] Nick. McKeown, Tom. Anderson, Hari. Balakrishnan, et al., Openflow: enabling innovation in cam pus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[14] A. Sivaraman, C. Kim, R. Krishnamoorthy, et al., Dc. p4:programming the forwarding plane of a data-center switch, in: ACM SIGCOMM, 2016.
[15] M. Karakus, A. Durresi, 'Quality of service (qos) in software defined networking (sdn): a survey', J. Netw. Comput. Appl. 80 (2017) 200–218.
[16] Ning Wang, Kin Ho, George Pavlou, Michael Howarth, An overview of routing optimization for internet traffc engineering, IEEE Commun. Surveys Tuts. 10 (1) (2008).
[17] M. Barabas, G. Boanea, R. Andrei Bogdan, Dobrota. V., Congestion control based on distributed statistical qos-aware routing management, Prz. Elektrotech. 89 (2b) (2013) 251–256.
[18] W. Huang, G. Song, H. Hong, K. Xie, Deep architecture for traffic flow prediction: deep belief networks with multitask learning, IEEE Trans. Intell. Transp. Syst. 15 (5) (2014) 2191–2201.
[19] N. Kato, Z.M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, K. Mizutani, The deep learn ing vision for heterogeneous network traffic control: proposal, challenges, and future perspective, IEEE Wirel. Commun. 24 (3) (2017) 146–153.

[20] J.A. Boyan, M.L. Littman, Packet routing in dynamically changing networks: a reinforcement learning approach, in: International Conference on Neural Information Processing Systems, 1993, pp. 671–678.
[21] Justin A. Boyan, Michael L. Littman, Packet routing in dynamically changing networks: a rein forcement learning approach, in: Advances in Neural Information Processing Systems, 1994, pp. 671–678.
[22] Shih-Chun Lin, Ian F. Akyildiz, Pu Wang, Min Luo, QoS-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach, in: 2016 IEEE International Conference on Services Computing (SCC), 2016, pp. 25–33.
[23] J. Jiang, L. Hu, P. Hao, et al., Q-fdba: improving qoe fairness for video streaming, Multimed. Tools Appl. (2) (2017) 1–20.
[24] Giorgio. Stampa, Marta. Arias, David. Sánchez-Charles, Victor. Muntés-Mulero, Albert. Cabellos, A deep-reinforcement learning approach for software-defined networking routing optimization, 2017, arXiv preprint arXiv:1709.07080.
[25] Y. Haipeng, M. Tianle, X. Xiaobin, et al., Networkai: an intelligent network architecture for self-learning control strategies in software defined networks, IEEE Internet Things J. (2018) 1-1.
[26] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, FlowSense: Monitoring network utilization with zero measurement cost, in: International Conference on Passive and Active Network Measurement, 2013, pp. 31–41.
[27] N.L.M. van Adrichem, C. Doerr, F.A. Kuipers, Opennetmon: network monitoring in openflow software-defined networks, in: Network Operations and Management Symposium (NOMS), vol. 2014, IEEE, 2014, pp. 1–8.
[28] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L.J. Wobker, B. Networks, In-band network telemetry via programmable dataplanes, 2015.
[29] Reitblatt. Mark, et al., Consistent updates for software-defined networks: change you can believe in!, in: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, ACM, 2011.
[30] McGeer. Rick, A safe, efficient update protocol for openflow networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ACM, 2012.
[31] T.P. Lillicrap, J.J. Hunt, A. Pritzel, et al., Continuous control with deep reinforcement learning, Comput. Sci. 8 (6) (2015) A187.
[32] D. Silver, G. Lever, N. Heess, et al., Deterministic policy gradient algorithms, in: International Conference on International Conference on Machine Learning. JMLR.org, 2014, pp. 387–395.
[33] V. Mnih, K. Kavukcuoglu, D. Silver, et al., Playing atari with deep reinforcement learning, Comput. Sci. (2013).
[34] A. Lakhina, K. Papagiannaki, M. Crovella, et al., Structural analysis of network traffic flows, in: Joint International Conference on Measurement and Modeling of Computer Systems, ACM, 2004, pp. 61–72.
[35] https://www.internet2.org/news/detail/4865/.

Penghao Sun (sphshine@126.com) is currently a Ph.D candidate in National Digital Switching System Engineering and Technological R&D Center (NDSC), China. He received the B.S and M.S degrees from NDSC in 2014 and 2017 respectively. His current research interests include network architecture, edge computing and machine learning.

Yuxiang Hu (chxachxa@126.com) received his Ph.D. degree in 2011 in National Digital Switching System Engineering and Technological Research Center of China. He is currently a associate professor, and his research interests include Internet architecture, novel switching and routing, multimedia network technology and so on.

Julong Lan (ndscljl@163.com) is currently a professor and the chief engineer in National Digital Switching System Engineering and Technological R&D Center (NDSC), China. His is also the Chief scientist of China National Program on Key Basic Research Project (973 Program). His current research interests include 5G communication system, next generation of computer network and artificial intelligence.



Tian Le (letian@126.com) is currently a Ph.D. student in University of Antwerp, Belgium. His research interests focus on protocol and algorithm design for Internet of things, especially on the scalability and energy efficiency of IEEE 802.11ah technologies.

Min Chen is a full professor in School of Computer Science and Technology at Huazhong University of Science and Technology (HUST) since Feb. 2012. He is the director of Embedded and Pervasive Computing (EPIC) Lab at HUST. He is Chair of IEEE Computer Society (CS) Special Technical Communities (STC) on Big Data. He was an assistant professor in School of Computer Science and Engineering at Seoul National University (SNU). He worked as a Post-Doctoral Fellow in Department of Electrical and Computer Engineering at University of British Columbia (UBC) for three years. Before joining UBC, he was a Post-Doctoral Fellow at SNU for one and half years. He received Best Paper Award from QShine 2008, IEEE ICC 2012, ICST IndustrialIoT 2016, and IEEE IWCMC 2016. He serves as technical editor or associate editor for IEEE Network, Information Sciences, Information Fusion, and IEEE Access, etc. He served as a leading Guest Editor for IEEE Wireless Communications, IEEE Network, and IEEE Trans. Service Computing, etc. He is a Series Editor for IEEE Journal on Selected Areas in Communications. He is Co-Chair of IEEE ICC 2012-Communications Theory Symposium, and Co-Chair of IEEE ICC 2013-Wireless Networks Symposium. He is General Co-Chair for IEEE CIT-2012, Tridentcom 2014, Mobimedia 2015, and Tridentcom 2017. He is Keynote Speaker for CyberC 2012, Mobiquitous 2012, Cloudcomp 2015, IndustrialIoT 2016, Tridentcom 2017 and The 7th Brainstorming Workshop on 5G Wireless. He has 300+ publications, including 200+ SCI papers, 100+ IEEE Trans./Journal papers, 28 ESI highly cited papers and 9 ESI hot papers. He has published eleven books: OPNET IoT Simulation (2015), Big Data Inspiration (2015), 5G Software Defined Networks (2016) and Introduction to Cognitive Computing (2017) with HUST Press, Big Data: Related Technologies, Challenges and Future Prospects (2014) and Cloud Based 5G Wireless Networks (2016) with Springer, Cognitive Computing and Deep Learning (2018) with China Machine Press, and Big Data Analytics for Cloud/IoT and Cognitive Computing (2017) with Wiley. His Google Scholar Citations reached 16,500+ with an h-index of 63 and i10-index of 203. His top paper was cited 1890+ times. He is an IEEE Senior Member since 2009. He was selected as Highly Cited Research at 2018. He got IEEE Communications Society Fred W. Ellersick Prize in 2017. His research focuses on cognitive computing, 5G Networks, embedded computing, wearable computing, big data analytics, robotics, machine learning, deep learning, emotion detection, IoT sensing, and mobile edge computing, etc.