ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/346541681

DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks

Article *in* Journal of Network and Computer Applications · November 2020

DOI: 10.1016/j.jnca.2020.102865

CITATIONS
19

READS
817

4 authors, including:

Waixi Liu
Guangzhou University
30 PUBLICATIONS **328** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

the National Key R&D Program of China View project

All content following this page was uploaded by Waixi Liu on 28 January 2021.

The user has requested enhancement of the downloaded file.

Journal of Network and Computer Applications 177 (2021) 102865

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks

Wai-xi Liu [a,*], Jun Cai [b,**], Qing Chun Chen [c], Yu Wang [c]

[a] Department of Electronic and Communication Engineering, Guangzhou University, Guangzhou, PR China
[b] Guangdong Polytechnic Normal University, Guangzhou, PR China
[c] Guangzhou University, Guangzhou, PR China

ARTICLE INFO

Index Terms:
Deep reinforcement learning
Routing
Network resource
Software-defined networking
Data-center networks

ABSTRACT

Data-center networks (DCN) possess multiple new features: coexistence of elephant flow/mice flow/coflow, and coexistence of multiple network resources (bandwidth, cache and computing). The cache should be a factor of effecting routing decision because it can eliminate redundant traffic in DCN. However, the conventional routing schemes cannot learn from their previous experiences regarding network abnormalities (such as, congestion), and their metric are still the single link state (such as, hop, distance, and cost) which does not include the effect of cache. Thus, they cannot enough efficiently allocate these resources to well meet the performance requirements for various flow types. Therefore, this paper proposes deep reinforcement learning-based routing (DRL-R). Firstly, we propose a method that recombines multiple network resources with different metrics, where we recombine cache and bandwidth by quantifying their contribution score in reducing the delay. Secondly, we propose a routing scheme with resource-recombined state. By optimally allocating network resources for traffic, a DRL agent deployed on a software-defined networking (SDN) controller continually interacts with the network to adaptively perform reasonable routing according to the network state. We employ deep Q-network (DQN) and deep deterministic policy gradient (DDPG) to build the DRL-R. Finally, we demonstrate the effectiveness of DRL-R through extensive simulations. Benefitting from continuous learning with a global view, DRL-R has lower flow completion time, higher throughput and better load balance as well as better robustness, compared to OSPF. In addition, because it efficiently utilizes the network resources, DRL-R can also outperform another DRL-based routing scheme (namely TIDE). Compared to OSPF and TIDE, respectively, DRL-R can improve throughput by up to 40% and 18.5%; DRL-R can reduce flow completion time by up to 47% and 39%; DRL-R can improve the link load balance by up to 18.8% and 9.3%. Additionally, we observed that DDPG has better performance than DQN.

1. Introduction

Cluster computing for big-data application (Mapreduce/Spark, Web search, etc.) creates considerable coflow (Wang et al., 2018a) in data-center networks (DCN). In other words, elephant flow/mice flow/coflow coexist in DCN. More importantly, these various types of flows have different performance requirements to network. Elephant flow expects higher throughput and smaller flow completion time (FCT). On the other hand, mice flow is generally a delay sensitivity application and expects higher deadline meet rate. Finally, a part of the coflow (Mapreduce/Spark application) expects smaller coflow completion time, and the other part (Web search) expects higher deadline meet rate.

Research (Cui et al., 2017) has shown that, MapReduce application, Internet search services, etc., generate lots of redundancy traffic (Cui et al., 2017; Chen et al., 2020), and deploying cache on the switch in a DCN can reduce the path length for accessing data, avoid data hotspots, and improve throughput. Since the cache has become a new network resource, DCN shows other new feature: multiple network resources (bandwidth, cache, and computing) coexist. Thus, the cache should be a factor of effecting routing decision. Additionally, the deployment of cache also leads the DCN to become a network with multiple data sources for users.

* Corresponding author.
** Co-corresponding author.
E-mail address: liuwaixi@sina.com (W.-x. Liu).

https://doi.org/10.1016/j.jnca.2020.102865
Received 2 October 2019; Received in revised form 20 September 2020; Accepted 3 October 2020
Available online 2 November 2020
1084-8045/© 2020 Published by Elsevier Ltd.

Owing to various types of flows competing for limited network resources, research (Chen Chenet al., 2016) has shown that existing flow-scheduling methods can well meet the performance requirement for one type of flow but at the expense of hurting the performance for other type of flow. Thus, from another perspective, in a DCN with multiple network resources, the essence of routing actually become into properly allocating network resources to meet different requirements of various type of flows. On the hand, the multiple network resources coexisted lets the selection of routing become more diverse and complex. On the other hand, elephant flow/mice flow/coflow coexisted lets the object of routing become more diverse and complex. Correspondingly, the two new coexists have brought the considerable complexity of allocating network resource and challenges to the routing of DCN.

In DCN, widely-used routing solutions include scheduling traffic via the shortest paths (e.g., open shortest path first, OSPF), equal-cost multipath routing (ECMP), based on queueing theory (Xu et al., 2011), distributed adaptive routing for larger and longer traffic flows (Zahavi et al., 2014), as well as software-defined networking (SDN)-based routing by hybrid addressing (Medhi and Saikia, 2017). However, they do not learn from their previous experiences regarding network abnormalities, such as congestion and so forth. Furthermore, existing methods cannot properly address network dynamics, including time-varying resource state and flow requirement (Valadarsky et al., 2017; Xu et al., 2018).

Recently, deep reinforcement learning (DRL) (Mnih et al., 2015), a combination of the perception ability of deep learning (DL) (LeCun et al., 2015) with the decision-making ability of reinforcement learning (RL) (Li, 2017) in a common form, has shown a dramatic improvement in decision-making and automated control problems. Owing to DRL's advantages, such as dealing with highly dynamic time-variant environments and handling a sophisticated state space, some researchers believe that DRL is especially promising for traffic engineering (Chen et al., 2018; Xu et al., 2018; Wang et al., 2018b; Luong et al., 2019), IP routing (Valadarsky et al., 2017; Sun et al., 2019) and optical routing (Suárez-Varela et al., 2019). However, they all neglect DCN's the new feature: multiple network resources coexist. Their routing metric is still the single link state, such as hop, distance, cost, etc. In other words, existing routing/flow-scheduling methods for DCN do not allocate these network resources enough efficiently to well meet the performance requirement for various types of flow.

Incorporating SDN (Kreutz et al., 2015) into DCN, software-defined data-center networks (SD-DCN) has gained attention recently (Jain et al., 2013). In SD-DCN, the SDN controller can centrally control the DCN's traffic from a global view; on the other hand, the DRL agent needs to continually interact (learn) with the network from a global view. Thus, deploying the DRL agent on the SDN controller is a good method to achieve intelligent routing for SD-DCN. With the design of a properly structured DRL, it can automatically derive the critical traffic patterns from data traces and learn the underlying mapping between the traffic patterns and routing path configurations.

Therefore, this paper proposes deep reinforcement learning-based routing (DRL-R) for SD-DCN. The main contributions of this paper are as follows:

(a) We propose a DRL-based intelligent routing scheme for SD-DCN. A DRL agent deployed on an SDN controller continually interacts (learns) with the network, from a global view, to adaptively make reasonable routing decisions according to the network state and to optimally allocate the cache and bandwidth for traffic in a coordinated manner. Differently from other DRL-based routing schemes, this paper proposes a novel DRL formulation representation for the DRL-based routing scheme where DRL-R uses distinct image with the pixel of resource-recombined to represent DRL's state and uses an action representation at path level.

(b) We propose a novel method that recombines multiple network resources with different metrics to a quantifiable and additive resource-recombined. Differently from other recombining methods, we recombine bandwidth and cache by quantifying their contribution scores of benefitting the delay reduction, i.e., we establish the mapping between performance and resource. The resource-recombined can be used as the unit of scheduling and allocating resources.

(c) We propose a routing scheme with resource-recombined state. Based on this abovementioned recombination, we change the routing metric from the single link state to the resource-recombined state. Differently from other routing schemes, DRL-R, except to the bandwidth, lets cache to be the one factor of effecting routing decision.

(d) We demonstrate the effectiveness of DRL-R solution in terms of lower FCT and higher throughput and better load balance as well as better robustness, compared to OSPF and TIDE (Sun et al., 2019), through extensive simulations.

The remainder of this paper is organized as follows: related work is presented in section II; section III introduces some background; section IV presents the details of DRL-R system; in section V we show experimental results, and we discuss related issues in section VI; section VII concludes the paper.

2. Related work

2.1. Deep learning for network

Deep learning has been applied to some networking contexts (Yao et al., 2018; Streiffer et al., 2018; Wang Cuiet al., 2018). Aibert Mestres et al. (2017) described the knowledge-defined networking (KDN) paradigm that accommodates and exploits SDN, network analytics and AI, and they presented a use-case of KDN for routing.

Fengxiao Tang et al. (2018) used DL to propose an intelligent network traffic control method, exploiting deep convolutional neural networks with uniquely characterized inputs and outputs for representing the considered Wireless Mesh Network backbone. Several attempts of using DL (Dong et al., 2015; Kong Zanget al., 2018) have been made to optimize the TCP congestion control algorithm due to the difficulty of designing a congestion control algorithm that can fit all the network states.

Shiqiang Wang et al. (2018c) considered the problem of DL model parameters from data distributed across multiple edge nodes, without sending raw data to a centralized place. Their focus was on a generic class of DL models that are trained using gradient-descent based approaches. Sandeep Chinchali et al. (2018) present a RL based method to optimally schedule IoT traffic, which can dynamically adapt to traffic variation.

To deep learning for network resource allocation, Hao Ye et al. (Ye and Li, 2018) developed a decentralized allocation scheme for vehicle-to-vehicle (V2V) communication systems based on DRL. Each V2V link is an agent that makes its own decisions to find the optimal sub-band and power level for transmission. DRL has been also used to address the problem of job-scheduling with multiple resource demands (Mao et al., 2016; Maoet al., 2019), where the objective is to minimize the average job slowdown and the reward function is based on the

reciprocal duration of the job. Yong Cui et al. (Wang et al., 2018b) summarized the basic workflow of applying the machine learning in the networking domain.

2.2. Deep reinforcement learning for routing

Bomin Mao et al. (2017) explored new opportunities in packet processing with deep learning for inexpensively shifting the computing needs from rule-based route computation to deep learning based route estimation. Each node trains a DL model, and its input is the real-time packet arrival rate. Owing to this complex process, the delay of decision cannot handle the flow-level routing at the scale of current DCN, because mice flows are usually gone before decisions can be made. Besides, because the DL model is trained according to the pre-established network topology, the DL model must be retrained when the network topology changes. This indicates that the scheme lack scalability. Thomas et al. (Pasca et al., 2017) proposed an application-aware multipath flow routing framework that integrates Machine Learning (ML) techniques in an SDN, where its controller is capable of prioritizing each of the flow by using ML, and assigns a path based on its classified priority.

In short, these works (Mao et al., 2017; Pasca et al., 2017) all use the DL-based method, which is a model-based predictive control method relying on accurate and mathematically solvable system models (such as queueing models). Differently from them, our DRL-based method is model-free, thereby enhancing its applicability in complex networks with random and unpredictable behaviors.

Asaf Valadarsky et al. (2017) initiated the study of data-driven routing, where RL extract information from the history of traffic scenarios to generate routing. QoS routing (Lin et al., 2016) build a three-levels SDN control plane, they use table-based RL agents, which fill a table [*state* x *action*] →*reward* used in operation during training. As a hybrid routing algorithm to ad hoc network, Q^2-Routing (Hendriks et al., 2018) make routing decisions by choosing the neighbor associated with the optimal Q-value for a given destination as the next hop, similar to Q-Routing (Boyan et al., 1994). At the same time, Q^2-Routing adds an efficient exploration strategy to support QoS.

In short, these works (Valadarsky et al., 2017; Lin et al., 2016; Hendriks et al., 2018; Boyan et al., 1994) all use the RL-based method. However, as shown in section 3.2.1, there are lots of possible {*s*, *a*} pairs in a slightly larger DCN, RL is impossible to store and find the optimal policy in tabular form in reasonable time. Our DRL-based method is capable of handling a dynamic and sophisticated state space.

Another method is DRL-based. By combining Q-routing and Deep Q-Network (DQN) (Mnih et al., 2015), multi-agent learning for routing is proposed (Mukhutdinov et al., 2019), where every router in the network is treated as a single agent that can observe only the local environment information and take actions according to its own routing policy. However, operating at per packet level, and in a decentralized fashion, poses significant challenges in terms of scalability and communication overhead. Further, at flow level, Zhiyuan Xu et al. (2018) designed actor-critic-based prioritized experience replay and TE-aware exploration to optimize the DRL for Traffic Engineering. Kai Chen et al. (2018) developed a two-level deep reinforcement learning system, AuTO, to achieve traffic optimization (TO). Where peripheral systems (PS) on end-hosts collect flow information and use multi-level feedback queueing to make TO decisions locally with minimal delay for mice flows. Central system (CS) can aggregate and process global traffic information by obtaining the PS's decisions. CS used DDPG (Lillicrap et al., 2015) to optimize and set the parameter of PS. CS further makes individual TO decisions for elephant flows by using Policy Gradient (PG) algorithm. AuTO adopts the *big-switch* assumption where the network is non-blocking with full bisection bandwidth and proper load balancing. However, this assumption is not always true.

Jose Suarez-Varela et al. (Mestres et al., 2017) argued that successfully applying DRL to routing requires a good representation of the network parameters. However, the past proposals (Valadarsky et al., 2017; Sun et al., 2019) using straightforward representations for both the state and the action space. For example, in TIDE (Sun et al., 2019), the action of the DRL agent is to find a tuple of optimal link weight with which shortest path algorithm (such as, Floyd-Warshall algorithm for OSPF-like protocol) is used to calculate the routing path. However, their actions depend on modifying link weights, the optimal routing path can only be obtained indirectly by shortest path algorithms.

In short, as a DRL-based routing method, DRL-R proposes a novel representation of DRL formulation (reward, action and state). Specially, differently from (Xu et al., 2018; Chen et al., 2018; Mukhutdinov et al., 2019; Lillicrap et al., 2015), DRL-R uses the distinct image to represent DRL's state where the resource-recombined is the pixel of image. Differently from (Valadarsky et al., 2017; Sun et al., 2019; Mestres et al., 2017), independent from the shortest path algorithm, DRL-R uses an action representation at path level where each action corresponds to select a specific end-to-end path.

At the path level for the action, Jose Suarez-Varela et al. (2019) used a statistics at the path level to represent the network state. They claim that this method can capture both the overall network utilization and the critical inter-dependencies among the paths resulting from the network topology. Qiongxiao Fu (Fuet al., 2020) also uses deep Q-learning to generate optimal routing paths for SD-DCN where the executable actions are in the alternative path set between the source and destination servers of the flow. Our previous work (Liu, 2019) introduce the basic architecture of DRL-R. By extending (Liu, 2019), this paper furtherly confirms the effectives of DRL-R by the extensive theoretical analysis and simulation.

Differently from all the past works, most important, DRL-R is a routing scheme with resource-recombined state instead of single link state. DRL-R first recombines bandwidth and cache, and then uses the resource-recombined as a measurement of routing paths' quality. In other words, except to the bandwidth, DRL-R lets cache to be the one factor of effecting routing decision. From the perspective of jointly optimal allocating the bandwidth and cache for traffic, DRL-R uses DRL to optimize routing decision in a coordinated manner for traffic variations across time and space.

In summary, the proposed DRL-R can overcome the shortcomings mentioned above and has some differences from the previous routing schemes. Firstly, DRL-R is a DRL-based routing scheme with resource-recombined state. Secondly, this paper employs DRL with novel representation of DRL formulation (reward, action and state).

3. Background

3.1. Overview of multisource transmission

We previously proposed information-centric networking with built-in network coding (ICN-NC) (Liu et al., 2017), where in-network caching is collaborated with network coding to achieve *multisource*

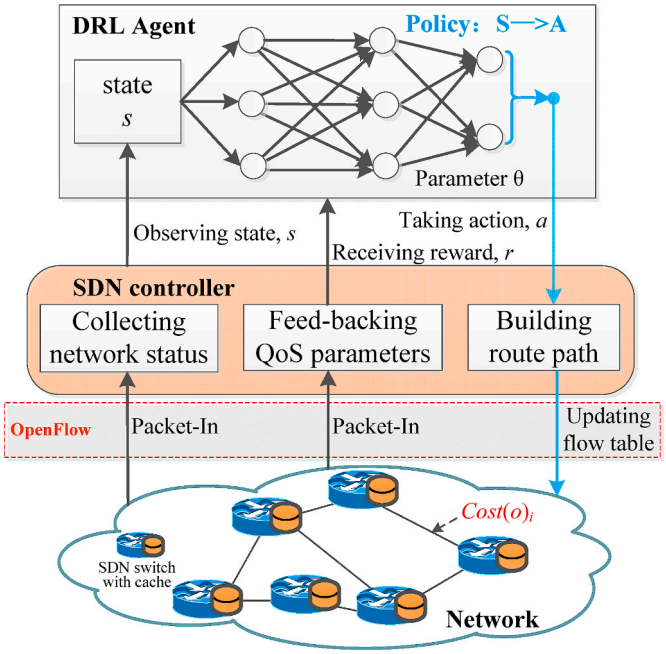


Fig. 1. System architecture of DRL-R.

transmission at the network-layer. The basic idea behind this *multisource transmission* is that, as long as the user can obtain the specified number of linear independent CMs (Coded Message with network coding) from single or multiple data sources (original content server or intermediate nodes) through single or multiple paths, it can decode them and further can complete the data transmission. Thus, how to find a good path for the data transmission becomes an important problem that is also the primary goal of DRL-R.

3.2. Deep reinforcement learning

The section overviews two typical DRL algorithms: Deep Q-Network (DQN) (Mnih et al., 2013, 2015) and Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015).

3.2.1. Reinforcement learning

Let us briefly review typical reinforcement learning algorithm—Q-learning before introducing DRL (Li, 2017). As shown in Fig. 1, an agent interacts with the environment. At each timestep t, the agent can observe a state s_t, and select an action a_t. Following this action, the environment's state transits to s_{t+1} and the agent receives a reward r_t. The state transitions and rewards are stochastic, and follow the markov decision process (MDP); *i.e.* the state transition probabilities and rewards depend only on the state of the environment s_t and the action taken by the agent a_t.

Note that the agent can only control its own actions and has no a priori knowledge of which state the environment would transit to or what the reward may be. As shown in Fig. 1, the agent takes actions based on a *policy* $\pi(a|s)$ which is the probability that the action a is taken in state s. Actually, it is a mapping from the state space to the action space, S - $>A$.

In summary, the goal of RL continuous learning is to maximize the expected cumulative discounted rewards$Q^{\pi}(s, a)$, i.e., obtain the following $Q^{*}(s,a)$,

$$Q^{*}(s,a) = \max Q^{\pi}(s,a) \tag{1}$$

where

$$Q^{\pi}(s,a) = E[R_t | s_t = s, a_t = a, \pi] \tag{2}$$

where

$$R_t = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3}$$

where $\gamma \in (0,1]$ is a discount factor for future rewards.

In most practical problems, there are lots of possible $\{s, a\}$ pairs, and up to $\infty$for the problem in this paper (see §4.1). Hence, RL is impossible to store and find the optimal policy in tabular form in reasonable time. Thus, DRL has been developed to overcome the shortcomings, and it is common to use *function approximator*. A *function approximator* has an adjustable parameter θ and then the *policy* is represented as $\pi(a|s, \theta)$. DRL is capable of handling a sophisticated state space.

3.2.2. Deep Q-Network (DQN)

DQN combines Q-learning with deep learning (Mnih et al., 2013, 2015). The basic idea behind DQN is using a deep neural network (DNN) as a *function approximator* with weights θ for the Q-network.

The Q-network updates its weights, θ, at each iteration to minimize the following loss function ($Loss(\theta)$) derived from the same Q-network with old weights,

$$Loss\left(\theta\right) = \sum\left(Y - Q(s_t, a_t, \theta)\right)^2 \tag{4}$$

Where

$$Y = r_t + \max_{a \in A} Q_{old}(s_t, a, \theta) \tag{5}$$

Compared to Q-learning, except for using DNN as the *function approximator*, DQN applies two key modifications to improve performance.

Firstly, DQN employed *experience replay* to surpass the difficulty of the training to converge in RL, where the samples are highly correlated and non-stationary. In *experience replay*, the agent's experiences at each timestep are stored, e_t (s_t, a_t, r_t, s_{t+1}), in $D_t = \{e_1, \ldots, e_t\}$, pooled over many episodes into an experience replay memory. Differently from Q-learning that uses only immediately collected samples, the DQN agent randomly takes sampling from the experience replay memory to break the correlation between sequentially generated samples, and thus can learn from more independently and identically distributed past experiences.

Secondly, DQN's modification to Q-learning, aimed at further improving the stability, is that using a separate *target network* $\widehat{Q}$ for generating the targets in the Q-learning update. Its parameters slowly update with the DQN weights every $C > 1$ steps (please see the Algorithm 1).

3.2.3. Deep deterministic policy gradient (DDPG)

DQN indirectly seeks the policy. Why we do not directly search for the optimal policy? This idea is the policy gradient method, and DDPG is the typical one among them. DDPG (Lillicrap et al., 2015) lets policy $\pi(a|s)$ to be approximated with an adjustable parameters θ (referred as $\pi(a|s, \theta)$). In DDPG, we calculate the action policy gradient and progressively adjust the action a following this gradient, and then gradually obtain the optimal policy. In other words, actually, DDPG adjusts θ of $\pi(a|s, \theta)$ according to the direction in which the Q value is the largest. Besides, DDPG is feasible in complex and large action spaces by avoiding the optimization of action at each step to obtain a greedy policy as in DQN. In summary, DDPG has the following three improvements compared to DQN.

Firstly, DDPG continues to employ the *experience replay* and *target network*. However, DDPG employs critic target network and actor target network separately, and updates them every smaller than C steps to further improve the stability. Secondly, adding *noise sample*: One difficulty of RL learning in continuous action space is the exploration of action. DDPG addresses this problem by adding *noise sample* based on action. Thirdly, compared to deterministic policy gradient (DPG), the most critical improvement of DDPG is using a convolutional neural network (CNN) as the approximator of *value function* Q and *policy* π. Further, they are trained by deep learning, where its activation function is the rectified non-linearity.

4. DRL-R design

4.1. Problem formulation

We assume that the SD-DCN network $\boldsymbol{w}$ has V switches and L links. This paper assumes that every switch (*i.e.*, node) in SD-DCN is equipped with some cache space, and they can cache the passing content. This paper uses the simplest cache everything everywhere (CEE) as the caching policy, and widely used least recently used (LRU) as the cache replacement policy (Cui et al., 2017). Following the *multisource transmission* in ICN-NC (Liu et al., 2017), in this paper, it is CMs instead of original content that the switch forward and cache.

Table 1
Notation acronyms list.

Notations	Definition
r	The reward for DRL
s	The state for DRL
a	The action for DRL
RR	Resource-recombined
L_Cost_i	The contribution score of the link's bandwidth between node i and its downstream node reducing delay
$C\_Cost_i(o)$	The contribution score of node's cache reducing delay for content o
$Cost(o)_i$	RR of node i for content o
$Cost$	Similar to OSPF, the metric of measuring link's quality

In the following parts of this paper, unless otherwise specified, three different network scales (one node, one path, and one area) have the following meanings as separately: one node consists of the node and the corresponding link between it and its downstream node,[1] one path is formed of multiple nodes, and one area is formed of multiple paths. At the same time, the multiple network resources in the rest of this paper refer to cache and bandwidth.

Because bandwidth and cache coexist in DCN, we let bandwidth and cache to be recombined as the resource-recombined (RR, please see section 4.5) which can be used as the unit of scheduling resource. Further, as presented in 4.5.2, the RR of one node ($Cost(o)_i$), the RR of one path and the RR of one area can separately reflect the performance of one node, one path and one area. More importantly, they are comparable and additive, thus a RR can be used as a metric of comparing performance of routing paths' quality and as a resource allocation metric to improve resource utilization. In this paper, we let $Cost(o)_i$ to the $Cost$ of the link between node i and its downstream node for routing content o. While choosing a routing path, the higher the sum of all links' $Cost$ of building a path is, and the better this path is.

In the process of recombining multiple network resources as mentioned later, we quantify the contribution score of network resources (cache and bandwidth) benefitting for meeting performance requirement (reduction of delay), that is, we establish the *mapping between performance and resource*. In this manner, we convert the performance requirements of flow into resources demands of the flow. Thus, if the performance requirement of each flow is known, the resource demand of each flow is also known. For example, the delay requirement of the path passed through by a flow can be described as the following: demanding x units of $Cost$. Due to dense links in DCN, there are multiple paths for a flow between the source and the destination. Further, if employing a multipath routing scheme, we can decompose these x units of $Cost$ into multiple paths to complete.

The bandwidth and cache is two different metric. The core idea of recombining multi-resource is that we use contribution score of bandwidth and cache reducing delay as the measures of conversion between them. Normally, the bandwidth and cache affect the *transmission delay* and *propagation delay* respectively. Thus, we quantify the contribution score of bandwidth and cache reducing delay respectively, and then recombine them (more details are shown in section 4.5). We consider that the packet of a flow arrives at a node in discrete timesteps. The DRL-R scheduler chooses one or more of waiting flows for scheduling at each timestep. We also assume that the performance requirement of each flow is known, which is classified into H levels. Owing to the above-mentioned establishment of *mapping between performance and resource*, in other words, we actually assume that the resource demand of each flow is known upon arrival. More specifically, the resource profile of flow requiring resource is given. For simplicity, we assume no preemption and a fixed allocation profile.

DRL-R dedicates itself to find a good path for the data transmission.

[1] Downstream node refer to the next neighbor node on the forward path.

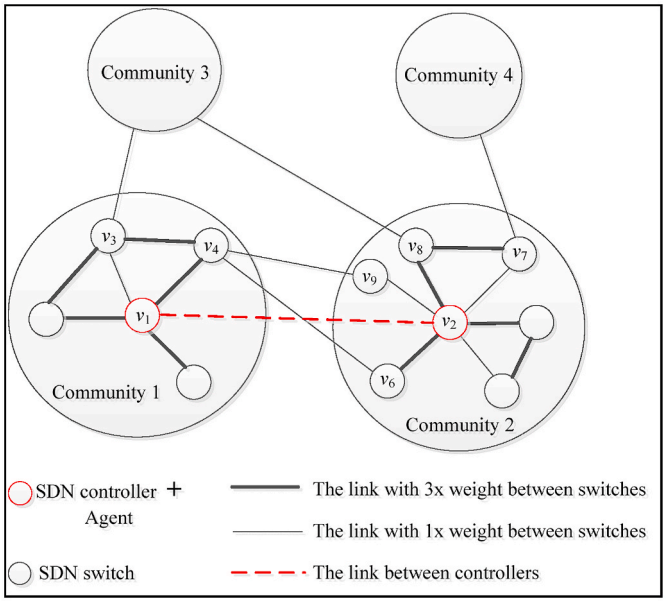


Fig. 2. Weighted community.

Thus, in a DCN with multiple network resources, the essence of the routing scheme become into properly allocating resources to meet the requirements of various type of flows. For example, selecting a path on which its links have some appropriate bandwidth and on which its nodes have some appropriate cache.

In summary, the most critical idea behind DRL-R is converting the performance requirements of a flow into its network resources requirements. So the routing problem can be converted into a job-scheduling problem in resource management, where this job is the flow waiting to be forwarded in this paper.

The key problem addressed this paper is whether deep reinforcement learning can efficiently allocate cache and bandwidth to improve the routing performance in SD-DCN, and how much. The first objective of the proposed DRL-R is maximizing the overall throughput of network when the FCT of two types of flow (i.e., mice flow and elephant flow) meets its own QoS requirement. Some main notation acronyms are shown in Table 1.

4.2. System overview of DRL-R

As shown in Fig. 1, the framework of DRL-R is composed of three planes, which is built upon a typical SDN network architecture. With the southbound interface (i.e., OpenFlow in this paper), the controller collects the network status (i.e., resource allocation state and resource demand state in this paper). With the northbound interface, the controller sends the global view of the network status as a *state* input to the DRL agent. Then, the DRL agent can output an *action* (i.e., a subset of nodes from the network, in this paper) to the controller. Based on this, the controller builds a route path and converts it to OpenFlow's flow tables that are installed on corresponding switches. In addition, the controller collects some quality of service (QoS) parameters (i.e., the FCT of each flow and the throughput of node, in this paper) which are used as the *reward* of the DRL agent to evaluate the performance of routing strategies. In each interaction with the network, the DRL agent adjusts the parameters to obtain a higher reward. After a period of training, the DRL agent can learn enough experience knowledge from the interaction with the network and produce a near-optimal routing strategy for the network.

4.3. Workflow of DRL-R

This section outlines the basic workflow of DRL-R in three steps, as the follows:

(1) Divide network domain in a community-based manner

Filiposka S et al. (Filiposka and Juiz, 2015) has shown that a community[2]-based data center can efficiently achieve optimal resource management, such as capacity allocation, and load balancing. Our previous work also confirmed that a community-based switch--to-controller mapping scheme (AAMcon) (Liu et al., 2020) presents good performance. AAMcon builds a community where the controller is placed at the most important node (with the highest betweenness) in this community, and it becomes the first mapping choice of the switch that needs support from the controller. The controller can respond to the switch with the least distance to minimize the communication time between the controller and switches. Simultaneously, AAMcon can achieve load balancing between controllers and avoid congestion.

Thus, in a community-based SD-DCN, the network is divided into some communities, one of them being one network domain where there is one controller to manage the intra-domain communication. As shown by the red dotted line in Fig. 2, these controllers are interconnected and form a flat distributed network of controllers to manage the inter-domain communication. In this paper, DRL-R aims for the intra-domain communication. We employ one DRL agent on each SDN controller, as shown by v_1 and v_2 in Fig. 2. Such a deployment can implement the network state upload and the decision download respectively with an aim to achieve a centrally control, from a global view, for a large-scale network.

(2) Build a network with resource-recombined

According to the global network topology, an SDN controller builds a network $\boldsymbol{w}$ with resource-recombined, where the *Cost* of the link between node i and its downstream node is represented as $Cost(o)_i$, as shown in Fig. 1. $Cost(o)_i$ covers the cache of node i, and the bandwidth of the link between node i and its downstream node.

(3) Build the DRL framework for routing scheme

This paper employs two DRL algorithms (DQN and DDPG) for building DRL-R, are abbreviated as DRL-R-DQN and DRL-R-DDPG respectively. DRL's formulation is presented in section 4.4.

On one hand, the agent interacts with the network based on RL, which include collecting *state* from the network $\boldsymbol{w}$, taking an *action* to $\boldsymbol{w}$, and finally feed backing *reward* from $\boldsymbol{w}$. Actually, the SDN controller uses OpenFlow's flow table to create the routing path and forwarding rules for switch according to this *action* taken by the agent. Without loss of generality, this paper uses flow as the granularity of forwarding.

On the other hand, the agent uses CNN to seek the mapping from state space S to action space A. In other words, after multiple interactions between the agent and the network $\boldsymbol{w}$, it can find the optimal action set that implements the optimal routing goal, i.e., *policy*.

DRL-R-DQN's detailed algorithm is shown as **Algorithm 1**. DRL-R-DDPG's detailed algorithm is shown as **Algorithm 2**, where DRL-R-DDPG constructed an action exploration policy by adding noise sampled from a noise process for the actor network to avoid the

[2] The appendix introduce more detail about community theory.

suboptimal result in the process of action exploration. In other word, after the actor network outputs an action, it is added N_t that is a temporally correlated value centered around 0 generated by the *Ornstein-Uhlenbeck Process*.

Algorithm 1
DRL-R-DQN

Input: hyper parameters of neural network, discount factor γ;
Output: at timestep t, the action a_t as the routing strategy parameter to the SDN controller;
1:Divide network domain in a community-based manner;
2:Build a network with resource-recombined;
3:Build the DRL framework for routing scheme;
4: Initialize experience replay memory D;
5: Initialize action-value function Q with random weights θ ;
6: Initialize target action-value function $\hat{Q}$ with weights $\theta^{-}=\theta$;
7: **For** episodes =1:M do
8: Initialize sequence $s_1\sim\{x_1\}$ and preprocessed sequence $\Phi_1=\Phi(s_1)$;
9: **For** timestep t=1:T do
10:With the probability ε select one random action a_t, otherwise select $a_t=\text{argmax}_a\, Q(\Phi(s_t), a; \theta)$; **Action selection**
11:Execute action a_t
12:Output the a_t as the routing strategy parameter to the SDN controller;
13:The SDN controller builds a route path and converts them to flow tables that are installed on corresponding switches;
14:Observe reward r_t;
15:Observe image x_{t+1}; // Collect the network state as the *state*.
16:Set $s_{t+1}=s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1}=\Phi(s_{t+1})$;
17:Store transition $(\Phi_t, a_t, r_t, \Phi_{t+1})$ in D; ***Experience replay***
18:Sample random mini-batch of transitions $(\Phi_j, a_j, r_j, \Phi_{j+1})$ from D;
19:Set $y_j =$

$$\begin{cases} r_j, & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\Phi_{j+1}, a'; \theta), & \text{otherwise} \end{cases}$$

Target network
20:Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j, \theta))^2$ with respect to the network parameters θ ;
21:Reset $\hat{Q} = Q$ every C steps;
22:**End For**
23:**End For**

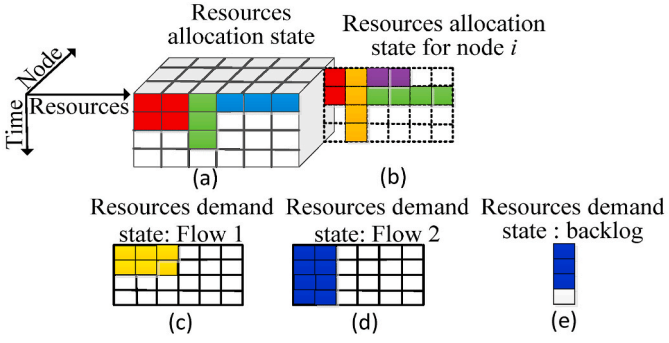


Fig. 3. An example of a state representation.

Algorithm 2
DRL-R-DDPG

Input: hyper parameters of neural network, discount factor γ, target smooth factor τ;
Output: at timestep t, the action a_t as the routing strategy parameter to the SDN controller;
1:Divide network domain in a community-based manner;
2:Build a network with resource-recombined;
3:Build the DRL framework for routing scheme;
4: Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor network $u(s|\theta^Q)$ with weights θ^Q;
5: Initialize Critic target network Q' and Actor target network u' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{u'} \leftarrow \theta^u$ separately;
6: Initialize experience replay memory D ;
7: **For** episodes = 1: M do
8: Initialize a random process N for action exploration;
9: Receive initial observation state s_1 ;
10:**For** timestep t = 1: T do **Noise**
11:Select action $a_t=u(s_t|\theta^u)+N_t$ according to the current policy and exploration noise;
12:Execute action a_t ;
13:Output the a_t as the routing strategy parameter to the SDN controller;
14:The SDN controller builds a route path and converts them to flow tables that are installed on corresponding switches;
15:Observe reward r_t ;
16:Observe new state s_{t+1}; // Collect the network state as the *state*.
17:Store transition (s_t, a_t, r_t, s_{t+1}) in D; ***Experience replay***
18:Sample a random mini-batch of N transitions (s_i, a_i, r_i, s_{i+1}) from D;
19:Set $y_i = r_i + \gamma Q'(s_{i+1}, u'(s_{i+1}|\theta^{u'})|\theta^{Q'})$;
//Update Critic network
20:Update critic by minimizing the loss:
$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
//Update Actor network
21:Update the actor policy using the sampled gradient:

$$\nabla_{\theta^u} u|s_t \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=u(s_t)} \nabla_{\theta^u} u(s|\theta^u)|s_t$$

22:Update Critic target network & Actor target network

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{u'} \leftarrow \tau\theta^u + (1-\tau)\theta^{u'}$$

23:**End For**
24:**End For**

4.4. Deep reinforcement learning formulation for DRL-R

Designing the reward function, state space and action space is critical to the success of a DRL method. Thus, our following design well captures the key components of the routing problem.

(1) Reward r

We craft the reward signal to guide the agent towards good solutions for our objective: maximizing the overall throughput of network when the FCT of two types of flow meet its own QoS requirement. The agent does not receive any reward for intermediate decisions during a time-step. Specifically, r_i is the reward at timestep i and it is set as,

$$r_i = \begin{cases} \sum_{j \in N} T_{ij} & F_{1i} < D_{Qos1} \\ & F_{2i} < D_{Qos2} \\ R & Otherwise \end{cases} \quad (6)$$

where N is the total number of network node, T_{ij} is the throughput of node j at timestep i, and R is a constant, $R < 0$. F_{1i} and F_{2i} are the average of FCT for the finished mice flow and elephant flow, respectively during timestep (i-1). D_{Qos1} and D_{Qos2} is a constant, $D_{Qos2} > D_{Qos1} > 0$. $F_{1i} < D_{Qos1}$ indicates that the FCT of mice flow meets its deadline, and $F_{2i} < D_{Qos2}$ indicates that the FCT of elephant flow satisfies its QoS requirement. Because the FCT could be longer than one timestep, we cannot immediately obtain the FCT of the flow taken by an action. Therefore, equivalently, we compute the average FCT of all finished flows during the previous one timestep. D_{Qos1} and D_{Qos2} can be determined according to the real traffic. Without losing generality, this paper sets D_{Qos1} as 20 ms and D_{Qos2} as 6s according to the experimental traffic.

(2) State s

State s includes resource allocation state and resource demand state: resource allocation state indicates how much resources of a node have been used, and resource demand state indicates how much resources are required by a flow waiting to be scheduled. We represent the state of DRL—the current resource allocation state and the resource demand state of flows waiting to be scheduled—as distinct *images* (see Fig. 3 for illustration, where different colors represent different flows (Mao et al., 2016)).

As shown in Fig. 3(a), the resource allocation state image is three-dimensional (node, *RR*, and time). As an example of resource allocation state image, Fig. 3(b) presents the resource allocation state of node *i*. Where the red, orange, purple, and green colors represent resource allocation state of four flows separately; for example, the red flow occupies 1 unit of *RR* and 2 units of time.

The resource demand state image is two-dimensional (*RR* and time), that represent the resource demands of awaiting flows. For example, Fig. 3(c) and (d) represent resource demand state of flow 1 and flow 2 respectively. Where this *RR* is the sum of flow requiring *RR* for the whole path passed-through from the source to the destination. For instance, the blue (flow 2) requires 2 units of *RR* and 4 units of time on its path.

Because the state (image) is the input of CNN which requires fixed input size, the input size of CNN is depend on the size of image and the number of image. On one hand, the size of image is depend on the period of recycling resource and the number of segmenting resources. Without losing generality, this paper sets this period as 10, and sets the number of segmenting resources as 10. On the other hand, the number of image for the resource allocation is depend on the node's number that is usually fixed. The number of image for the resource demand is depend on the number of flow waiting to be scheduled. Thus, we maintain images for only the first F flows to arrive (which have not been scheduled yet). As shown in Fig. 3(e), the information about any flows beyond the first F is summarized in the *backlog* component of the state, which simply counts the number of such flows. Actually, this approach has also the added advantage of constraining the action space that makes the learning process more efficient.

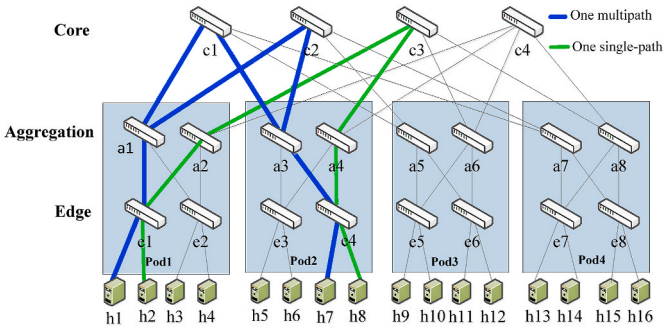


Fig. 4. Building one single-path or multipath for Fat-tree with K = 4.

Real network may suffer from some types of failures such as node failure, link failure, interface lockout, transmission error, and packet drop due to congestion etc. In order to research how these failures affect routing robustness, we assume that each link has a certain failure rate. For simplification, we let its corresponding node's resource allocation state to be occupied with 1 unit of RR and z units of time when the link is interrupted in the experiment. The z is determined by how long the link is interrupted.

(3) Action a

In this paper, taking an action a for a flow is to select a subset of nodes[3] from the network w. In other words, the action is the list of the nodes and theirs corresponding links which build a single-path or a multipath to deliver traffic from the source to the destination. Thus, this is factually an action representation at path level where each action corresponds to select a specific end-to-end path.

At each point in time, the scheduler may want to admit any subset of the F flows and this would require an action space of size 2^F. Further, action a for each flow means selecting a subset of nodes from the network w. In order to prevent the message from running around the network, we limit the hops of routing path to H_1 and prohibit loop of routing simultaneously. Thus, to a network with V nodes, the maximum number of paths for each flow is (V-1) (V-2) …. (V–H_1). Thus, the total action space has size (V-1) (V-2) …. (V–H_1)2^F. However, this could make DRL-R's learning very challenging. To limit the size of the action space, we set $F = 4$, and we can also set H_1 as a small value. H_1 can be determined according to the topology in DCN. For example, $H_1 = 4$ in a K-pod fat-tree topology, whatever the value of K.

Under a K-pod fat-tree with K = 4, Fig. 4 presents an example of taking an action to build one single-path (multipath) routing for a flow between the pair of $h2$ and $h8$ (the pair of $h1$ and $h7$).

4.5. Recombining multi-resource

This paper abstracts bandwidth and cache as a quantifiable and additive resource-recombined (RR) by mining the common characteristics of them reducing delay.

4.5.1. Quantifying the score of bandwidth and cache

Evidently, shorter physical distance and larger bandwidth of a link can produce less delay. In fact, the physical distances of links in DCN are short, thus the bandwidth is the main influencing factor. Thus, considering that a user requests content o, we extract the contribution score of the link's bandwidth between node i and its downstream node reducing delay as a function of its bandwidth,

$$L\_Cost_i = \frac{\alpha_1}{B_i} \quad (7)$$

where B_i is the bandwidth of the link from node i to downstream node, and its unit is *Mbps*. The α_1 is the adjustable factor. Following the principle of OSPF setting its cost, we can set $\alpha_1 = 1000$ because the links

[3] In this paper, one node is consist with the node and the corresponding link between it and its downstream node.

bandwidth in modern DCN are generally higher than 1 Gbps.

When a user wants to obtain the content o, the basic idea behind *multisource transmission* in ICN-NC (Liu et al., 2017) is the following: the user can complete this transmission task as long as it can obtain specified number (referred as $E(o)$) of linear independent CMs of content o from one node, one path, and one area. Thus, as the basic unit among them, the more CMs one node has, the less the time of completing transmission task. Therefore, there is a mapping relationship between the proportion of the CMs' number of content o on a node with the specified number and reducing delay. More importantly, because the additivity of this proportion causes the cache to have a similar additivity with bandwidth, it is possible to execute addition and compare size between the cache of one node, one path, and one area.

Therefore, we extract the contribution score of node's cache reducing delay as a function,

$$C\_Cost_i(o) = \frac{M_i(o)}{E(o)} \tag{8}$$

where $M_i(o)$ is the number of node i caching CMs of content o. $E(o)$ is the specified number of CMs, which is a constant and also called as *generation size* in random linear network coding (Liu et al., 2017), to decode CMs to obtain the content o. Normally, $E(o)$ is set as 8 (Liu et al., 2017).

4.5.2. Resource recombined of different network scale

When selecting the routing path, there are two reasonable rules: the higher links' bandwidth is, and the better the path; similarly, the higher node' cache is, and the better the path. Specially, in this paper, while choosing a routing path, the higher the sum of *Cost* ($Cost(o)_i$) of all link built the path is, and the better the path.

After the score of bandwidth and cache are quantified respectively, it is comparable between the size of cache and bandwidth. Then, they can be unified abstracted into *RR* of three network scales, i.e., *RR* of a node, *RR* of a path and *RR* of an area. Thus, the specific process is described as follows.

When we recombine the cache of node i with the bandwidth of link between it and its downstream node for content o, we refer to this as *RR* of node i for content o ($Cost(o)_i$). To support the mentioned above rule, it is defined as the following,

$$Cost(o)_i = \frac{\beta^* C\_Cost_i(o)}{\alpha^* L\_Cost_i} \tag{9}$$

where, α and β are the weight of bandwidth and cache mapped to delay separately. Without loss of generality, $\alpha = 1$ and $\beta = 1$.

We assume that the path j includes k nodes, and the area v includes g paths. Thus, the *RR* of path j is the sum of *RR* of k nodes, and it is defined as the following,

$$Cost_j(o)_{path} = \sum_{i=1}^{k} Cost(o)_i \tag{10}$$

If there are g paths doing *multisource transmission* in area v, the *RR* of area v is the sum of g paths' *RR*, and it is defined as the following,

$$Cost_v(o)_{area} = \sum_{j=1}^{g} Cost_j(o)_{path} = \sum_{j=1}^{g}\sum_{i=1}^{k} Cost(o)_i \tag{11}$$

After the multiple network resources are recombined, it can be used as the unit of scheduling resource. As a use case, this paper lets $Cost(o)_i$ to be the *Cost* of link.

Table 2
Parameters of experiments.

Parameters	Default value	Range
Training steps	50 k	2 k-100 k
Episodes	100	
Relative cache size for node's cache capacity	1%	0.4%–20%
Link capacity	2 Gbps	1-10Gbps
Traffic intensity	50%	10%–120%
Probability of link failure	0%	0%–50%
Interaction period/signaling interval	3s	4s–1s

5. Simulation experiments

5.1. Experiment scenarios and setup

5.1.1. Methodology

To assess the performance of the proposed DRL-R, we generate a Fat-tree topology with $K = 8$, where nodes have differential cache capacities and links have uniform bandwidth. We highlight that this scale of simulation is sufficient enough to demonstrate that the proposed DRL based routing scheme outperforms the conventional routing strategies such as OSPF. Our simulation is conducted on a workstation with a six-core i7 3.3 GHz processor and 16 GB RAM.

We use OMNet++ as the simulation framework. We use 10 traffic intensity (TI) levels, ranging from 10% to 120% of the total network capacity. For each TI, we generate 100 traffic matrix (TMs, being the *RR* request between each source-destination pair) of equal total traffic using a gravity model (Roughan, 2005). Therefore, there are 1000 distinct traffic configurations varying over both the total volume of traffic and its distribution.

As a typical combination of parameters, for DRL-R-DQN, this paper sets its *function approximator* to be a DNN that consists of 2 fully connected hidden layers with 30 and 30 neurons respectively. For DRL-R-DDPG, the CNN in critic network as well as in actor network consists of 2 max pooling layer, 3 convolutional layers with 8 filters and 1 fully connected layer with 30 neurons. Their activation functions all are the rectified linear unit (ReLU). We update the *policy network* parameters of actor network and critic network with a learning rate of 0.0001 and 0.001 respectively. The target smooth factor is 0.001, and the batch size is 32. Besides, we selected a value of 0.99 for the *discount factor* (γ) and 0.8 for the *exploration parameter* (λ), which can obtains the best results. Table 2 shows other key parameters used in our experiments and their values.

To evaluate the performance of DRL-R, it is reasonable to choose some existing routing solutions as the compared benchmark in the simulation. This paper selects OSPF with Dijkstra algorithm and the latest routing scheme DRL-based TIDE (Sun et al., 2019) as the compared benchmark. The former is a widely used routing solution and the latter is the latest routing scheme DRL-based.

5.1.2. Performance measures of evaluation

(1) FCT

The *FCT* is the average of flow completion time for all flows.

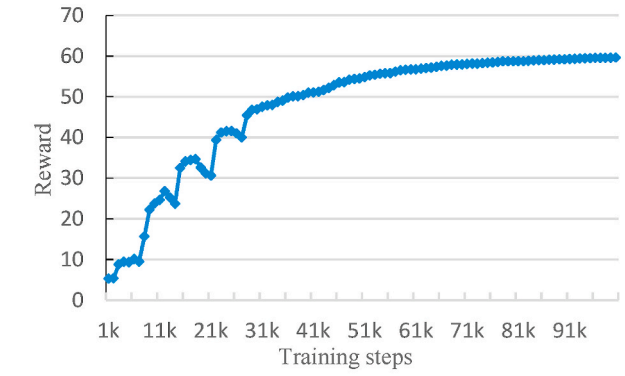
(2) Throughput (*throughput_mean*)

The *throughput_mean* is the average of the throughput of all nodes.

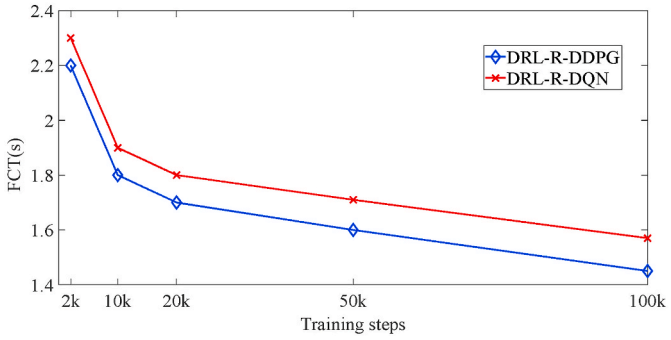
(3) Balance of link load (*Load_G*)

In this paper, we introduce the *Gini coefficient* of link load (*Load_G*) to analyze the load balance between each link. The larger the *Gini coefficient* is, the more unbalanced the load. The *Load_G* is defined as:

$$Load\_G = \frac{\sum_{i=1}^{L}\sum_{j=1}^{L}\left|\vartheta_i - \vartheta_j\right|}{2L^2\overline{\vartheta}} \tag{12}$$



(a) The reward during training



(b) The FCT during training

Fig. 5. Observation of DRL-R learning.

Where $\vartheta_i$is the load of link i, and $\overline{\vartheta}$ is the average load of L links.

(4) Cache hit ratio

Cache hit ratio is a traditional metric for caching performance. It is the probability of the request being responded by intermediate node's cache instead of the original content server. Obviously, the higher cache hit ratio is, the higher the caching efficiency.

5.2. Observation of DRL-R learning

Because we represent the *policy* as a CNN (called *policy network*) which takes as input the collection of images (*state*) described above, and outputs a probability distribution over all possible actions, we train the CNN using a variant of the REINFORCE algorithm described in (Sutton et al., 2000; Chen et al., 2018).

In order to train a generalized *policy network* we consider multiple examples of flow arrival sequences during training, henceforth called *flowset,* which come from the open datasets such as CAIDA (A. http://www.caida.o, 2013). In each training iteration, we simulate some episodes for each *flowset* to explore the probabilistic space of possible actions using the current policy, and use the resulting data to improve the policy for all *flowsets*.

One resource allocation state image or one resource demand state image represents one state in one example of flow arrival sequences. Obviously, more images can help to train a more generalized *policy network*. To the *flowset* from CAIDA that include more than 3 million flows, we can factually obtain almost any number of images. However, considering the tradeoff of training time and generalization, this paper uses 60 thousand images to train the *policy network*.

DRL-R is a process of continuous learning, and it is reasonable that more training steps can bring better performance. For example, in the first few thousands of training steps, DRL-R does not learn enough knowledge, and therefore the agent uses stochastic strategy that will lead to big FCT. After some training steps, the FCT of DRL-R will decline, and it means that DRL-R has intelligently gained routing knowledge without human experience. The following experiment results confirm this analysis.

On the one hand, we can observe the learning process of DRL-R from the view of the reward during the training. As shown in Fig. 5 (a), the reward rises gradually and the overall trend is increasing when the training steps increase, though there are some little fluctuations. This shows that the *policy network* is in active training, improving scheduling policy, and the system is constantly optimizing traffic. On the other hand, we can also observe the learning process of DRL-R from the view of network performance. As shown in Fig. 5 (b), the FCT of DRL-R-DDPG and DRL-R-DQN decreases with the increase in training steps as is reasonably expected. Especially, when the training steps increase from 2 k to 10 k, the FCT of two schemes quickly drop. Fig. 5 shows, when the training steps close to 100 k, that the reward and the FCT tend to converge. Our experimental results show that other metrics (such as, throughput and *Gini coefficient*) also show such a similar trend. Owing to space limitation, we have not presented all results.

Besides, our experiment results show that DRL-R already outperforms OSPF when the training step is 30 k. Thus, considering training time, our largest training steps is set to 100 k. Unless otherwise stated, the training steps of DRL-R are set as 50 k in the following parts of this paper.

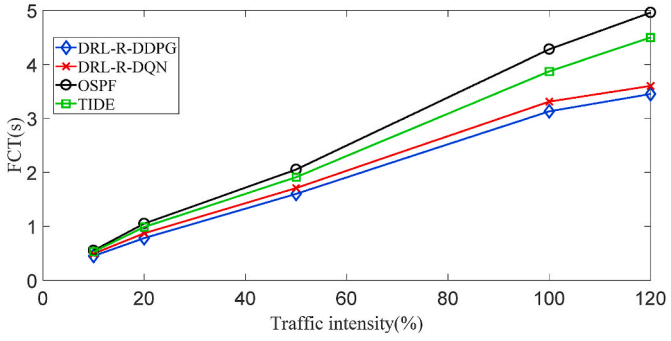


Fig. 6. FCT vs Network loads.

5.3. Experiment results

We run 10 simulations, composed of 100 episodes (i.e., $M = 100$ in Algorithm 1 and Algorithm 2) each; the following are the average

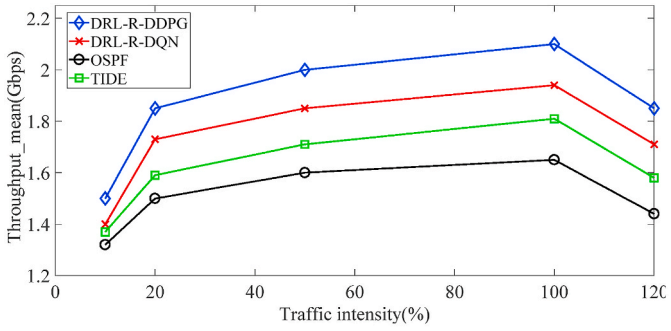


Fig. 7. Throughput_mean vs Network loads.

results.

5.3.1. Network performance under different network loads

To compare the performance under various network loads, we change the traffic intensity when the probability of link failure is fixed at 0% and then record the values of *FCT*, *throughput_mean*, and *Load_G*.

(1) *FCT*

From Fig. 6, we can observe the following results.

Firstly, DRL-R-DDPG's FCT is always lower than DRL-R-DQN's, and the FCT of DRL-R-DDPG reduces by up to 11% compared to DRL-R-DQN when the traffic intensity is 20%. However, compared to DRL-R-DQN, DRL-R-DDPG has a smaller and smaller advantage with the increase in traffic intensity. The reason of this result is the following: Because DRL-R uses an action representation at path level where each action corresponds to select a specific end-to-end path, it need adjust the path to delicately manipulate all traffic, and thus the path action space is large in a scaled network. Differently from DQN, DDPG's output is not discretized as a small set of fixed actions and this makes DDPG suitable for the generation of continuous actions. Furthermore, as shown in section 3.2.3, because of three improvements compared to DQN, DDPG is better feasible for routing with a complex and large action space.

Secondly, compared to OSPF, the FCT of DRL-R-DDPG and DRL-R-DQN show a more and more big advantage with the increase in traffic intensity. The FCT of DRL-R-DDPG reduces by up to 31% compared to OSPF when the traffic intensity is 120%. This mainly results from the fact: although packet-processing delay in the protocol stack also contributes a little to the overall FCT, the transmission delay caused by packet queueing in switches is a more dominating source when the traffic intensity is higher. When the traffic intensity is higher, routing strategy, which determines the transmission delay, becomes more important.

Thirdly, because DRL-R-DDPG can find some available cache (actually, some data sources) to reduce the path length of transmitting data, DRL-R-DDPG needs less FCT than TIDE, though DRL-R-DDPG and TIDE both employ the DDPG algorithm. Compared to TIDE, DRL-R-DDPG's FCT reduces by an average of 19% and by up to 23%, besides, this advantage is more and more with the increase in traffic intensity.

Fourthly, we can also observe that, no matter which scheme, their FCT decreases with the increase in network traffic load. This is easy to understand because higher traffic load leads to longer waiting time even congestion.

(2) Throughput

Fig. 7 demonstrates that four schemes' *throughput_mean* goes up with traffic demand before the traffic intensity reaches 100%. This is because injecting more traffic into the network brings larger throughput because serious congestion does not occur before the traffic intensity is than

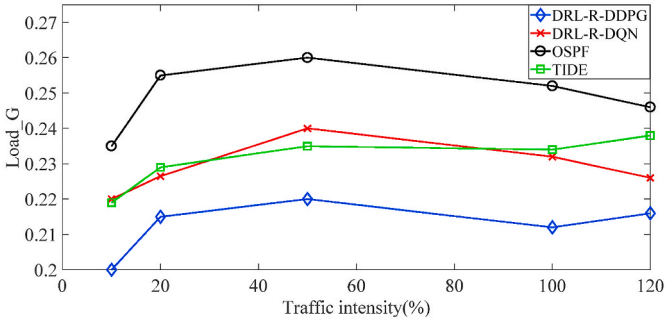


Fig. 8. Load_G vs Network loads.

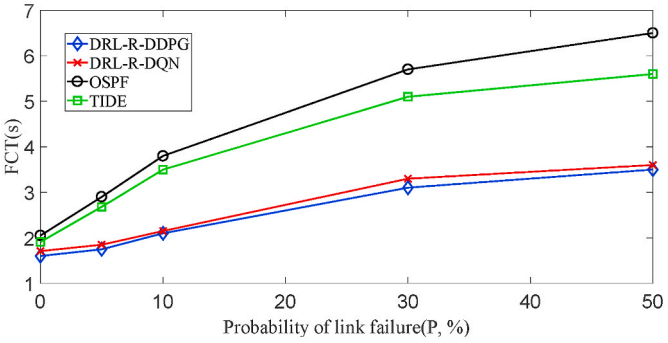


Fig. 9. FCT vs Network failure.

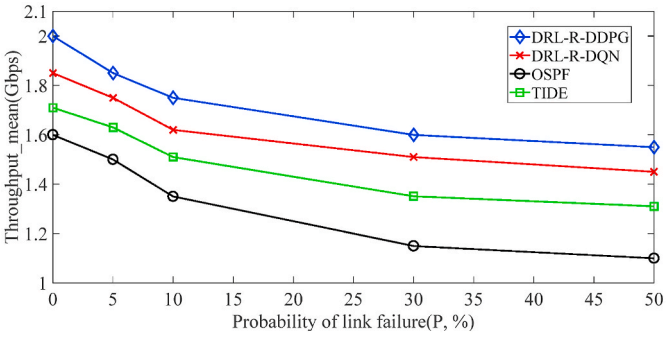


Fig. 10. Throughput_mean vs Network failure.

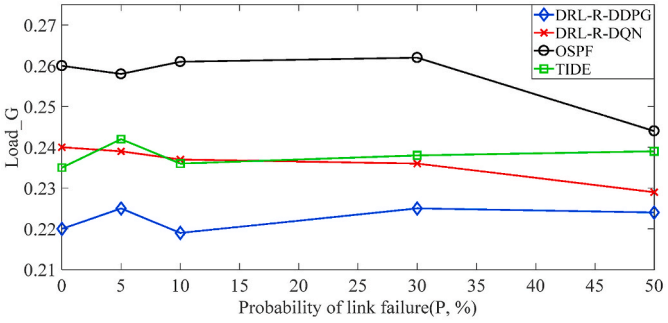


Fig. 11. Load_G vs Network failure.

100%. However, naturally, after the traffic intensity exceeds 100%, the throughput of four schemes decreases sharply due to a serious congestion caused by the overloading in the network.

Compared to OSPF, the *throughput_mean* of DRL-R-DDPG and DRL-R-DQN holds a larger advantage with the increase in traffic intensity. The *throughput_mean* of DRL-R-DDPG increases by up to 28% than OSPF's when the traffic intensity is 120%. Compared to TIDE, DRL-R-DDPG's *throughput_mean* increases by an average of 15% and by up to 17%. Actually, these results further confirms the FCT results above-mentioned, because larger throughput can reduce FCT. In the network with OSPF, this can be explained by the traffic congestion and packet losses that occur when the traffic intensity increases to a certain value. On the contrary, as shown in section 5.3.3, the DRL-R can reduce the probability of congestion by effectively finding data cached to reduce the length of routing path or choosing non-congested path. However,

OSPF selects the shortest path depending on the link's bandwidth only, and not considering the distribution of node's cache.

Besides, *throughput_mean* of DRL-R-DDPG is always higher than that of DRL-R-DQN under different traffic intensity. The improvement is up to 8.2%.

(3) Balance *of Link Load*

This section analyzes the load between each link for four schemes by the metric of *Load_G*.

Because OSPF finds the shortest path by Dijkstra algorithm, this scheme guides the traffic to the same shortest path for every source-destination pair even if it is congested when other paths are idle. This leads to an unbalance of link load. However, the DRL-based methods interact with the network step by step to maximize its objective, and they can find different paths for a source-destination pair under different network state. Such an idea of constantly trying can avoid unbalance of link load.

Fig. 8 confirms the above-mentioned inference. The *Load_G* of DRL-based methods (DRL-R-DDPG, DRL-R-DQN and TIDE) are always significantly lower compared to OSPF. The *Load_G* of OSPF increased by up to 18.8% than DRL-R-DDPG's. Besides, the *Load_G* of DRL-R-DDPG is always lower than TIDE's, and it reduced by up to 9.3%.

5.3.2. Routing robustness under different network failure

The failure of some links lead to the change of network topology and network resource distribution, and different routing schemes may exhibit different robustness for these changes. In fact, it is easy to understand that DRL-R can learn and adapt to these changes faster because DRL agent keeps interacting with the network and making wise decisions. Thus, DRL-R can find the optimal paths from the dynamic network easier than other scheme. This intuitive inference will be confirmed by the later experiment results.

In the experimental process, we randomly select 20% of all links and let them interrupt with a probability of *P* (referred as the probability of link failure). In order to compare the routing robustness under various degree of network failure, we change the *P* from 0% to 50% when the traffic intensity is fixed at 50%.

From Fig. 9, Fig. 10, and Fig. 11, whether *FCT*, *throughput_mean* or *Load_G*, DRL-R-DDPG and DRL-R-DQN always outperform OSPF under different *P*. Moreover, their performance degradation caused by network failure is lower than that of OSPF. The *FCT* of DRL-R-DDPG reduced by up to 47% than OSPF's when the *P* is 50%. On other hand, the *throughput_mean* of DRL-R-DDPG improves by up to 40% compared to OSPF when the *P* is 50%.

Moreover, whether considering *FCT*, *throughput_mean* or *Load_G*, DRL-R-DDPG and DRL-R-DQN always outperform TIDE under different *P*. Compared to TIDE, the *FCT* of DRL-R-DDPG reduced by an average of

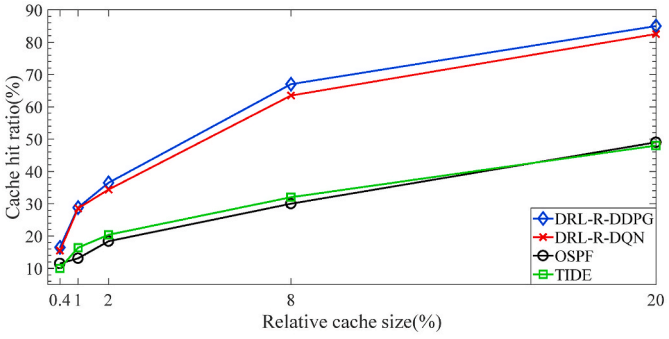


Fig. 12. Cache hit ratio vs Relative cache size.

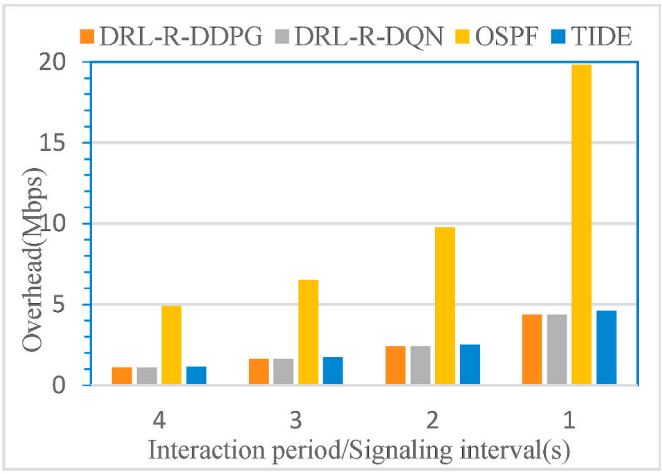


Fig. 13. overhead vs interaction period/signaling interval.

33% and by up to 39%. On other hand, the *throughput_mean* of DRL-R-DDPG improves by an average of 17% and up to 18.5% than TIDE. It reduced *Load_G* by up to 7.3%.

In summary, for the case of topology changes not frequently and greatly, the results mentioned above confirm the robustness of DRL-R.

5.3.3. The utilization efficiency of cache

As described previously in this paper, we convert the routing problem into a job-scheduling problem in resource management. This section demonstrates utilization efficiency of cache by different schemes.

Commonly, the cache size in the Internet is much smaller than the content size within the Internet. We use relative cache size to evaluate the scarcity degree of cache. It is defined as the percentage of the size of all nodes' cache and the size of all content.

DRL uses the resource-recombined as the *Cost* of link between two nodes. In other words, we transit the routing metric from single link state to the state of resource-recombined that combines bandwidth and cache. Furthermore, by continually interacting with the network and making wise decisions, DRL-R can find near data cached to reduce the length of routing path. However, OSPF always selects the shortest path but not considering the cache. It may fortuitously and randomly hits the cache on the shortest path instead of actively looking for cache as DRL-R done. Moreover, though TIDE also employs DRL to find optimal routing path, it does not actively look for cache as DRL-R does. Thus, its cache hit ratio is the result of fortuitously and randomly hitting cache, like as OSPF. The following experiment result can confirm these analyses.

Fig. 12 plots the cache hit ratio with relative cache size varying widely from 0.4% to 20%. It is intuitive that cache hit ratio of four schemes increase as relative cache size grows. The cache hit ratio of DRL-R-DDPG and DRL-R-DQN always hold a significant advantage over OSPF under different relative cache size. Compared to OSPF, DRL-R-DDPG can improve cache hit ratio by up to 2.23X. As expected, TIDE's

Table 3
DRL-R's additional cost.

The actor network: hidden layers/ neurons of each hidden layer	The critic network: hidden layers/ neurons of each hidden layer	Training time for 50 k training steps(s)	Added CPU utilization when running (%)	Response delay (ms)
5/30	1/30	8665	0.71	1.11
2/30	3/30	8020	0.63	1.09
1/200	1/200	8150	0.58	1.01
1/30	1/30	7147	0.56	1.04

cache hit ratio is far less than DRL-R's. We can also see that OSPF's and TIDE's cache hit ratio show slower growth than that of DRL-R-DDPG and DRL-R-DQN with the increase in relative cache size. Actually, these advantages already have been converted into lower FCT and higher throughput as well as better load balance as mentioned above.

5.3.4. Routing overhead

The routing overhead of DRL-R comes from the communication overhead of interacting between agent and network every one period, such as collecting *state* from the network *w* and feed-backing *reward* from *w*. We refer this period as *interaction period*. Among them, collecting *state* (resource allocation *state* and resource demand *state*) is the main part of this communication overhead. Obviously, a shorter *interaction period* can let the agent acquire more frequent reports of network status and performance, and then can control the network more finely, but brings more routing overhead. Thus, we need to consider a tradeoff between performance and overhead.

At the same time, according to its algorithm, OSPF needs bidirectional exchange, every one *signaling interval*, the routing information between two neighbors for all switches in the network. In other words, this is a periodic flood-style exchange. This directly generates routing overhead.

We assume that the SD-DCN network *w* has *V* switches and *L* links. A link connects two adjacent neighbors where routing information is sent 2 times between them, thus, to OSPF, the number of periodic flood-style exchange is $\propto\left(2 \times L \times \frac{1}{signaling\ interval}\right)$. In addition, except to such a periodic exchange, OSPF also triggers a flood-style exchange when the one link's state changes. Thus, in a dynamic network, such a frequently temporary triggering also leads to lots of additional routing overhead. However, to DRL-R, all switches unidirectional send resource allocation *state* to the controller (agent), thus, the number of its exchange is $\propto\left(V \times \frac{1}{interaction\ period}\right)$. Besides, resource demand *state* of a flow is only sent once before it finishes. DCN generally has dense links, $L \gg V$. Thus, DRL-R's routing overhead should be less than OSPF's routing overhead when *interaction period* = *signaling interval*. This analysis is confirmed by the following experiment results.

As shown in Fig. 13, the routing overhead of four schemes all increase with the decrease in *interaction period/signaling interval* as reasonably expected. We can also see that DRL-R-DDPG and DRL-R-DQN have almost the same routing overhead due to their similar workflow. However, the routing overhead of DRL-R-DDPG and DRL-R-DQN are always far less than OSPF's, and DRL-R's routing overhead is 22–25% of OSPF's.

TIDE collects the network status (including link capacity and link utilization) and the reward (including delay, jitter, throughput and loss). However, DRL-R collects the network status (only including resource allocation state and resource demand state) and the reward (only including the FCT of each flow and the throughput of node). Thus, TIDE generally generates slightly more (about 5%) routing overhead than DRL-R.

5.3.5. Additional cost of DRL-R

This paper evaluates the additional cost of DRL-R from the view of offline training and online running phases. For running phases, the cost refers to the delay, computation load and storage load added by the use of DRL, where they can be measured by response delay, CPU utilization and memory utilization, respectively. Taking DDPG as an example, the results using different parameters are shown in Table 3.

From the view of time cost, the training time for 50 k training steps is less than 8700 s. Actually, the training of DRL-R is operated offline to avoid performance bottleneck and can be speed further by GPU or other hardware accelerators. Correspondingly, in the running phase, the response delay is less than 1.11 ms. DRL-R also requires very low computation (added CPU utilization is less than 0.71%). These costs are sufficiently low to satisfy line speed forwarding.

From the view of storage cost, when our experiment set the *experience replay memory D* in Algorithm 1 and Algorithm 2 as 5000, to a SDN controller with 16 GB RAM, DRL-R also requires very little memory (the added memory utilization is less than 0.01%). Actually, the memory of SDN controller in a real network is more than 64 GB, thus, the added memory utilization is almost negligible.

Finally, note that the *policy network* only needs to be offline trained once, and retraining is only needed when the network environment notably changes. Furthermore, the near-optimal performance can be calculated in single step after the DRL model is trained. Compared with OSPF, this presents a huge advantage for real time network applications.

In summary, DRL-R's additional cost is minimal to satisfy the two features of scalability well and is deployable in a large-scale network.

6. Discussion

In this section, we discuss several challenging research directions, constituted of our future works.

6.1. The method of agent interacting with network

Because DRL is a continuous learning process, how the DRL agent efficient interacts with network is an interest research direction. This interaction includes two directionally issues: fine-grain network monitoring (collecting *state* from the network *w* and feed backing *reward* from *w*) and controlling the switch (taking an *action* to *w*).

For fine-grain network monitoring, in-band network telemetry (INT) (Kim, 2016) is a good option. INT is a framework designed to allow the collection and reporting of network state, by the data plane, without requiring intervention or work by the control plane. INT's one of the greatest advantages is that its real-time feature can let the delay between the occurrence of the event and its availability in the *state* or *reward* information to be almost negligible.

For controlling the switch, this paper uses OpenFlow's flow table to create the routing path and forwarding rules for switches. Actually, the programming protocol-independent packet processors (P4) (Bosshart et al., 2014) is a better choice compared to OpenFlow's flow table. Benefitted from P4's programmability to data plane, a variety of forwarding granularities (e.g., packet, flow, and coflow etc) can be created to meet the demand of different applications.

6.2. Directly allocating resource

In a DCN with multiple network resources, this paper lets multiple network resources to recombined, and then lets *RR* to be used as a measurement of routing paths' quality. Actually, we can also let DRL agent directly allocate these resources instead of firstly recombining and then allocating. Directly allocating resource means more fine-grain controlling, however it will bring more enormous state space due to dynamic resource allocation state and resource demand state. This requires a lager input layer of CNN as the *policy network*.

6.3. Generalization ability of DRL-R

Though DRL-R is experience-driven and model-free, the self-learned knowledge may be not efficient when the topology frequently and greatly changes, thus a re-training process is required. Therefore, the generalization ability is very critical when DRL is used to the dynamic network scenario. However, as shown in Table 3, the training time for 50 k training steps is less than 8700 s, and this time is enough short to the relatively stable DCN. In fact, the experiment results show that DRL-R already outperforms OSPF when the training step is 30 k whose training time is 5220 s. In other words, it means that DRL only needs a training time of 5220 s to obtain a good performance. Naturally, when

DRL-R is applied to the frequently and greatly changed network, the generalization ability is an open challenge. On one hand, GPU or other hardware acceerators can further decrease the training time. On other hand, a knowledge-transfer scheme, such as, transfer learning (Weiss et al., 2016), should be researched to ensure that a trained model can be transplanted to other networks with similar topology without much re-training cost. Finally, online learning (Shalev-Shwartz, 2012) aims to make an accurate prediction given knowledge (maybe partial) of the correct answer to previous prediction tasks, thus, online learning and Graph Neural Networks (Almasan et al., 2019) may be also a good manner to deal with this challenge.

6.4. Reward function of DRL-R

In DRL, the reward function is a good signal to guide the agent towards good solutions for an objective. Thus, we can set different reward functions according to different objectives. This paper uses maximizing the throughput of overall network as the primary objective. Actually, DRL-R can also realize other optimizing objectives by using other reward functions (Li et al., 2018; Maoet al., 2019; Saraswat et al., 2019), for example, minimizing end-to-end delay, maximizing deadline meet rate for mice flow, and maximizing load balance.

Our previous work (Liu et al., 2019) proposed a mix-flow scheduling scheme based on DRL where three private link sets for three types of flows are established and then DRL is employed to adaptively allocate bandwidth for each private link set. Where we use the following reward to guide the agent towards simultaneously maximizing the deadline meet rate for mice flow and minimizing the FCT for elephant flow.

$$r_i = \frac{1}{2}(DMR_{i+1} + DMR)_i \times (FCT_{i+1} - FCT_i) \tag{13}$$

DMR_i and FCT_i is the deadline meet rate and flow completion time respectively at timestep i.

7. Conclusion

This paper proposed DRL-R to efficiently allocate cache and bandwidth to improve the routing performance in SD-DCN. Firstly, we recombine multiple network resources (cache and bandwidth) by quantifying the contribution score of them benefitting for the delay reduction. Secondly, the DRL agent deployed on SDN controller continually interacts with network to adaptively make reasonable routing according to the network state, and optimally allocate network resources for traffic, where DQN and DDPG are employed to build DRL-R. Finally, extensive simulation results demonstrate the effectiveness of DRL-R solution in terms of lower FCT, higher throughput, and better load balance as well as better robustness, compared to OSPF and TIDE. DRL-R can improve throughput by up to 40% and 18.5% over OSPF and TIDE, respectively. DRL-R can reduce flow completion time by up to 47% and 39% over OSPF and TIDE respectively. DRL-R can improve the load balance of link by up to 18.8% and 9.3% over OSPF and TIDE respectively. Besides, it was observed that DDPG has better performance than DQN.

CRediT authorship contribution statement

Wai-xi Liu: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing - original draft, Visualization, Supervision, Project administration, Funding acquisition. **Jun Cai:** Software, Visualization, Funding acquisition. **Qing Chun Chen:** Supervision, Project administration. **Yu Wang:** Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (61872102, 61972104, 61802080), the Guangdong Basic and Applied Basic Research Foundation (2018A0303130045), National Key Research and Development Program of China(2018YFB1501201), the International Collaborative Research Program of Guangdong Science and Technology Department (2020A0505100061, 2020A0505100060), the National Social Science Fund of China (15CTQ034), the Science and Technology Innovative Research Team Program in Higher Educational Universities of Guangdong Province (2017KCXTD025), and Project of Guangzhou University (YK2020011), China.

Appendix

Researches have shown that the community structures in a real network is common very much, which are an important property of complex networks. For example, tightly connected groups of nodes in a social network represent individuals belonging to social communities (Newman, 2004).

Given a graph G, a community is a subgraph G', whose nodes are tightly connected, i.e. cohesive. A type of definition of community structures is based on the relative frequency of links. In this case communities are seen as groups of nodes within which connections are dense, and between which connections are sparser. An example is shown in Fig. 2. The simplest formal definition in this class has been proposed: G' is a community if the sum of all degrees within G' is larger than the sum of all degrees toward the rest of the graph.

In complex networks, inter-community communication depends on few key nodes when intra-community communication is frequent. The importance (such as *degree* and *betweenness*) of each node in a community is not equal. The key nodes among the community are not only more easily accessible by nodes within the community, but are also easier to access by the external nodes as well. If the weight of a link is taken into account, it is named as the weighted complex network. For example, as shown in Fig. 2, the entire topology is one community if we do not consider the weight of the link. However, given that the network traffic behavior leads to the variation of link weight, there are two obvious communities, where $v1$ and $v2$ are two key nodes for these two communities respectively.

When the community structure is unknown, modularity is the first and still currently the most popular measurement of the quality of a partition into communities.

The modularity value suggested an alternative approach to find community structure. Considering the tradeoff between computational complexity and performance, this paper used *Fast algorithm* (Newman, 2004) to achieve the community partition without overlap.

References

Caida. http://www.caida.org/data/passive/passive_2013_dataset.xml.
Almasan, P., Suárez-Varela, J., Badia-Sampera, A., et al., 2019. Deep Reinforcement Learning Meets Graph Neural Networks: an Optical Network Routing Use case[J] arXiv preprint arXiv:1910.07421.
Bosshart, P., Daly, D., Gibb, G., et al., 2014. P4: programming protocol-independent packet processors [J]. Comput. Commun. Rev. 44 (3), 87–95.
Boyan, J.A., Littman, M.L., 1994. Packet routing in dynamically changing networks: a reinforcement learning approach. In: Cowan, D.J., Tesauro, G., Alspector, J. (Eds.), Advances in Neural Information Processing Systems 6 (NIPS), vol. 6, pp. 671–678.
Chen, Li, Chen, Kai, et al., 2016. Scheduling mix-flows in commodity datacenters with karuna. Proceedings of the 2016 ACM SIGCOMM Conference. ACM.
Chen, Li, Lingys, Justinas, Chen, Kai, et al., 2018. AuTO: scaling deep reinforcement learning for datacenter-scale Automatic traffic optimization. In: Proceedings of the ACM SIGCOMM Conference. ACM.
Chen, T., Gao, X., Liao, T., Chen, G., 1 Feb. 2020. Pache: a packet management scheme of cache in data center networks. IEEE Trans. Parallel Distr. Syst. 31 (2), 253–265.

Chinchali, S., Hu, P., Chu, T., et al., 2018. Cellular network traffic scheduling with deep reinforcement learning[C]. In: //Thirty-Second AAAI Conference on Artificial Intelligence.
Cui, Y., Song, J., Li, M., et al., 2017. SDN-based big data caching in ISP networks[J]. IEEE Transactions on Big Data (2), 1–13.
Dong, M., Li, Q., Zarchy, D., et al., 2015. PCC: Re-architecting Congestion Control for Consistent High Performance[C]//NSDI, vol. 1, p. 2, 2.3.
Filiposka, S., Juiz, C., 2015. Community-based complex cloud data center[J]. Phys. Stat. Mech. Appl. 419, 356–372.
Fu, Q., et al., 2020. Deep Q-learning for routing schemes in SDN-based data center networks. IEEE Access 8, 103491–103499. https://doi.org/10.1109/ACCESS.2020.2995511.
Hendriks, T., Camelo, M., Latré, S., 2018. Q2-Routing : a qos-aware Q-routing algorithm for wireless ad hoc networks. In: 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 108–115. https://doi.org/10.1109/WiMOB.2018.8589161. Limassol.
Jain, Sushant, et al., 2013. B4: experience with a globally-deployed software defined WAN. Comput. Commun. Rev. 43. No. 4. ACM.
Kim, Changhoon, 2016. Parag Bhide. In-Band Network Telemetry, vol. 6. INT. http://p4.org.
Kong, Yiming, Zang, Hui, et al., 2018. Improving TCP congestion control with machine intelligence. In: Proceedings of the ACM SIGCOMM Workshop on Network Meets AI & ML. ACM.
Kreutz, D., Ramos, F.M.V., Verissimo, P.E., et al., 2015. Software-defined networking: a comprehensive survey [J]. Proc. IEEE 103 (1), 14–76.
LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning [J]. Nature 521 (7553), 436–444.
Li, Y., 2017. Deep Reinforcement Learning: an overview[J] arXiv preprint arXiv: 1701.07274.
Li, Jin, et al., 2018. Multi-authority fine-grained access control with accountability and its application in cloud. J. Netw. Comput. Appl. 112, 89–96.
Lillicrap, T.P., Hunt, J.J., Pritzel, A., et al., 2015. Continuous Control with Deep Reinforcement learning[J] arXiv preprintarXiv:1509.02971.
Lin, S.C., Akyildiz, I.F., Wang, P., et al., 2016. QoS-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach[C]// Services Computing (SCC). In: 2016 IEEE International Conference on. IEEE, pp. 25–33.
Liu, W., 2019. Intelligent routing based on deep reinforcement learning in software-defined data-center networks. In: 2019 IEEE Symposium on Computers and Communications. ISCC, Barcelona, Spain, pp. 1–6. https://doi.org/10.1109/ISCC47284.2019.8969579.
Liu, W.X., et al., 2017. Information-centric networking with built-in network coding to achieve multisource transmission at network-layer. Comput. Network. 115 (3), 110–128.
Liu, W.X., Cai, J., Wang, Y., et al., 2019. Mix-flow scheduling using deep reinforcement learning for software-defined data-center networks [J]. Internet Technology Letters 2 (3), e99.
Liu, W.X., et al., 2020. AAMcon: an adaptively distributed SDN controller in data center networks [J]. Front. Comput. Sci. 14 (1), 146–161.
Luong, N.C., Hoang, D.T., Gong, S., et al., 2019. Applications of deep reinforcement learning in communications and networking: a survey [J]. IEEE Communications Surveys & Tutorials 21 (4), 3133–3174.
Mao, H., Alizadeh, M., Menache, I., et al., 2016. Resource Management with Deep Reinforcement Learning[C]//HotNets, pp. 50–56.
Mao, B., Fadlullah, Z.M., Tang, F., et al., 2017. Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning [J]. IEEE Trans. Comput. 66 (11), 1946–1960.
Mao, Hongzi, et al., 2019. Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM SIGCOMM Conference. ACM.
Medhi, N., Saikia, D.K., 2017. OpenFlow-based scalable routing with hybrid addressing in data center networks[J]. IEEE Commun. Lett. 21 (5), 1047–1050.
Mestres, A., Rodriguez-Natal, A., Carner, J., et al., 2017. Knowledge-defined networking [J]. Comput. Commun. Rev. 47 (3), 2–10.
Mnih, V., Kavukcuoglu, K., Silver, D., et al., 2013. Playing Atari with Deep Reinforcement learning[J] arXiv preprint arXiv:1312.5602.
Mnih, V., Kavukcuoglu, K., Silver, D., et al., 2015. Human-level control through deep reinforcement learning [J]. Nature 518 (7540), 529–533.
Mukhutdinov, D., Filchenkov, A., Shalyto, A., et al., 2019. Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system[J]. Future Generat. Comput. Syst. 94, 587–600.
Newman, M.E.J., 2004. Fast algorithm for detecting community structure in networks. Phys. Rev. 69 (6), 066133.
Pasca, S., Valerrian, Thomas, , Siva Sairam Prasad Kodali, Kataoka, Kotaro, 2017. AMPS: application aware multipath flow routing using machine learning in SDN. In: Communications (NCC), 2017 Twenty-Third National Conference on. IEEE.
Roughan, Matthew, 2005. Simplifying the synthesis of Internet traffic matrices. Comput. Commun. Rev. 35 (5), 93–96.
Saraswat, Surbhi, et al., 2019. Challenges and solutions in software defined networking: a survey. J. Netw. Comput. Appl. 141, 23–58.
Shalev-Shwartz, S., 2012. Online learning and online convex optimization[J]. Foundations and Trends® in Machine Learning 4 (2), 107–194.
Streiffer, Christopher, et al., 2018. DeepConfig: automating data center network topologies management with machine learning. In: Proceedings of the ACM SIGCOMM Workshop on Network Meets AI & ML. ACM.
Suárez-Varela, J., Mestres, A., Yu, J., et al., 2019. Routing Based on Deep Reinforcement Learning in Optical Transport networks[C]//Optical Fiber Communication Conference. Optical Society of America. M2A. 6.
Suarez-Varela, J., Mestres, A., Yu, J., et al., 2019. Feature engineering for deep reinforcement learning based routing[C]//. In: IEEE International Conference on Communications (ICC). IEEE, pp. 1–6.
Sun, P., Hu, Y., Lan, J., et al., 2019. TIDE: time-relevant deep reinforcement learning for routing optimization [J]. Future Generat. Comput. Syst. 99, 401–409.
Sutton, R.S., McAllester, D.A., Singh, S.P., et al., 2000. Policy Gradient Methods for Reinforcement Learning with Function approximation[C]//Advances in Neural Information Processing Systems, pp. 1057–1063.
Tang, F., Mao, B., Fadlullah, Z.M., et al., 2018. On removing routing protocol from future wireless networks: a real-time deep learning approach for intelligent traffic control [J]. IEEE Wireless Communications 25 (1), 154–160.
Valadarsky, A., Schapira, M., Shahaf, D., et al., 2017. Learning to route[C]//Proceedings of the 16th ACM workshop on hot topics in networks. ACM 185–191.
Wang, Mowei, Cui, Yong, et al., June 2018. Neural network meets DCN: traffic-driven topology adaptation with deep learning. Proc. ACM Meas. Anal. Comput. Syst. 2 (2), 25. https://doi.org/10.1145/3224421. Article 26.
Wang, S., Zhang, J., Huang, T., et al., 2018a. A survey of coflow scheduling schemes for data center networks [J]. IEEE Commun. Mag. 56 (6), 179–185.
Wang, M., Cui, Y., Wang, X., et al., 2018b. Machine learning for networking: workflow, advances and opportunities [J]. IEEE Network 32 (2), 92–99.
Wang, S., Tuor, T., Salonidis, T., et al., 2018c. When edge meets learning: adaptive control for resource-constrained distributed machine learning[C]. In: //Proceedings of IEEE International Conference on Computer Communications. IEEE INFOCOM, pp. 63–71, 2018.
Weiss, K., Khoshgoftaar, T.M., Wang, D.D., 2016. A survey of transfer learning [J]. Journal of Big data 3 (1), 9.
Xu, D., Chiang, M., Rexford, J., 2011. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. IEEE/ACM Trans. Netw. 19 (6), 1717–1730.
Xu, Z., Tang, J., Meng, J., et al., 2018. Experience-driven networking: a deep reinforcement learning based approach[C]//Proceedings of IEEE International Conference on Computer Communications. IEEE INFOCOM 2018, 1871–1879.
Yao, H., Mai, T., Xu, X., et al., 2018. NetworkAI: an intelligent network architecture for self-learning control strategies in software defined networks [J]. IEEE Internet of Things Journal 5 (6), 4319–4327.
Ye, H., Li, G.Y., 2018. Deep reinforcement learning for resource allocation in V2V communications[C]//2018. In: IEEE International Conference on Communications (ICC). IEEE, pp. 1–6.
Zahavi, E., Keslassy, I., Kolodny, A., 2014. Distributed adaptive routing convergence to non-blocking DCN routing assignments [J]. IEEE J. Sel. Area. Commun. 32 (1), 88–101.

Wai-xi Liu (M′2012) received the Ph.D. degree in communication and information system from Sun Yat-Sen University, China, in 2013. He is currently an associate professor in the Department of Electronic and Communication Engineering, Guangzhou University. His research interests are in future network, SDN and P4. He has published more than 25 papers. Mailing address: Department of Electronic and Communication Engineering, Guangzhou University, Guangzhou, P. R. China, 510,006. Email address: liuwaixi@sina.com

Jun Cai received the Ph.D. degree in communication and information system from Sun Yat-Sen University, China, in 2012. He is currently an associate professor with Guangdong Polytechnic Normal University, Guangzhou, China. Future networking has been of particular interest over recent years. Email address: caijun@gpnu.edu.cn

Qingchun Chen (M′06–SM′14) received the B·Sc. and M.Sc. degrees (Hons.) from Chongqing University, China, in 1994 and 1997, respectively, and the Ph.D. degree from Southwest Jiaotong University, China, in 2004. In 2004, he joined Southwest Jiaotong University as an Associate Professor, and has been a Full Professor since 2009. He is currently a Full Professor in Guang Zhou University, since 2018. He has authored and co-authored over 100 research papers, two book chapters, and 40 patents. His research interest includes wireless communication, wireless network, channel coding, and signal processing. He was a recipient of the 2016 IEEE GLOBECOM Best Paper Award. He has been serving as an Associate Editor for IEEE ACCESS since 2015. Email address: Qingchun@gzhu.edu.cn

Yu Wang received the Ph.D. degree in computer science from Deakin University, Victoria, Australia. He is currently associate professor with Guang Zhou University. His research interests include network traffic modeling and classification, social networks, mobile networks, and network security. Mailing address: Department of Electronic and Information Engineering, Guang Zhou University, Guangzhou, P. R. China, 510,006. Email address: Yuwang@gzhu.edu.cn

View publication stats