IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 2, 2019, accepted March 1, 2019, date of current version April 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2904539

Deep Reinforcement Learning for Router Selection in Network With Heavy Traffic

RUIJIN DING[1,2,3], YADONG XU[4], FEIFEI GAO[1,2,3], XUEMIN SHEN[5], (Fellow, IEEE), AND WEN WU[5]

[1]Institute for Artificial Intelligence, Tsinghua University (THUAI), Beijing 100084, China
[2]State Key Laboratory of Intelligent Technologies and Systems, Tsinghua University, Beijing 100084, China
[3]Beijing National Research Center for Information Science and Technology (BNRist), Department of Automation, Tsinghua University, Beijing 100084, China
[4]Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, China
[5]Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Corresponding author: Feifei Gao (feifeigao@ieee.org)

This work was supported in part by the National Natural Science Foundation of China under Grant 61831013, Grant 61771274, and Grant 61531011, in part by the Beijing Municipal Natural Science Foundation under Grant 4182030 and Grant L182042, in part by the Guangdong Key Laboratory Project under Grant 017B030314147, and in part by the Key Laboratory of Universal Wireless Communications (BUPT), Ministry of Education, China.

ABSTRACT The rapid development of wireless communications brings a tremendous increase in the amount number of data streams and poses significant challenges to the traditional routing protocols. In this paper, we leverage deep reinforcement learning (DRL) for router selection in the network with heavy traffic, aiming at reducing the network congestion and the length of the data transmission path. We first illustrate the challenges of the existing routing protocols when the amount of the data explodes. We then utilize the Markov decision process (RSMDP) to formulate the routing problem. Two novel deep Q network (DQN)-based algorithms are designed to reduce the network congestion probability with a short transmission path: one focusing on reducing the congestion probability; while the other focuses on shortening the transmission path. The simulation results demonstrate that the proposed algorithms can achieve higher network throughput comparing to existing routing algorithms in heavy network traffic scenarios.

INDEX TERMS Deep reinforcement learning, routing, network congestion, network throughput, deep Q network.

I. INTRODUCTION

The fifth generation (5G) of cellular mobile communications is coming [1], which targets high data rate [2], ultrashort latency, high energy efficiency [3], and massive device connectivity [4]. The number of devices has reached 8.4 billions in 2017 and will further increase to 30 billions by 2020, as predicted in [5]. Such massive amount devices would significantly grow the network traffic data. As a result, the existing routing protocols would face tremendous pressure in maintaining the users' Quality of Experience.

Specifically, the existing routing protocols such as OSPF [6], IS-IS [7], RIP [8], EIGRP gradually become unsuitable for the network with big data, high data rate, and low latency requirements. The key reason is that these protocols rely on calculating the shortest path from a source router to its destination [9] without considering the actual network states such as the remaining buffer size of each router. When the amount of data is small, these shortest-path based protocols bring low latency to the network. However, when the network data traffic volume dramatically increases, certain routers selected by multiple paths may suffer from terrible traffic load. Especially, when the data volume exceeds the buffer size of the selected routers, the network will be congested, which decreases the network throughput and increases the network delay. In other words, the existing routing protocols are not intelligent enough to adjust their transmission strategies according to actual network states.

On the other side, with the growth of computing capability and the explosion of data, Artificial Intelligence (AI) is drastically promoted in recent years, where the great computing capability enables to imitate deeper neural network (DNN) while the big data could provide sufficient training samples. Probably the most successful example is the deep learning (DL) [10] that emerges from the artificial neural network (ANN). DL could build DNN to simulate human brain in order to learn and recognize abstract patterns [11] and has been widely applied in image classification [12]–[14],

The associate editor coordinating the review of this manuscript and approving it for publication was Longzhi Yang.

object detection [15]–[19], communications [20]–[25], as well as many other fields.

DL has also been adopted in routing problems. For example, it could imitate the OSPF protocol [26] to reduce the signaling overhead. However, the algorithm in [26] is essentially an imitation of traditional protocols, and is insufficiently intelligent to deal with complicated network states. Following [26], a deep convolutional neural network based routing algorithm has been proposed in [27], which utilizes the neural network to judge the network congestion caused by the path combination. However, building a neural network for each possible path combination would result in a large number of neural networks for training, and therefore increasing the demand on computing resources.

However, DL generally requires label information for the training data, which then demands for massive manual efforts. In addition, DL is inherently an approximation of certain function and is not suitable for decision-making problems, such as routing, energy allocation, and recommender system. In this case, deep reinforcement learning (DRL) [28] emerges as an alternative to solve decision-making type problems. Compared with traditional reinforcement learning methods[1] [29], DRL takes advantage of function approximation ability of DL to solve practical problems with large-scale state and action space [30]–[32]. For instance, DRL could help the energy harvesting devices allocate the energy to maximize the sum rate of the communications, predict the battery power accurately [33], or guide the two-hop communications to achieve high throughput [34]. Moreover, DRL has been utilized to rank in E-commerce search engine for improving the total transaction amount [35].

In this paper, we design two DRL-based online routing algorithms to address the network congestion problem. The proposed algorithms can reduce the probability of network congestion and shorten the length of transmission paths, i.e., the number of hops from the source router to the destination. The main contributions of this paper are summarized as follows:

- We leverage router selection Markov decision process (RSMDP) concepts to formulate the routing problem and define the corresponding state space, action space, reward function, and value function.
- We propose two online routing algorithms, i.e., source-destination multi-task deep Q network (SDMT-DQN) and destination-only multi-task deep Q network (DOMT-DQN), which can learn from past experiences and update routing policies in real time.
  SDMT-DQN is able to significantly reduce the congestion probability, while the corresponding path length may occasionally be long. In comparison, DOMT-DQN can significantly shorten the path length as well as maintaining the congestion probability at an acceptably lower level.

[1]Reinforcement learning (RL) is a learning technique that an agent learns from the interaction with the environment via trial-and-error.

The rest of the paper is organized as follows. Section II states the routing problem and outlines the system model. In Section III, we introduce RSMDP in detail and analyze the setting of some parameters. The proposed DRL algorithms are detailed in Section IV. Section V provides the simulation results while Section VI concludes the paper.

II. PROBLEM STATEMENT AND SYSTEM MODEL

A. PROBLEM STATEMENT

We assume that the network operates in a time-slotted fashion with normalized time slot. Transmitting a data packet from the source router to the destination is regarded as a *data transmission task*. At each time slot, a task selects the next hop router and the data packet is transferred to it. This process is continued until the data packets arrive at the destination. Network congestion happens when the size of the arriving packet exceeds the remaining buffer size of the router.

The traditional routing protocols are formulated as a classical combinatorial optimization problem, where the data packets are transmitted along the shortest path. Under such shortest path principle, certain routers may be simultaneously selected for multi-tasks, which then very likely leads to network congestion due to the finite buffer size of the routers.

For example, as shown in Fig. 1, three packets from L_0, L_1, L_2 are transmitted to the destination L_8. Based on the shortest path principle, L_4 would be chosen as the next hop for the packets by the traditional protocols. When the packets are relatively large, the remaining buffer size of L_4 will be not sufficient and the network is prone to congestion. Moreover, when the same or similar situation appears again, traditional routing protocols would fall into the congestion again. Even though the network congestion has occurred many times before, the traditional routing protocols would still select the same/similar routing path. Therefore, it is necessary and important for the routing strategy to learn from the past experience and make itself sufficiently intelligent to choose optimal routing paths according to the network states.

B. SYSTEM MODEL

Consider a general backbone network with N routers in the set $\mathcal{L} = \{L_1, L_2, \ldots, L_N\}$. Define $\mathcal{L}_s$, $\mathcal{L}_d$, and $\mathcal{L}_r$ as the disjoint sets of source routers, destination routers, and regular routers, respectively, with $\mathcal{L} = \mathcal{L}_s \cup \mathcal{L}_d \cup \mathcal{L}_r$. Moreover, there are $|\mathcal{L}_s| \triangleq N_s$, $|\mathcal{L}_d| \triangleq N_d$, $|\mathcal{L}_r| \triangleq N_r$, and $N_s + N_d + N_r = N$.

Let $D_{i,t}$ and $B_{i,t}$ denote the total size of all packets and the remaining buffer size in L_i at time slot t, respectively. Define $\boldsymbol{B}_t = \left[B_{1,t}, \cdots, B_{N,t}\right]$ and $\boldsymbol{D}_t = \left[D_{1,t}, \cdots, D_{N,t}\right]$. We denote the size of the packet newly generated by data source i at time slot t by $V_{i,t}$ and define $\boldsymbol{V}_t = \left[V_{1,t}, \cdots, V_{N_s,t}\right]$ as the size vector of all input packets. The data generation is set as a Poisson process. The state of the network at time slot t can be characterized by a tuple $(\boldsymbol{V}_t, \boldsymbol{D}_t, \boldsymbol{B}_t)$.

During time slot t, the input packets are generated by data sources, and then flow to the source routers and change the

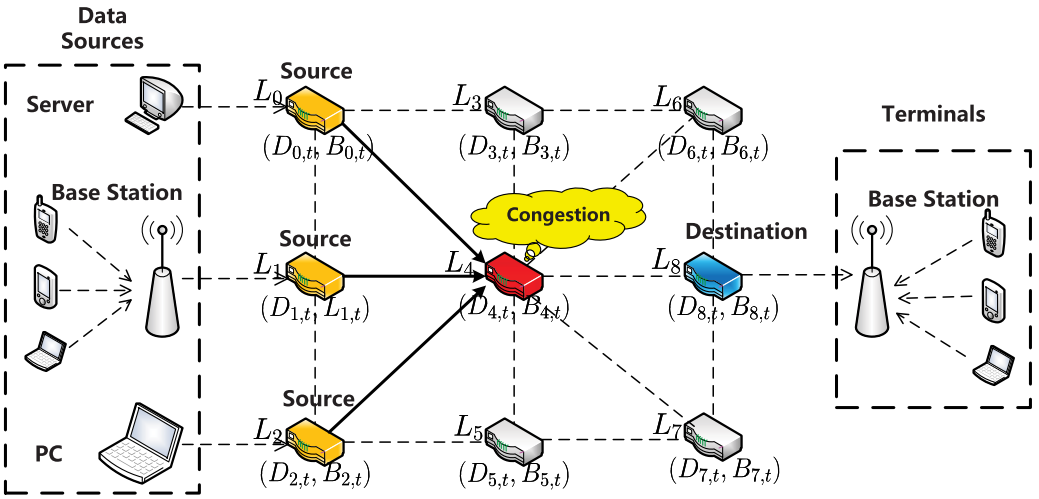


FIGURE 1. The network topology.

remaining buffer size of the source routers. We assume that a packet can be completely transferred from one router to another in one time slot and the values of $D_{i,t}$ and $B_{i,t}$ would change during the transmission process. For instance, if a data packet of size f flows from L_i to L_j at time slot t, then at time slot $t+1$, the tuple $(D_{i,t+1}, D_{j,t+1}, B_{i,t+1}, B_{j,t+1})$ has 6 situations, as shown in (1), shown at the bottom of this page.

When L_i or L_j is the source router, the newly generated data should be considered. And if L_j is the destination router, the data will be transferred to the terminals directly without stored in the buffer.

Note that the current location and the size of data packets would also affect the selection of the next hop router. We then adopt modified *one-hot encoding* vector $\boldsymbol{O}_t$ of size N to represent these characteristics. When the packet is in router L_i, the i^{th} element of $\boldsymbol{O}_t$ is the size of data packet, while the other elements are all zeros. Such modified one-hot encoding can help the computer understand the size and position of the packet. Overall, we can denote the state of each task by $\boldsymbol{S}_t = (\boldsymbol{V}_t, \boldsymbol{D}_t, \boldsymbol{B}_t; \boldsymbol{O}_t)$.

Moreover, the network can be represented by a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of all vertexes corresponding to the routers and $\mathcal{E}$ is the set of edges corresponding to the links between the routers. The data transmission task chooses *action* according to the network state along with the position and size of the packet, where *action* is defined as the link between the current router and the next hop router. For instance, the task whose packet in L_i selects L_j as the next router, which means that $link(i,j) \in \mathcal{E}$ is selected as the action. Besides, the link between two routers is bidirectional, i.e., a data packet can be transferred from L_i to L_j or conversely, denoted by $link(i,j)$ and $link(j,i)$, respectively. Let $\mathcal{A}$ denote the set of all possible actions, i.e. $\mathcal{A} = \mathcal{E}$, with cardinality $|\mathcal{A}| = N_a$. Note that not all actions are valid for a data transmission task, since the packet can only be transferred to the router connecting to its current position. Namely, the task can only choose the link starting from the current position of its packet as the valid action. Therefore, during the transmission process, the valid actions of the task are always changing according to its current position.

III. ROUTER SELECTION MARKOV DECISION PROCESS

In this section, we formulate the routing process as a Markov Decision Process (MDP), where the agent is the data transmission task and the environment is the network.

A. DEFINITION OF RSMDP

In the considered scenario, the tasks decide the next hop routers, and the corresponding decision-making process can be modeled as a MDP with rewards and actions. The MDP is represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where

- The state space is denoted by $\mathcal{S}$, which consists of the terminal state and the nonterminal states. The terminal state is a special state, which indicates that the

$$\begin{cases} (D_{i,t}+V_{i,t}-f, D_{j,t}, B_{i,t}-V_{i,t}+f, B_{j,t}) & i\in\mathcal{L}_s, j\in\mathcal{L}_d \\ (D_{i,t}+V_{i,t}-f, D_{j,t}+V_{j,t}+f, B_{i,t}-V_{i,t}+f, B_{j,t}-V_{j,t}-f) & i\in\mathcal{L}_s, j\in\mathcal{L}_s \\ (D_{i,t}+V_{i,t}-f, D_{j,t}+f, B_{i,t}-V_{i,t}+f, B_{j,t}-f) & i\in\mathcal{L}_s, j\in\mathcal{L}_r \\ (D_{i,t}-f, D_{j,t}, B_{i,t}+f, B_{j,t}) & i\notin\mathcal{L}_s, j\in\mathcal{L}_d \\ (D_{i,t}-f, D_{j,t}+V_{j,t}+f, B_{i,t}+f, B_{j,t}-V_{j,t}-f) & i\notin\mathcal{L}_s, j\in\mathcal{L}_s \\ (D_{i,t}-f, D_{j,t}+f, B_{i,t}+f, B_{j,t}-f) & i\notin\mathcal{L}_s, j\in\mathcal{L}_r \end{cases} \quad (1)$$

task terminates. If the action is invalid or causes the network congestion, then the state turns into the terminal state. Besides, if the data packet arrives at the destination router, then the task also terminates. The nonterminal states contain all continuing events, where the packets are transferred to the next hop routers without congestion, and have not reached the destination.

- The action space is denoted by $\mathcal{A}$, which corresponds to all the edges of the network topology graph. The actions are divided into valid and invalid parts, depending on the current location of its packet.
- The state transition probability function is denoted by $\mathcal{P}(s, a, s') = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$. In the considered scenario, the state transition probability function is related to the probability distribution of the size of the packets newly generated by the data sources. Because in the state tuple, the vector of newly generated packet size $\boldsymbol{V}_t$ is random.
- The immediate reward on the transition from state s to s' under action a is denoted by $\mathcal{R}(s, a, s')$.
- The discount rate is denoted by $\gamma \in [0, 1)$, which determines the present value of future rewards [29].

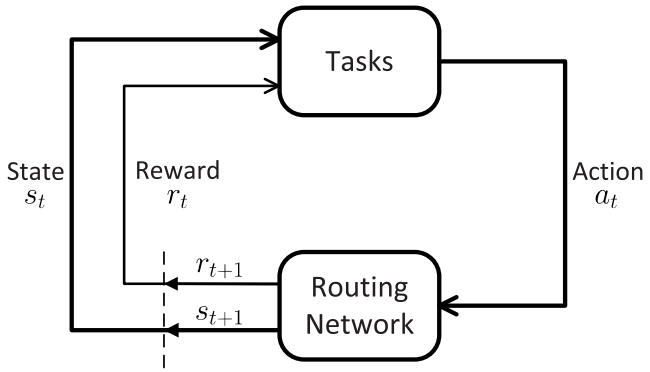


FIGURE 2. Router selection Markov decision process.

As Fig. 2 shows, at each time slot, the task selects the next hop router based on its current state, and the corresponding reward is obtained. The above decision-making and reward feedback process is repeated, which is named as the RSMDP.

A MDP should satisfy the Markov property, which means the future state is independent of the past state given the present state. Mathematically, the Markov property for the MDP is defined as follows:

$$\mathbb{P}(s_{t+1} | s_0, a_0, s_1, \cdots, s_t, a_t) = \mathbb{P}(s_{t+1} | s_t, a_t). \tag{2}$$

From (1), it is obvious that the next state is only related to the current state and the current action. Hence the router selection process satisfies Markov property.

B. REWARD FUNCTION

For any state $s \in \mathcal{S}$, $\mathcal{R}(s, a, s')$ is the immediate reward that numerically characterizes the performance of action a taken with the state transiting from s to s'.

For the problem defined in Section II-A, avoiding network congestion is the prerequisite of seeking for the shortest path. Thus, the reward should first punish the network congestion and then minimize the path length. As described in Section II, since each task can only choose the edge that starts from the router where the packet currently stays, the reward function is supposed to punish the invalid action. Moreover, the reward function needs to consider the path length for the task. In summary, we set the reward function $\mathcal{R}(s, a, s')$ as follows:

$$\mathcal{R}(s, a, s') = \begin{cases} r_c & \text{if network congestion occurs,} \\ r_e & \text{if } a \text{ is invalid,} \\ 0 & \text{if packet arrives destination,} \\ -1 & \text{otherwise,} \end{cases} \tag{3}$$

where the reward -1 helps record the number of hops the data packet is transferred in the network. The constant r_c is the congestion reward that takes a negative value smaller than -1 since the network congestion should be avoided, while constant r_e is the error reward when an invalid action is chosen, which is a negative value smaller than -1 too. The network will feed back a non-negative reward only when the packets arrive at the destination routers. As a result, to avoid the network congestion/invalid action and reduce the path length of each data transmission task, the objective of the routing algorithm should be expressed as finding the optimal policy to maximize the expected cumulative reward for each task. The details will be described in the next subsection.

C. VALUE FUNCTION

From (3), the reward at time slot t can be denoted by $R_t = \mathcal{R}(s_t, a_t, s_{t+1})$. Assume the task turns into the terminal state after T time slots. Then, the cumulative discounted reward from time slot t can be expressed as

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_{t+T} = \sum_{k=1}^{T} \gamma^{k-1} R_{t+k}. \tag{4}$$

Define policy π as a probability distribution over action a, given state s as:

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]. \tag{5}$$

In the considered problem, policy π determines which router should be chosen as the next hop router conditioned on the current state of the transmission task.

Define

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{6}$$

as the action-value function based on policy π of the MDP, i.e., the expectation of the cumulative discounted reward starting from s, taking action a, and following policy π.

The objective of the routing algorithm is to find a policy to maximize the action-value function, i.e.,

$$Q_*(s, a) = \max_\pi Q_\pi(s, a). \tag{7}$$

The optimal policy can be found by maximizing over the optimal action-value function $Q_*(s, a)$ as

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} Q_*(s, a) \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

From (8), if the optimal action-value function $Q_*(s, a)$ can be obtained, we can input $(\boldsymbol{V}_t, \boldsymbol{D}_t, \boldsymbol{B}_t; \boldsymbol{O}_t)$ to compute the value of each action, and then choose the action that maximizes $Q_*(s, a)$. As Section III-B mentioned, the optimal policy obtained from (8) could reduce the path length while avoid network congestion.

One possible way to obtain the optimal action-value function $Q_*(s, a)$ is *Q-learning*, which can be iteratively implemented as

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) \\ &+ \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \end{aligned} \tag{9}$$

during the training process, where α is the learning rate. Iteration (9) updates estimates of the values of states based on values of successor states, which is called *bootstrapping*. In this case, the learned action-value function will converge to the optimal action-value function Q_* [29].

To obtain the value of every action, the reinforcement learning algorithm must try every possible action. However, if the task only chooses the action that maximizes $Q(s, a)$ during the training, then the actions that have not been tried before will be barely chosen, which makes the action-value function fall into the local optimum. Therefore, the algorithm should not only exploit the actions that have been tried before, but also explore new actions. Hence, the ϵ-*greedy* method is usually applied as

$$a = \begin{cases} \arg\max_a Q(s, a), & \text{with probability } 1 - \epsilon \\ \text{random action}, & \text{with probability } \epsilon, \end{cases} \tag{10}$$

where ϵ is the probability of randomly choosing actions.

D. DISCOUNT RATE

In this subsection, we consider the influence of discount rate γ in RSMDP. From (4), we know the cumulative discounted reward leads to "myopic" or "far-sighted" evaluation when γ is close to 0 or 1, respectively. Specifically, when γ is close to 0, the future rewards are hardly considered, while when γ is close to 1, the future rewards are taken into account with heavier weight. The value of the discount rate γ will affect the DRL-based routing algorithm mainly in two aspects:

- How does the objective balance the congestion reward r_c, the error reward r_e, and the remaining cumulative reward?
- What is the relationship between the cumulative reward and the hops of the packet to arrive its destination?

1) REWARDS OF DIFFERENT TYPES

In RSMDP, there are three situations that can terminate the tasks: (i) the packet has reached its destination; (ii) the transmission of the packet results in the congestion in the next hop router; (iii) the action chosen by the task is invalid for transmission. The latter two situations should be averted, which is the prerequisite before shortening the length of transmission paths. Therefore, we should guarantee that the congestion reward and error reward are smaller than the cumulative reward starting from current state. According to the reasons for the termination of the task, there are three cases of the cumulative reward:

- The task reaches the destination router at time slot T. In this case, $R_t = -1$ for $(t = 1, \cdots, T)$. Then the cumulative reward for the whole transmission process of the task equals to

$$G_t = \sum_{t=1}^{T} \gamma^{t-1} R_t = -\sum_{t=1}^{T} \gamma^{t-1} = -\frac{1 - \gamma^T}{1 - \gamma}. \tag{11}$$

- The task chooses the action that leads to the network congestion at time slot T. In this case, $R_T = r_c$, while $R_t = -1$ for $(t = 1, \cdots, T - 1)$. Then the cumulative reward for the whole transmission process of the task equals to

$$\begin{aligned} G_t = \sum_{t=1}^{T} \gamma^{t-1} R_t &= -\sum_{t=1}^{T-1} \gamma^{t-1} + \gamma^{T-1} r_c \\ &= -\frac{1 - \gamma^{T-1}}{1 - \gamma} + \gamma^{T-1} r_c. \end{aligned} \tag{12}$$

- The task chooses the invalid action at time slot T. In this case, $R_T = r_e$, while $R_t = -1$ for $(t = 1, \cdots, T - 1)$. Then the cumulative reward for the whole transmission process of the task equals to

$$\begin{aligned} G_t = \sum_{t=1}^{T} \gamma^{t-1} R_t &= -\sum_{t=1}^{T-1} \gamma^{t-1} + \gamma^{T-1} r_e \\ &= -\frac{1 - \gamma^{T-1}}{1 - \gamma} + \gamma^{T-1} r_e. \end{aligned} \tag{13}$$

Then, we should set r_c and r_e as

$$\begin{aligned} r_c, r_e < \min\{ &-\frac{1 - \gamma^T}{1 - \gamma}, -\frac{1 - \gamma^{T-1}}{1 - \gamma} + \gamma^{T-1} r_c, \\ &-\frac{1 - \gamma^{T-1}}{1 - \gamma} + \gamma^{T-1} r_e \} \end{aligned} \tag{14}$$

As we mentioned in Section III-B, both r_c and r_e are less than -1, therefore

$$-\frac{1 - \gamma^T}{1 - \gamma} > -\frac{1 - \gamma^{T-1}}{1 - \gamma} + \gamma^{T-1} r_c, \tag{15}$$

and

$$-\frac{1 - \gamma^T}{1 - \gamma} > -\frac{1 - \gamma^{T-1}}{1 - \gamma} + \gamma^{T-1} r_e. \tag{16}$$

As a result, (14) can be transformed into

$$r_c, r_e < \min\{-\frac{1-\gamma^{T-1}}{1-\gamma}+\gamma^{T-1}r_c, -\frac{1-\gamma^{T-1}}{1-\gamma}+\gamma^{T-1}r_e\}. \tag{17}$$

We can observe the symmetry of r_c and r_e, therefore, let $r_c = r_e$, then we get

$$r_c = r_e < -\frac{1-\gamma^{T-1}}{1-\gamma}+\gamma^{T-1}r_e. \tag{18}$$

Then we get:

$$r_c = r_e < -\frac{1}{1-\gamma}. \tag{19}$$

2) TRANSMISSION PATH LENGTH

(19) guarantees that the invalid actions and the actions causing network congestion are rarely chosen. When γ equals to 1, the cumulative reward becomes the opposite number of actual hops of the whole transmission process. Then, maximizing the cumulative reward directly leads to the minimization of the path length. However, this property does not hold when $\gamma < 1$. Therefore, considering future rewards in a router selection MDP, we ought to set γ as close to 1 as possible.

IV. DRL BASED ROUTING ALGORITHM

In this section, we design two online routing algorithms with the aid of DQN to handle large-scale state space of RSMDP.

A. DEEP Q NETWORK FOR RSMDP

In the considered scenario, the state includes the size of newly generated data $\boldsymbol{V}_t$, the total data packet size of all routers $\boldsymbol{D}_t$, the remaining buffer size of all routers $\boldsymbol{B}_t$, and the position and size of the current data packet $\boldsymbol{O}_t$. Therefore, the number of the states is huge and we resort to DQN that could utilize DNN to represent the action-value function and tackle the large-scale state space.

As shown in Fig. 3, the input of the neural network is state $\boldsymbol{S}_t$, while the output is the value of each action. Let θ denote the neural network parameters. Then, the action-value function under θ can be represented by $Q(s, a; \theta)$. DQN tries to minimize the loss function defined as

$$L(\theta) = \left[r + \gamma \max_{a'} Q(s, a'; \theta) - Q(s, a; \theta)\right]^2, \tag{20}$$

i.e., the square of temporal-difference error (TD error). Differentiating the loss function with respect to θ, we get the following update:

$$\theta \leftarrow \theta + \alpha \left[r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right] \nabla Q(s, a; \theta) \tag{21}$$

A general assumption for training the deep neural network is that the input data is independently and identically distributed. However, if we utilize the data generated in chronological order $< s_0, a_0, r_1, s_1, \cdots, s_t, a_t, r_{t+1}, s_{t+1} >$,

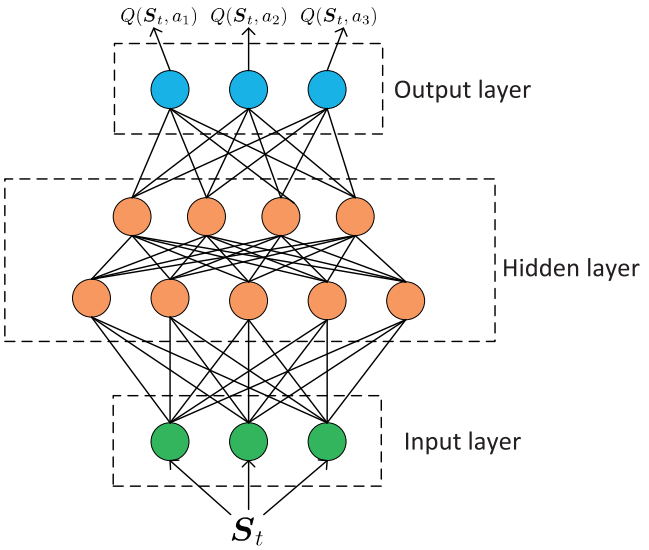


FIGURE 3. Neural network in DQN.

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
s_4, a_4, r_5, s_5
s_5, a_5, r_6, s_6
.....

FIGURE 4. Experience replay memory.

the correlation among input data is quite high, which would affect the performance of neural network. In this case, we can use experience replay to break the correlation among data. The router selection can be divided into the experience tuples (s, a, r, s') as shown in Fig. 4, and the experience tuples are stored in the replay memory, denoted by $\mathcal{D}$. Then, the training data of the neural network is sampled uniformly and randomly from $\mathcal{D}$. Normally, $\mathcal{D}$ can only store the last M experience tuples.

In order to further reduce the correlation among input data, a *target network* is built to deal with the TD error. As shown in (21), the network parameter θ used to compute the target $r + \gamma \max_{a'} Q(s', a'; \theta)$ is the same as that of the action-value function $Q(s, a; \theta)$. An update that increases $Q(s, a; \theta)$ would also increase $Q(s', a'; \theta)$, and therefore bringing correlation and possibly leading to oscillations or divergence of the policy [30]. To further reduce the correlation, DQN uses a separate network to generate the target, whose network parameters are denoted by θ^-. More precisely, network Q is cloned to obtain a target network $\hat{Q}$ every N_u steps. Therefore, the network parameters update to:

$$\theta \leftarrow \theta + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-) - Q(s, a; \theta)\right] \nabla Q(s, a; \theta). \tag{22}$$

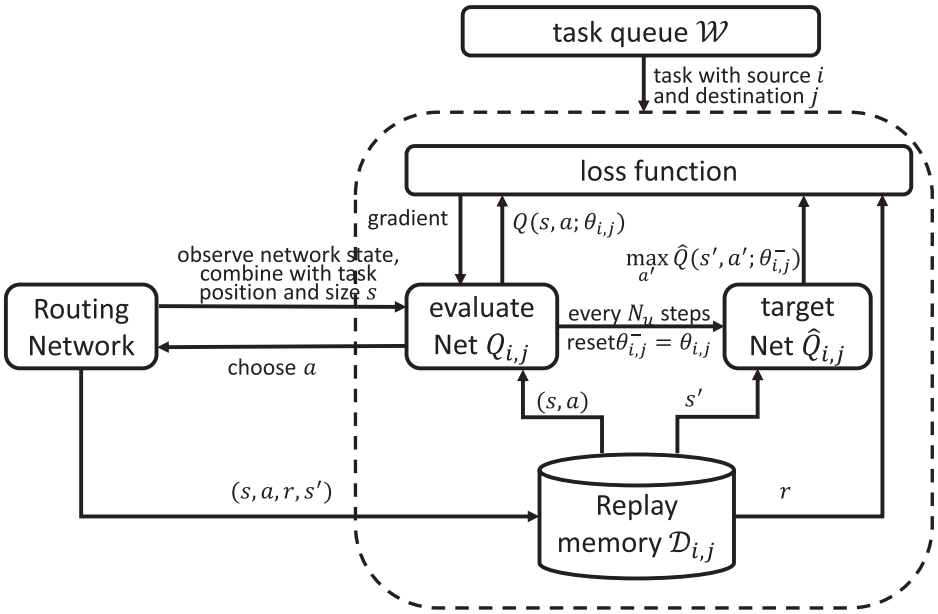


FIGURE 5. SDMT-DQN algorithm.

B. THE PROPOSED SDMT-DQN AND DOMT-DQN ALGORITHMS

Originally, DQN is designed for single agent which cannot help the multi-tasks choose the next hop routers. To tackle this issue, we assume there is a centralized controller with sufficient computation ability that can collect information about the input packets and instruct the router to send the data packets to the next hop router.

We are interested in a distributed solution to find the routing policies of the tasks. Even if the data packets of different tasks are currently in the same router, the tasks may choose different actions, due to their different goals. Hence, the centralized controller needs multiple neural networks to instruct every router for delivering the packets properly. Furthermore, we should categorize the tasks into different classes and apply one uniform neural network for each class. In this paper, we adopt two criteria to classify the tasks, which yields two different algorithms, respectively:

1) THE SDMT-DQN ALGORITHM

In SDMT-DQN algorithm, we classify all the data transmission tasks into $N_s \times N_d$ categories based on their source routers and destination routers. Specifically, all data tasks with the same source router and the same destination router can be considered as the same type of tasks, to share the same neural network. As a result, $N_s \times N_d$ neural networks are needed to represent the action-value functions of all kinds of tasks. For those tasks from source router i to destination router j, there is a corresponding replay memory $\mathcal{D}_{i,j}$ with capacity C to store the experience tuples for training. Moreover, there is a target network to reduce the correlation among input data. The algorithm can be illustrated in Fig. 5.

At the centralized controller, we set a task queue $\mathcal{Z}$ to store the information of the tasks, e.g., the source, and destination router of the task, the packet size and the current position. The centralized controller selects the task in $\mathcal{Z}$ one by one. Then the neural network corresponding to the source router and destination router of the selected task takes the state of the selected task as input, and outputs the value of each action. Afterwards, the centralized controller chooses action for the data packet based on ϵ-*greedy* method. If the selected action is invalid, then the centralized controller: (i) regards the task as termination and stores the corresponding state, action, reward r_e and terminate state in corresponding experience memory; (ii) re-chooses the action whose value is the largest among the valid actions and continues the transmission. Therefore, the invalid action will lead to two experience tuples. This procedure can guarantee the validity of the selected action while storing the invalid action with r_e in the memory, therefore reducing the probability of choosing invalid action afterwards. Then, according to the selected action, the centralized controller can know the next hop router and determine the next state of the task. The possible situations can be listed as follows.

- If the next router is the destination router, then the data transmission task is complete and the state turns into terminal state. The corresponding reward is 0 in this case.
- If the action causes congestion, then the task is terminated and the reward is r_c.
- Otherwise, the centralized controller updates the state of the task and re-appends it to the end of $\mathcal{Z}$. Moreover, the network will return a reward -1.

Then, the centralized controller stores the experience tuples in the corresponding experience memory $\mathcal{D}$, and the neural network samples data from $\mathcal{D}$ for training. Repeat the above procedures until each task in the queue has selected an action. Finally, the centralized controller sends the action

commands of each task to the routers, and the routers send their packets to the next hop routers in accordance with these commands. With such an online algorithm, the neural networks can utilize the latest experiences to improve the performance. The overall algorithm is summarized in Algorithm 1.

2) THE DOMT-DQN ALGORITHM

The DOMT-DQN algorithm can reduce the number of the required neural networks, which differs from the SDMT-DQN algorithm mainly in that the data transmission tasks are classified into N_d categories that only correspond to their destination routers. Hence, the corresponding neural network and the replay memory only depend on the destination of the task. As the number of categories is reduced, the number of tasks for each category increases. Therefore, there is more sufficient training data for each corresponding neural network, which leads to faster convergence.

Note that DOMT-DQN can be demonstrated by modifying Algorithm 1. Specifically, the replay memory $\mathcal{D}_{1,1}, \mathcal{D}_{2,1}, \cdots, \mathcal{D}_{N_s,N_d}$ are changed into $\mathcal{D}_1, \mathcal{D}_1, \cdots, \mathcal{D}_{N_d}$, and the parameters of the neural networks $\theta_{1,1}, \theta_{2,1} \cdots, \theta_{N_s,N_d}$ are substituted with $\theta_1, \theta_2 \cdots, \theta_{N_d}$. The remaining procedures are very similar to Algorithm 1, and the overall steps of DOMT-DQN are summarized in Algorithm 2.

V. SIMULATION RESULTS

In this section, simulations are conducted to evaluate the performance of SDMT-DQN and DOMT-DQN. The probability of randomly choosing action ϵ is set to 0.9. We use Python and the deep learning framework Pytorch for coding and the program is executed on a computer with an Intel Core i7-8700k CPU, 32GB random access memory (RAM), and Nvidia GTX 1070 GPU. The operating system is Ubuntu 16.04.

We compare the performance of the proposed algorithms with the deep learning based algorithm [27] and the traditional routing protocol OSPF. To better demonstrate the performance comparison, we consider the simple network with topology depicted in Fig. 1. Each node is deemed as a router and each edge is deemed as a transmission link. The routers L_0, L_1, L_2 are set as source routers that receive input packets from the data sources and transmit them to the destination router L_8. All the routers in the network can receive and send the data packets. We assume that no matter how big the data packets are, they can be transferred from one router to another in one time slot. If the network congests in a time slot, we will mark it, then compute the network congestion probability by calculating the proportion of time slots that are congested in every 1000 time slots. The buffer size of each router is set to 45 MB, and the packet generation process is set as Poisson.

Algorithm 1 Source-Destination Multi-Task Deep Q Network (SDMT-DQN)

Initialize the task queue $\mathcal{Z}$, the reply memories with capacity C for every source-destination pair $\mathcal{D}_{1,1}, \mathcal{D}_{2,1}, \cdots, \mathcal{D}_{N_s,N_d}$, action-value functions Q with random parameters θ for every source-destination pair $\theta_{1,1}, \theta_{2,1}, \cdots, \theta_{N_s,N_d}$, the corresponding target action-value functions $\hat{Q}$ with parameters $\theta^-_{1,1} = \theta_{1,1}, \cdots, \theta^-_{N_s,N_d} = \theta_{N_s,N_d}$, the buffer size of all the routers, and the network state.
for $t = 1, 2 \ldots, T$ **do**
  The sources generate data tasks and append them to $\mathcal{Z}$.
  The controller obtains the information of the new generated tasks and computes the network state.
  for $n = 1, \ldots, N_t$ (N_t is the number of tasks in $\mathcal{Z}$) **do**
    Pop a task n from $\mathcal{Z}$, combine the current network state and the position and size of task n to get state $s_{t,n}$.
    Select neural network based on source router i and destination router j with parameters $\theta_{i,j}$.
    Choose a random action $\hat{a}$ with probability ϵ, otherwise selecting $\hat{a} = \arg\max_a Q(s_{t,n}, a; \theta_{i,j})$.
    if $\hat{a}$ is invalid **then**
      Store the experience tuple $(s_{t,n}, \hat{a}, r_{t,n}$, terminal state) in $\mathcal{D}_{i,j}$.
      Re-choose a valid action $a_{t,n}$ with the largest value.
    else
      $a_{t,n} = \hat{a}$.
    end if
    Simulate execution action $a_{t,n}$ in the controller, get reward $r_{t,n}$ and next state $s'_{t,n}$, then update the network state.
    Store the experience tuple $(s_{t,n}, a_{t,n}, r_{t,n}, s'_{t,n})$ in $\mathcal{D}_{i,j}$.
    Sample random minibatch of experience tuples (s_k, a_k, r_k, s'_k) from $\mathcal{D}_{i,j}$.
    Set $y_k = \begin{cases} r_k & ① \\ r_k + \gamma \max_{a'} \hat{Q}(s'_k, a'; \theta^-_{i,j}) & ② \end{cases}$.
    ①: if the task terminates.
    ①: otherwise.
    Perform a gradient descent step with a learning rate α on $\left(y_k - Q\left(s_k, a_k; \theta_{i,j}\right)\right)^2$ with respect to the network parameters $\theta_{i,j}$.
    Reset $\hat{Q}_{i,j} = Q_{i,j}$ every N_u steps.
  end for
  The controller sends N_t commands to all routers, and the routers send packets according the commands.
end for

A. COMPLEXITY ANALYSIS

Based on the definition of the input state in Section II, there are $3 \times N + N_s = 30$ units in the input layer of the neural network, while the number of units in the output layer is $N_a = 32$ since the output represents the value of each action. The controller should choose the next hop router

Algorithm 2 Destination-Only Multi-Task Deep Q Network (DOMT-DQN)

1: Initialize the whole system, including the buffer size of all the routers, the network state, the task queue $\mathcal{Z}$, the replay memory $\mathcal{D}_1, \mathcal{D}_1, \cdots, \mathcal{D}_{N_d}$, the action-value functions with random parameters $\theta_1, \theta_2, \cdots, \theta_{N_d}$, and the corresponding target network $\theta_1^- = \theta_1, \cdots, \theta_{N_d}^- = \theta_{N_d}$
2: **for** $t = 1, 2 \ldots, T$ **do**
3: The sources generate data tasks and append them to $\mathcal{Z}$.
4: **for** $n = 1, \ldots, N_t$ (N_t is the number of tasks in $\mathcal{Z}$) **do**
5: Select the corresponding neural network based on the destination router i of task n, θ_i.
6: Choose action with ϵ-*greedy*, obtain the next state, and store the experience tuples.
7: Sample random minibatch of experience tuples (s_k, a_k, r_k, s'_k) from $\mathcal{D}_i$ and update the corresponding parameters θ_i with gradient descent method.
8: Reset $\hat{Q}_i = Q_i$ every N_u steps.
9: **end for**
10: The controller sends N_t commands to all routers, and the routers execute these actions.
11: **end for**

for each task in a very short time, therefore light-weight neural networks ought to be used. The specific neural network architectures for SDMT-DQN and DOMT-DQN are shown in Table 1.

TABLE 1. The neural network architecture.

Algorithm	Input layer	Hidden layers		Output layer
SDMT-DQN	30	50		32
DOMT-DQN		50	40	

The number of the required neural networks for our algorithms is significantly reduced compared with DL-based method in [27]. To be specific, $N_s \times N_d$ and N_d neural networks are required for SDMT-DQN and DOMT-DQN, respectively. For example, considering the network topology of Fig. 1, SDMT-DQN requires three neural networks while DOMT-DQN only needs one neural network.

In addition, the required number of floating point operations (FLOPs) is used as the metric of computational complexity. For convolutional layers, the number of FLOPs is:

$$\text{FLOPs} = 2H_{in}W_{in}(C_{in}K^2 + 1)C_{out}, \tag{23}$$

where H_{in}, W_{in} and C_{in} are height, width and number of channels of the input feature map, K is the kernel size, and C_{out} is the number of output channels.

For fully connected layers, FLOPs is computed as:

$$\text{FLOPs} = (2N_{in} - 1)N_{out}, \tag{24}$$

where N_{in} is the number of input neurons and N_{out} is the number of output neurons [36].

The total computational complexity can be summarized in Table 2. Compared with the DL-based method, the proposed algorithms has much fewer FLOPs for each neural network and number of neural networks. Therefore, the total computational complexity of the two proposed algorithms are extremely lower.

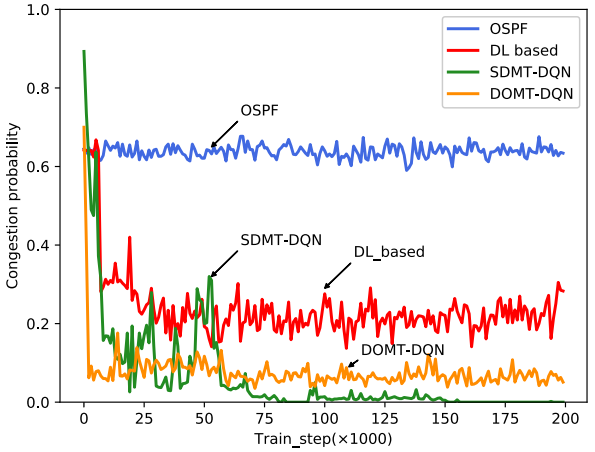


FIGURE 6. The performance comparison between our proposed algorithms and tradition protocol as well as DL based algorithm in terms of congestion probability.

B. PERFORMANCE COMPARISON

In Fig. 6, we compare congestion probabilities of SDMT-DQN, DOMT-DQN, DL based algorithm and OSPF versus the number of training steps. The discount rate γ is set to 0.9, and the mean of Poisson data generation process is set to 15 MB per time slot. The congestion probabilities of OSPF stays at a high level due to the lack of intelligence. In contrast, the congestion probabilities of SDMT-DQN and DOMT-DQN significantly decrease with the increase of training steps because the network has learned from the past congestion and then generates a policy to reduce congestion probability. Moreover, both two proposed algorithms can achieve lower congestion probability compared with the DL based algorithm [27]. This is because the DL based algorithm can only choose from the pre-defined path combinations, instead of exploring the best possible paths from the instantaneous states. We see that the training process of DOMT-DQN converges faster than that of SDMT-DQN. The reason can be explained as follows: The training data of SDMT-DQN is divided into $N_s \times N_d$ categories, while that of DOMT-DQN is only divided into N_d categories. Therefore, at the beginning of the training process, the training data for each neural network in DOMT-DQN is more sufficient than that in SDMT-DQN. It is also seen that with the process of training, the congestion probability of SDMT-DQN can reduce to almost zero, while that of DOMT-DQN maintain at an acceptably low level, because adopting more neural networks of SDMT-DQN could provide better learning ability than DOMT-DQN. Besides, since further classifying the data transmission tasks based on the source routers makes the

TABLE 2. The total complexity comparison of the three algorithms for the network topology in Fig. 1.

Algorithm	Number of required neural networks	FLOPs of each neural network
DL-based method in [27] (no longer than 3 hops)	$6 \times 5 \times 6 = 180$	531.5×10^3
DL-based method in [27] (no longer than 4 hops)	$14 \times 15 \times 14 = 2940$	531.5×10^3
SDMT-DQN	3	4.5×10^3
DOMT-DQN	1	8.2×10^3

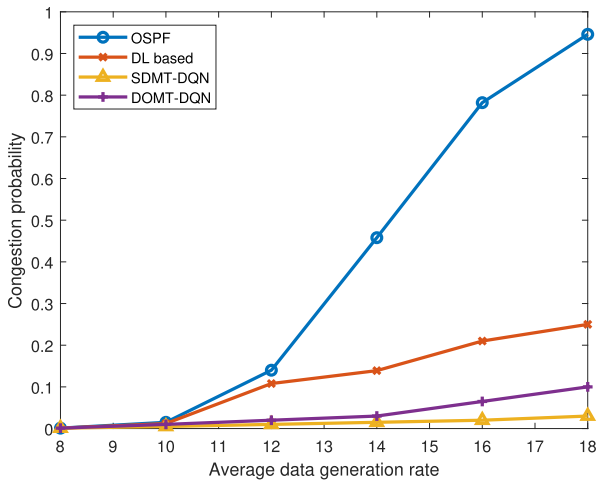


FIGURE 7. Network congestion probability comparison for various packet generation rates.

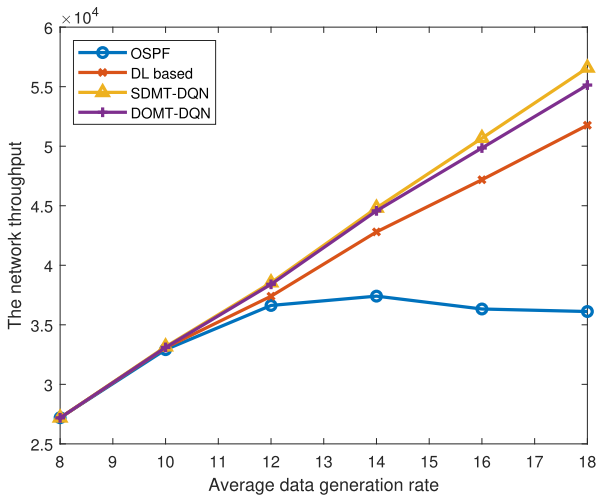


FIGURE 8. Network throughput comparison for various packet generation rates.

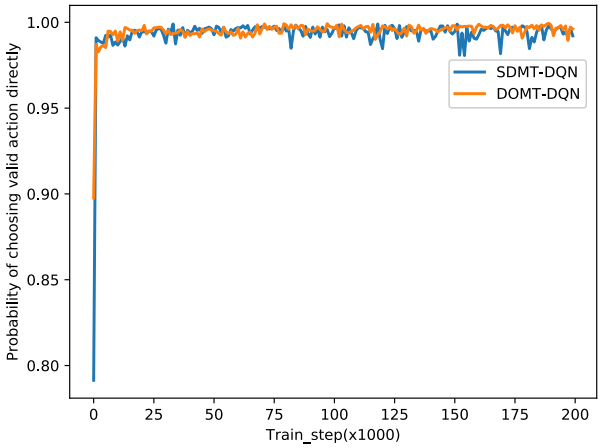


FIGURE 9. The probability of choosing the invalid actions.

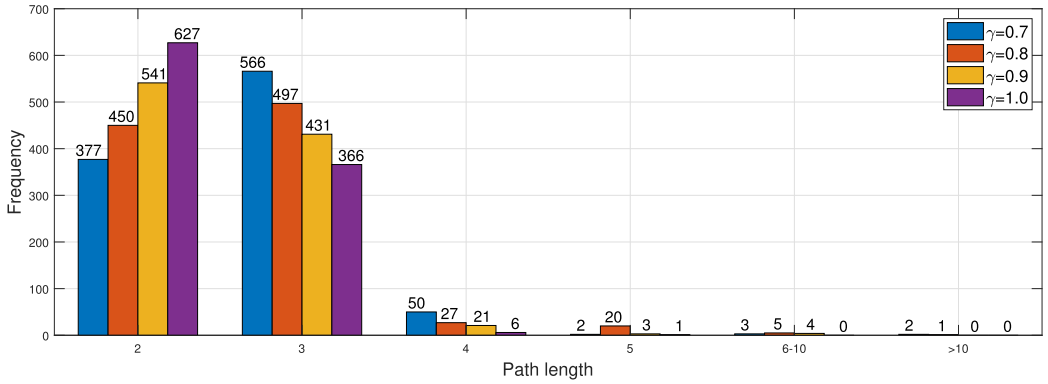
learning process easier for each neural network, SDMT-DQN would yield lower congestion probability than DOMT-DQN.

Next, we compare the congestion probability versus different data generation rates in Fig. 7, where the curves of SDMT-DQN, DOMT-DQN, and the DL based algorithm are calculated by the network parameters after sufficient rounds of training. We can see that when the data generation rate is slow, i.e., the network is idle, the data packets are unimpeded in the network. In this case, none of the four compared methods would cause congestion. However, when the amount of data in the network increases, the congestion probability of OSPF increases significantly. In contrast, the congestion probabilities of SDMT-DQN and DOMT-DQN stay at a low level, which indicates that OSPF can only perform well when the network is idle, while the proposed ones can deal with large amount of data. In addition, the proposed algorithms outperform the DL based algorithm.

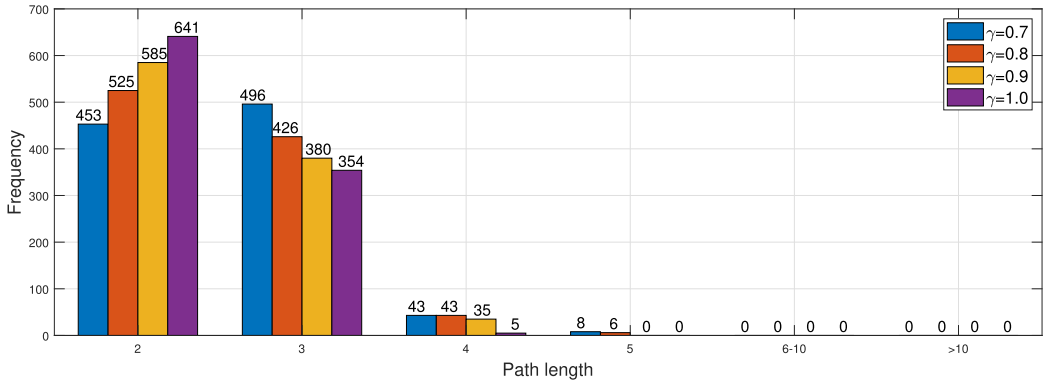
Fig. 8 plots network throughput versus packet generation rates for different algorithms. Similar to Fig. 7, when the network is idle, the performance of OSPF performs similarly to the other three algorithms. However, when the network traffic becomes heavier, OSPF drops a larger number of packets due to the increasing congestion probability. This in turn leads to a decrease in the network throughput. On the contrary, the proposed algorithms can improve the network throughput when the data generation rate increases because the congestion probability can always be maintained at a very low level. Due to the lower congestion probability, SDMT-DQN performs better than DOMT-DQN in terms of the network throughput.

Fig. 9 plots the probability of choosing valid actions in the first trial versus the number of training steps for the proposed SDMT-DQN and DOMT-DQN algorithms. We see that the invalid actions are rarely chosen after very few training steps. Therefore, SDMT-DQN and DOMT-DQN will not require much additional computation to re-choose valid actions.

In Fig. 10, we compare the path length of 1000 transmission tasks under different discount rates of SDMT-DQN and DOMT-DQN. It is seen that the closer γ is to 1, the shorter the path length will be, which is consistent with the analysis in Section III-D. When $\gamma = 1$, (19) seems impossible to satisfy. But in fact, as long as r_c and r_e are smaller than -1,



(a)



(b)

FIGURE 10. The comparison of different discount rates γ in terms of path length. (a) Path length under different discount rates γ based on SDMT-DQN. (b) Path length under different discount rates γ based on DOMT-DQN.

the task tends to choose the actions which would not cause congestion or be invalid along with the training. As a result, when $\gamma = 1$, our algorithms can also reduce the congestion probability, just slightly slower. In addition, DOMT-DQN performs better than SDMT-DQN. Specially, for SDMT-DQN, there are very few paths that are longer than 10, which never happens for DOMT-DQN. This is because when we use SDMT-DQN, the task from one source router may choose another source router as hop router occasionally. Since the probability of this behavior is very low, the training data that guides the network to handle this situation cannot be sufficient. Then, the data packets may be repeatedly transferred between two source routers, and the path length of the corresponding task then becomes very long. On the other hand, DOMT-DQN does not differentiate the tasks according to their source routers. Hence, no matter which router the data packet is transferred to, there can always be sufficient training samples.

In the last example, we demonstrate the scalability of SDMT-DQN and DOMT-DQN in a more complicated network as shown in Fig. 11. In Fig. 12, we compare the proposed algorithms with OSPF in terms of congestion probability.[2] Both the proposed algorithms can significantly

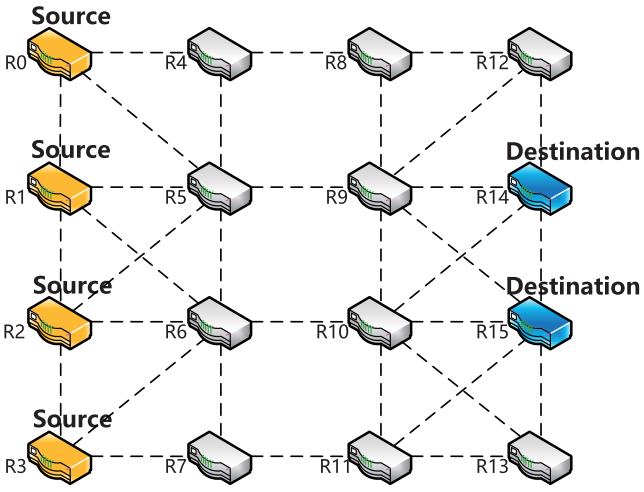


FIGURE 11. A more complicated network topology.

reduce the congestion probability. Similar to Fig. 6, SDMT-DQN performs better than DOMT-DQN in terms of the congestion probability while DOMT-DQN converges faster than SDMT-DQN. Both SDMT-DQN and DOMT-DQN converge slower when being applied in a more complicated network, and the corresponding congestion probability after training will be slightly increased. This is because when the number of routers in the network increases,

[2]The DL based algorithm [27] cannot be implemented in the current computer configuration, since the number of the possible path combinations is too large.

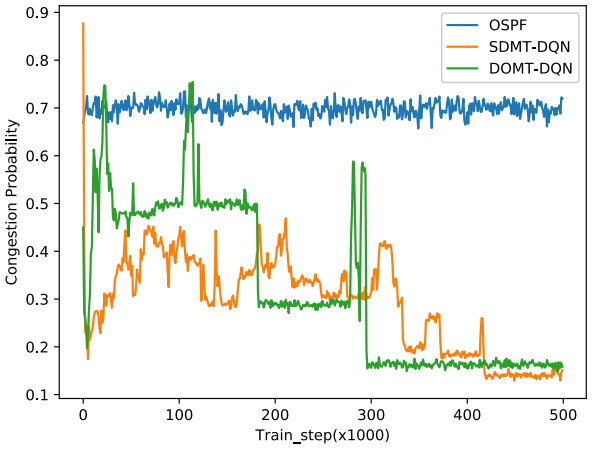


FIGURE 12. The performance comparison of our proposals and tradition protocol OSPF in terms of congestion probability in a more complicated network.

the proportion of valid actions for each task decreases significantly, making it more difficult to learn a good policy for the task.

VI. CONCLUSIONS

In this paper, we have proposed two DRL-based online algorithms to reduce the congestion probability and shorten the transmission path when the network traffic is quite heavy. The simulation results demonstrate that the two algorithms can achieve high throughput in contrast with the traditional routing protocols due to the low congestion probability. Besides, the proposed algorithms have lower computational complexity compared with the DL-based method. It is worth noting that in this article, we only consider the update of the parameters of the neural networks. In the future, we will consider the neural network with dynamic architecture to achieve better performance. Nevertheless, our study demonstrates that DRL is feasible to be applied to routing problem.

REFERENCES

[1] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5G," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 74–80, Feb. 2014.
[2] J. G. Andrews *et al.*, "What will 5G be?" *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.
[3] C.-X. Wang *et al.*, "Cellular architecture and key technologies for 5G wireless communication networks," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 122–130, Feb. 2014.
[4] T. S. Rappaport *et al.*, "Millimeter wave mobile communications for 5G cellular: It will work!" *IEEE Access*, vol. 1, pp. 335–349, May 2013.
[5] A. Nordrum, "Popular Internet of Things forecast of 50 billion devices by 2020 is outdated," *IEEE Spectr.*, vol. 18, no. 6, 2016.
[6] J. Moy, *OSPF Version 2*, Standard RFC-2178, Jul. 1997.
[7] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.
[8] C. Hedrick, *Routing Information Protocol*, Standard RFC-1058, 1988.
[9] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 232–243, Apr. 2002.
[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.
[11] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 98–113, Jan. 1997.
[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Neural Inf. Process. Syst.*, Dec. 2012, pp. 1097–1105.
[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[14] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 2261–2269.
[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.
[16] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 6517–6525.
[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
[18] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2016, pp. 1440–1448.
[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[20] X. You, C. Zhang, X. Tan, S. Jin, and H. Wu. (2018). "AI for 5G: Research directions and paradigms." [Online]. Available: https://arxiv.org/abs/1807.08671
[21] C. Zhang, P. Patras, and H. Haddadi. (2018). "Deep learning in mobile and wireless networking: A survey." [Online]. Available: https://arxiv.org/abs/1803.04311
[22] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.
[23] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *Proc. IEEE 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2017, pp. 1–6.
[24] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "Deep learning-based channel estimation for beamspace mmWave massive MIMO systems," *IEEE Wireless Commun. Lett.*, vol. 7, no. 7, pp. 852–855, Oct. 2018.
[25] C.-K. Wen, W.-T. Shih, and S. Jin, "Deep learning for massive MIMO CSI feedback," *IEEE Wireless Commun. Lett.*, vol. 7, no. 5, pp. 748–751, Oct. 2018.
[26] N. Kato *et al.*, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 146–153, Jun. 2017.
[27] F. Tang *et al.*, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 154–160, Feb. 2018.
[28] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," in *Proc. NIPS Deep Learn. Workshop*, 2013, pp. 1–9.
[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT press, 1998.
[30] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[31] Z. Wang *et al.*, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 2016, pp. 1995–2003.
[32] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[33] M. Chu, H. Li, X. Liao, and S. Cui, "Reinforcement learning based multi-access control and battery prediction with energy harvesting in IoT systems," *IEEE Internet Things J.*, to be published.
[34] A. Ortiz, H. Al-Shatri, T. Weber, and A. Klein. (2017). "Multi-agent reinforcement learning for energy harvesting two-hop communications with full cooperation." [Online]. Available: https://arxiv.org/abs/1702.06185
[35] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application," in *Proc. KDD*, 2018, pp. 368–377.
[36] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. ICLR*, 2017, pp. 1–17.

Authors' photographs and biographies not available at the time of publication.