Toward Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning

Xinyu You, Xuanjie Li, Yuedong Xu, Hui Feng, *Member, IEEE*, Jin Zhao, *Senior Member, IEEE*, and Huaicheng Yan, *Member, IEEE*

***Abstract*—Packet routing is one of the fundamental problems in computer networks in which a router determines the next-hop of each packet in the queue to get it as quickly as possible to its destination. Reinforcement learning (RL) has been introduced to design autonomous packet routing policies with local information of stochastic packet arrival and service. However, the curse of dimensionality of RL prohibits the more comprehensive representation of dynamic network states, thus limiting its potential benefit. In this article, we propose a novel packet routing framework based on *multiagent* deep RL (DRL) in which each router possess an *independent* long short term memory (LSTM) recurrent neural network (RNN) for training and decision making in a *fully distributed* environment. The LSTM RNN extracts routing features from rich information regarding backlogged packets and past actions, and effectively approximates the value function of Q-learning. We further allow each route to communicate periodically with direct neighbors so that a broader view of network state can be incorporated. The experimental results manifest that our multiagent DRL policy can strike the delicate balance between congestion-aware and shortest routes, and significantly reduce the packet delivery time in general network topologies compared with its counterparts.**

***Index Terms*—Deep reinforcement learning (DRL), local communications, multiagent learning, packet routing.**

I. Introduction

PACKET routing is a very challenging problem in distributed and autonomous computer networks, especially in wireless networks in the absence of centralized or coordinated service providers. Each router decides to which neighbor it should send his packet in order to minimize the delivery time. The primary feature of packet routing resides in its fine-grained per-packet forwarding policy. No information regarding the network traffic is shared between neighboring nodes. In contrast, exiting protocols use flooding approaches either to maintain a globally consistent routing table (e.g., DSDV [10]), or to construct an on-demand flow level routing table (e.g., AODV [9]). The packet routing is essential to meet the dynamically changing traffic pattern in today's communication networks. Meanwhile, it symbolizes the difficulty of designing fully distributed forwarding policy that strikes a balance of choosing short paths and less congested paths through learning with local observations.

Reinforcement learning (RL) is a bio-inspired machine learning approach that acquires knowledge by exploring the interaction with local environment without the need of external supervision [1]. Therefore, it is suitable to address the routing challenge in distributed networks where each node (interchangeable with router) measures the per-hop delivery delays as the reward of its actions and learns the best action accordingly. Boyan and Littman [5] proposed the first multiagent Q-learning approach for packet routing in a generalized network topology. This straightforward routing policy achieves much smaller mean delivery delay compared with the benchmark shortest-path approach. Xia *et al.* [33] applied dual RL-based Q-routing approach to improve convergence rate of routing in cognitive radio networks. Lin and van der Schaar [3] adopted the joint Q-routing and power control policy for delay sensitive applications in wireless networks. More applications of RL-based routing algorithms can be found in [11]. Owing to the well-known "curse of dimensionality" [14], the state-action space of RL is usually small such that the existing RL-based routing algorithms cannot take full advantage of the history of network traffic dynamics and cannot explore sufficiently more trajectories before deciding the packet forwarding. The complexity of training RL with large state-action space becomes an obstacle of deploying RL-based packet routing.

The breakthrough of deep RL (DRL) provides a new opportunity to a good many RL-based networking applications that are previously perplexed by the prohibitive training burden. With deep neural network (DNN) as a powerful approximator

Manuscript received November 17, 2019; revised January 18, 2020; accepted July 13, 2020. This work was supported in part by the Natural Science Foundation of China under Grant 61772139; in part by the Shanghai–Hong Kong Collaborative Project under Grant 18510760900; in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010166003; in part by China Mobile Chengdu Institute of Research and Development; and in part by the Open Project of State Key Laboratory of Novel Software Technology (Nanjing University) under Grant KFKT2019B03. A preliminary part of this work was invited to RAWNET workshop collocated with WiOpt'2019 [48]. This article was recommended by Associate Editor T. Li. *(Xinyu You and Xuanjie Li are co-first authors.) (Corresponding author: Yuedong Xu.)*

Xinyu You, Xuanjie Li, Yuedong Xu, and Hui Feng are with the School of Information Science and Technology, Fudan University, Shanghai 200237, China (e-mail: xyyou18@fudan.edu.cn; xuanjieli16@fudan.edu.cn; ydxu@fudan.edu.cn; hfeng@fudan.edu.cn).

Jin Zhao is with the School of Computer Science, Fudan University, Shanghai 200237, China (e-mail: jzhao@fudan.edu.cn).

Huaicheng Yan is with the Key Laboratory of Advanced Control and Optimization for Chemical Process of Ministry of Education, East China University of Science and Technology, Shanghai 200237, China, and also with the College of Mechatronics and Control Engineering, Hubei Normal University, Huangshi 435002, China (e-mail: hcyan@ecust.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2020.3012832

of Q-table, the network designer can leverage its advantages from two aspects: 1) the neural network can take much more information as its inputs, enlarging the state-action space for better policy making and 2) the neutral network can automatically abstract invisible features from high-dimensional input data [17], thus achieving an end-to-end decision making yet alleviating the handcrafted feature selection technique. Recent successful applications include cloud resource allocation [21], adaptive bitrate video streaming [22], and cellular scheduling [23]. DRL is even used to generate routing policy in [24] against the dynamic traffic pattern that is hardly predictable. However, Stampa *et al.* [24] considered a centralized routing policy that requires the global topology and the global traffic demand matrix, and operates at the flow-level. Inspired by the power of DRL and in view of the limitations of Q-routing [5], we aim to make an early attempt to develop fully distributed packet routing policies using multiagent DRL.

In this article, we propose a novel multiagent DRL algorithm named deep Q-routing with communication (DQRC) for fully distributed packet routing. Each router uses a carefully designed long short term memory (LSTM) RNN to learn and infer the dynamic routing policy independently. DQRC takes the high-dimensional information as its input: the destinations of head-of-line (HOL) as well as backlogged packets, the action history, and the queue length of neighboring routers that are reported periodically. The action of DQRC is the next-hop of the HOL packet, and the packet delivery time is chosen as the reward of Q-learning so as to train the LSTM neural network at each router. The intuitions of our design stand for twofold relation to the queueing process. On one hand, the action history is closely related to the congestion of next hops, the number of backlogged packets indicates the load of the current router, and knowing the destinations of outgoing packets avoids pumping them into the same adjacent routers. On the other hand, with a lightweight communication scheme, an agent can acquire the queue information of its direct neighbors and learn to send packets to less congested next hops. With such a large input space, existing RL approaches such Q-routing [5] cannot handle the training online so that the training of DNNs using RL rewards becomes a necessity. DQRC is fully distributed in the sense that each router is configured with an independent neural network for parameter update and decision making. This differs from the recent multiagent DRL learning framework in other domains [6] where the training of neural networks are simultaneous and globally consistent. The training of multiagent DRL is usually difficult (e.g., convergence and training speed), while DQRC proves the feasibility of deploying DRL-based packet routing in the dynamic environment.

With Q-routing [5] and Backpressure [34] as benchmarks, our experimental results reveal a few interesting observations. First, DQRC significantly outperforms the other two representative algorithms in terms of the average delivery delay with different network loads and topologies. With careful examination of the DQRC routing policy, we observe that each router makes adaptive routing decision by considering more information than the destination of the HOL packet, thus avoiding congestion on "popular" paths. Second, the lightweight communication mechanism is very beneficial to DQRC. An agent only shares its raw queue length other than a host of DNN parameters with its neighbors, and this sharing can be infrequent or delayed for several time slots with merely gentle increase of delivery time. Third, DQRC is robust to the hyper-parameters of the LSTM neural network, indicating that a moderate complexity of neural networks (e.g., three hidden layers and 128 neurons in a hidden layer) is sufficient.

The remainder of this article is organized as follows. Section II reviews the background knowledge of RL, DRL, and partially observable Markov decision processe (POMDP). Section III presents our design of DQRC. The delivery delay of the proposed algorithm are evaluated in Section IV with Q-routing and Backpressure as the benchmark. Section V is devoted to making discussions about future study and challenges. Section VI concludes this work.

II. Background and Literature

In this section, we briefly review the traditional routing algorithms, RL and DRL techniques, and their applications to routing problem. We then put forward the necessity of fully distributed learning for real-world routing problem. Finally, the background of POMDP is included.

A. Traditional Routing Algorithm

As a profound and globally applicable routing algorithm, shortest-path algorithm reveals how to transfer all the data as quickly as possible. Within all the algorithms in shortest-path class, Bellman–Ford algorithm [39] and Dijkstra's algorithm [40] are crucial to the development of network protocols. The definition of a shortest path may vary with different contexts, such as transmission delay or a number of hops in the network. It is easy to understand that along the shortest path between two nodes, data packets can be delivered costing the least amount of time provided that there is no congestion along the route. However, these assumptions are unrealistic for the real network. When the network load is heavy, the shortest-path algorithm will lead to severe backlogs in busy routers and thus necessitates manual monitoring and adjustment.

Backpressure [34] has shown its efficiency in dynamic traffic routing in multihop network by using congestion gradients. Each node maintains different queues for all of the potential destination nodes and each packet is assigned to one queue according to its destination. The basic idea of Backpressure is to utilize the differential of the queue backlogs between the current node and its neighbor nodes as the pressure to drive packet transmission. Backpressure has been widely studied in the literature because of its simplicity and asymptotic optimality at heavy traffic regime, but there are still some drawbacks hindering its wider application: backpressure may suffer from poor delay performance particularly in light load, in which case there is not enough pressure to push data toward the destination and therefore packets may choose unnecessary longer paths and even loops.

B. RL Routing Algorithm

Based on the mapping relationship between observed state and execution action, RL aims to construct an agent to maximize the expected discounted reward through the interaction with the environment following Markov chain [43]–[45]. Without prior knowledge of which state the environment would transition to or which actions yield the highest reward, the learner must discover the optimal policy by trial-and-error.

The first attempt to apply RL in the packet routing problem is Q-routing algorithm, which is a variant of Q-learning [1]. Since Q-routing is essentially based on multiagent approach, each node is viewed as an independent agent and endowed with a Q-table to restore Q-values as the estimate of the transmission time between that node and the others. With the aim of shortening average packet delivery time, agents will update their Q-table and learn the optimal routing policy through the feedback from their neighboring nodes when receiving the packet sent to them. Despite the superior performance over shortest-path algorithm in dynamic network environment, Q-routing suffers from the inability to fine-tune routing policy under heavy network load and the inadequate adaptability of network load change. To address these problems, other improved algorithms have been proposed, such as PQ-routing [7] which uses previous routing memory to predict the traffic trend and DRQ-routing [8] which utilizes the information from both forward and backward exploration to make better decisions.

C. DRL Routing Algorithm

DRL embraces the advantage of DNNs [41], [42] to the training process, thereby improving the learning speed and the performance of RL [4]. One popular algorithm of DRL is deep Q-learning (DQL) [25], which implements a deep Q-network (DQN) instead of Q-table to derive an approximation of Q-value with special mechanisms of experience replay and target Q-network.

Recently, network routing problems are solved using DRL under different environment and optimization targets. Based on the control model of the agent, these algorithms can be categorized as follows.

Class 1 (Single-Agent Learning): Single-agent algorithm treats the network controller as a central agent which can observe the global information of the network and control the packet transmission at each router. Both the learning and execution process of this kind of algorithm are centralized [28], in other words, the communication between routers are not restricted during training and execution.

SDN-Routing [24] presents the first attempt to apply single-agent DRL in the routing optimization of traffic engineering. The traffic demand which represents the bandwidth request between each source-destination pair is viewed as the environment state. The network controller determines the transmission path of packets to achieve the objective of minimizing the network delay. Another algorithm [20] considers a similar network model while taking minimum link utilization as the optimization target.

Class 2 (Multiagent Learning): Cooperative control in multiagent systems provides an alternative way to solve a complicated problem that are hard to be performed for one agent [26], [27]. In multiagent learning, each router in the network is treated as a single agent which can observe partial environment information and take actions according to its own routing policy.

The first multiagent DRL routing algorithm is DQN-routing [6] that is the combination Q-routing and DQN. Each router is regarded as an agent whose parameters are shared by each other and updated at the same time during training process (centralized training), but it provides independent instructions for packet transmission (decentralized execution). The comparison with contemporary routing algorithms confirms a substantial performance gain.

Nevertheless, algorithms with centralized learning process stated above are hard to be applied in the realistic network. The centralized learning controller is usually unable to gather environment transitions from widely distributed routers once an action is executed somewhere and to update the parameters of each neural network simultaneously caused by the limited bandwidth [31], [32].

Accordingly, for better application in real-world scenario, the routing algorithm we propose is executed in a fully distributed way [19], which means both the training process and the execution process are decentralized. Under these circumstances, each agent owns its unique neural network with independent parameters for policy update and decision making, thereby avoiding the necessity for the communications among routers in the process of environment transition collection and parameter update.

D. POMDP

In the real-world environment, it is rare that agents can observe the full state of the system. This type of problem can be modelled as POMDPs where the observation agents receive is only the partial glimpse of the underlying system state [35]. The partial observation of system state will aggravate the nonstationarity problem in multiagent RL (MARL).

To tackle with this problem, an efficient solution is to utilize deep recurrent Q-network (DRQN) [35] which is capable of capturing invisible features based on time-continuous observations. Derived from DQN, this ingenious neural network architecture substitutes some fully connected layers by LSTM [36] layers which can not only store and reuse history information to explore temporal features but also split inputs into several parts to search connections between them. The LSTM layer contains special units called memory blocks in which memory cells can store the temporal state of the network and gates are used to control the flow of information [37]. The existence of LSTM helps alleviate the dilemma that the present task cannot receive the relevant information long before it.

Another promising direction is to design a cooperation or communication mechanism to help agents learn from not only its own knowledge but also public information broadcast from other agents. Two MARL algorithms are put forward in [28] to address partially observable multiagent decision making

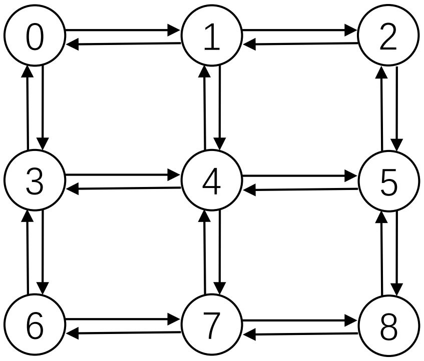


Fig. 1. 3×3 network topology.

problems: reinforced interagent learning (RIAL) and differentiable interagent learning (DIAL). The former uses independent DQN, while the latter backpropagates error derivatives through communication channels during learning. With centralized learning but decentralized execution, the communication between agents promise the success in MARL domain.

Inspired by these ideas, we propose our algorithm DQRC with fully distributed multiagent DRL. DQRC amalgamates traditional DQN with LSTM architecture as well as communication mechanism to address POMDPs in packet routing problem.

III. Design

We establish the mathematical model of the packet routing problem and describe the representation of each element in the RL formulation. We then propose a novel deep RNN architecture and the corresponding training algorithm.

A. Mathematical Model

We consider a network consisting of routers and links connected between them, and packet routing is the transmission of packets from a source to its destination through intermediate nodes. We now present the mathematical model of the packet routing problem.

Network: The network is modeled as a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ are defined as finite sets of nodes and transmission links between them, respectively. A simple network topology can be found in Fig. 1, containing 9 nodes and 12 pairs of bidirectional links. Each packet is originated from node s and destined for node $d : s, d \in \mathcal{N}$ and $s \neq d$ with randomly generated intervals.

Routing: The mission of packet routing is to transfer each packet to its destination through the relaying of multiple routers. The queue of routers follows the first-in first-out (FIFO) criterion. Each router n constantly delivers the HOL packet to its neighbor node v until that packet reaches its termination.

Target: The packet routing problem aims at finding the optimal transmission path between source and destination nodes based on some routing metric, which, in our experiment, is defined as the average delivery time of packets. Formally, we denote the packet set as $\mathcal{P}$ and the total transmission time as t_p for each packet $p : p \in \mathcal{P}$. Our target is to minimize the average delivery time $T = \sum_{p \in \mathcal{P}} t_p / K$, where K denotes the number of packets in $\mathcal{P}$.

B. Reinforcement Learning Formulation

Packet routing can be modeled as an MARL problem with POMDPs [30]. Each node is an independent agent and learns its routing policy by observing the local network state and communicating with its neighbor nodes. Therefore, we will describe the definitions of each element in RL for a single agent.

State Space: The packet p to be sent by agent n is defined as *current packet*. We denote the state space of agent n as $S_n : \{d_p, E_n, C_n\}$, where d_p is the destination of the current packet, E_n is some extra information related to agent n, C_n is the information shared form the neighbor nodes of agent n. At different time steps, the state observed by the agent is time varying due to the dynamic change of network traffic.

Action Space: The action space of agent n is defined as $A_n : \mathcal{V}_n$, where $\mathcal{V}_n$ is the set of neighbor nodes of node n. Accordingly, for each agent, the size of action space equals to the number of its adjacent nodes. For example, node 4 in Fig. 1 has four candidate actions. Once a packet arrives at the head of queue at time step t, agent n observes the current state $s_t \in S_n$ and picks an action $a_t \in A_n$, and then the current packet is delivered to the corresponding neighbor of node n.

Reward: We craft the reward to guide the agent toward effective policy for our target: minimizing the average delivery time. The reward at time step t is set to be the sum of queueing time and transmission time: $r_t = q + l$, where the former q is the time spent in the queue of agent n, and the latter l is referred to as the transmission latency to the next hop.

C. Deep Neural Network

We will introduce the architecture of the neural network designed for our algorithm in this part. Note that, in the formulation of fully distributed RL, each node is an individual agent and therefore possesses its own neural network for decision making. Accordingly, the following description of the neural network architecture is tailored for a single agent.

As shown in Fig. 2, we build a deep RNN with three fully connected layers and one LSTM layer. The input of the neural network can be classified into four parts.

1) *Current Destination:* The destination node of the current packet.
2) *Action History:* The executed actions for the past k packets sent out just before the current packet.
3) *Future Destinations:* The destination nodes of the next m packets waiting behind the current packet.
4) *Max-Queue Node:* The node which has the longest queue among all the neighbor nodes of the current node.

In the formulation of the state space in Section III-B, the current node corresponds to d_p, the action history and the future destinations corresponds to E_n, the max-queue node corresponds to C_n. The intuitions of our design stand for twofold relation to the queueing process. On one hand, the

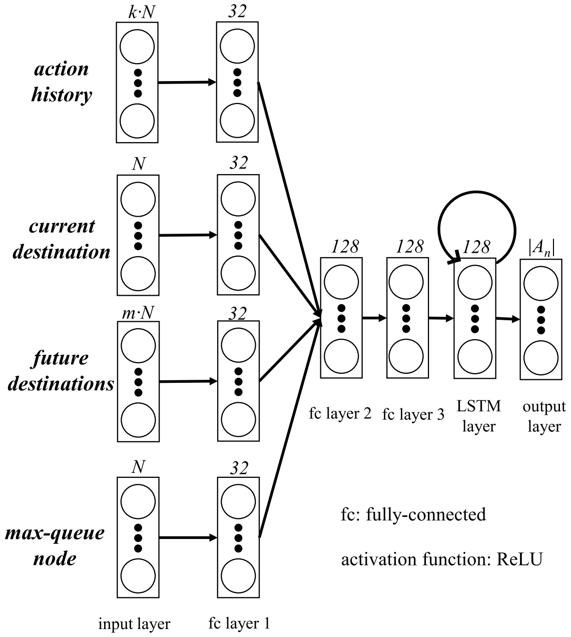


Fig. 2. LSTM neural network with ReLU activation.

action history is closely related to the congestion of next hops, and knowing the destinations of the coming outgoing packets avoids pumping them into the same adjacent routers. On the other hand, by integrating the max-queue node in the state, the agent can learn to avoid sending packets to such nodes and switch to less congested next hops.

Before being input into the neural network, all of the above information will be processed with one-hot encoding. Take the first part of input information as an example, if the current packet is destined to node 4, the one-hot encoding result of current destination is [000010000]. Therefore, the number of input neurons equals to $(1+k+m+1)\times N$, where N is defined as the total number of nodes in the network topology.

The first hidden layer is a concatenation of four subsets of hidden neurons. Each subset possesses 32 neurons and is fully connected with the corresponding part of the input neurons independently. Following the first hidden layer, another two hidden layers with 128 neurons are added.

In partially observable environment, each agent can only receive an observation s which is correlated with the full environment state. Inspired by DRQNs [35], we add an LSTM layer to maintain an internal state and aggregate observations over time. Despite the partial observation s, the hidden state of the agent h will be included to represent Q-value which is defined as $Q(s, h, a)$.

Furthermore, the size of the output layer and the agent's action space $|A_n|$ are identical, and the value of each output neuron is the estimated Q-value of the corresponding action. With this change of the representation for Q-value, we try to update the parameter of neural networks instead of the value of the Q-table.

We use rectified linear unit (ReLU) as the activation function and root mean square prop (RMSProp) as the optimization algorithm.

Algorithm 1 DQRC

```
Input: current destination $d_p$, action history and future destinations $E_n$, max-queue node $C_n$
Output: the neighbor node to transmit the current packet $a_t$
// initialization
for agent $i = 1, N$ do
    Initialize replay buffer $D_i \leftarrow \varnothing$
    Initialize Q-network $Q_i$ with random weights $\theta_i$
end for
// training process
for episode = 1, $M$ do
    for each decision epoch $t$ do
        Assign current agent $n$ and packet $p$
        Observe local information $d_p$ and $E_n$
        Collect shared information $C_n$
        Integrate current state $s_t : \{d_p, E_n, C_n\}$ and hidden state $h_t$
        Select and execute action
            $a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ argmax_a Q_n(\theta_n) & \text{with probability } 1-\epsilon \end{cases}$
        Forward $p$ to next agent $v_t$
        Calculate and collect reward $r_t$
        Observe next state $s_{t+1}$ and next hidden state $h_{t+1}$
        Set transmission flag $f_t = \begin{cases} 1 & v_t = d_p \\ 0 & \text{otherwise} \end{cases}$
        Store transition $(s_t, h_t, r_t, v_t, s_{t+1}, h_{t+1}, f_t)$ in $D_n$
        Sample a random batch $(s_j, h_j, r_j, v_j, s_{j+1}, h_{j+1}, f_j)$ from $D_n$
        Set $y_j = r_j + \gamma max_{a'} Q_{v_j}(s_{j+1}, h_{j+1}, a'; \theta_{v_j})(1-f_j)$
        $\theta_n \leftarrow GradientDescent\ ((y_j - Q_n(s_j, h_j, a_j; \theta_n))^2)$
    end for
end for
```

D. Learning Algorithm

By integrating Q-routing and DRQN, we propose the packet routing algorithm with multiagent DRL, where both training and execution process are set decentralized. The pseudo-code of the learning algorithm, which we call DQRC, is shown in Algorithm 1, in which the initialization and the training process are identical for each node.

Each node i is treated as an individual agent and possesses its own neural network Q_i with particular parameter θ_i to estimate the state-action value function $Q_i(s, h, a; \theta_i)$, which represents the expected delivery time for a packet to reach the destination when the agent executes action a in state s and hidden state h. Replay memory D_i with capacity of 100 is also initialized independently for each agent to restore its environment transitions, and from it a random min-batch with size of 16 will be sampled for the update of its network parameters.

For each decision epoch t when a packet p arrives at the head of line of a certain node n, agent n will observe local information d_p and E_n and collect shared information C_n

through the communication with neighbor nodes. By integrating current state $s_t : \{d_p, E_n, C_n\}$ and hidden state h_t, agent n will execute an action a_t based on ϵ-greedy policy, which means agent n will choose a random action from its action space A_n with probability ϵ or choose the action with the highest Q-value with probability $1 - \epsilon$. The assignment of a_t is given by

$$a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \text{argmax}_a Q_n(s_t, h_t, a_t; \theta_n) & \text{with probability } 1 - \epsilon. \end{cases} \tag{3.1}$$

Then the current packet p is forwarded to the corresponding neighbor node v_t, and the reward r_t is calculated and sent back to agent n. Current state and hidden state will transition to s_{t+1} and h_{t+1}, respectively. Besides, the transmission flag f_t will be set to 1 if the next node v_t matches the packet's destination d_p or set to 0 otherwise. The assignment of f_t is given by

$$f_t = \begin{cases} 1 & v_t = d_p \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

After the feedback of these information, agent n will record this transition $(s_t, h_t, r_t, v_t, s_{t+1}, h_{t+1}, f_t)$ into its replay memory D_n. Different from the sequential update process of DRQN, a training batch $(s_j, h_j, r_j, v_j, s_{j+1}, h_{j+1}, f_j)$ is sampled randomly from D_n to avoid violating the DQN random sampling policy. As a result of the unstable environment caused by the multiagent characteristic, the remaining delivery time τ that packet p is expected to spend from v_t to d_p need to be recalculated before the training process. τ is given by

$$\tau = \max_{a'} Q_{v_j}\left(s_{j+1}, h_{j+1}, a'; \theta_{v_j}\right). \tag{3.3}$$

At the end of the decision epoch t, the gradient descent method is used to fit the neural network $Q_n(\theta_n)$. Since the function of each agent's neural network Q_i is to estimate the expected delivery time of a packet from node i to the destination node, we set the target value y_j to the sum of the transmission time between the current node and the next node (represented by the reward r), and the transmission time from the next node to the destination node (represented by the output of the next agent's Q-network Q_v). The target value y_j can be expressed as

$$y_j = r_j + \gamma\tau\left(1 - f_j\right) \tag{3.4}$$

where γ is the discount factor. The Bellman optimality condition [1] can be written as

$$Q^*(s, a) = \mathbb{E}_{s'}\left[R\left(s, a, s'\right) + \gamma \max_{a'} Q^*\left(s', a'\right)\right] \tag{3.5}$$

where Q^* is the optimal Q-value and R is the immediate reward. In order to satisfy the Bellman optimality criteria and adjust estimated Q-value Q_n toward optimal Q-value Q^*, we update the parameters of $Q_n(\theta_n)$ by minimizing the mean squared error between the target Q-value y_j and current Q-network output. The loss function L_t is given by

$$L_t = \left(y_j - Q_n\left(s_j, h_j, a_j; \theta_n\right)\right)^2. \tag{3.6}$$

After differentiating the loss function L_t with respect to the weights θ_n, we can update θ_n by performing gradient descent across all transition samples in the training batch

$$\theta_n \leftarrow \theta_n + \alpha\nabla_{\theta_n}\left(y_j - Q_n\left(s_j, h_j, a_j; \theta_n\right)\right)^2 \tag{3.7}$$

where α is the learning rate. In this way, the network parameters of each agent are updated with episodic training until convergence.

IV. Evaluation

We conducted several experiments in the simulation environment of the computer network to evaluate the performance of DQRC in both online and offline training mode. Our experiments cover a broad set of network conditions and topologies. Then we will give an explanation for the comparison results from the perspective of input information and learned policy. Furthermore, a deep dive into DQRC will be made to test its robustness to various parameters.

A. Simulation Environment

We now describe the settings of our simulation environment and another two algorithms to be compared with DQRC.

Topology: The topology of the network we used is the same as Fig. 1, which remains static in the whole experiment. Despite the simple structure, we can explore new insights into packet routing, and actually a more complex network will lead to similar results. All the nodes and links in the network share the same attributes: the buffer size of each node is unlimited and the bandwidth of each link equals to the packet size, in which case only a single packet can be transmitted at a time.

Packet: A certain proportion, named *distribution ratio*, of packets are generated from node 0 (busy ingress-router) to node 8 (busy egress-router), while the other packets' source and destination are chosen uniformly. Packets are introduced into the network with the same size and their generated intervals are fixed at a certain value which is inversely proportional to the load of network traffic.

Time Setting: The time during the simulation is measured by milliseconds. The transmission time between adjacent nodes a packet has to spend is set to 1.0 ms. The performance criterion of the experiment is the average delivery time of packets within a certain period.

Benchmark Algorithms: Both Q-routing and Backpressure algorithms stated in Section II are included for performance comparison. Since packet routing problem deals with the transmission of one specific packet instead of data stream, OSPF will be tailored to traditional shortest path algorithm. In addition, [5] has proved the inferior performance of shortest path algorithm to Q-routing, and therefore it will not be taken for comparison. In our experiment, in order to follow the FIFO criterion, the traditional Backpressure algorithm is tailored to route the HOL packet to the neighbor node with the minimal number of packets with the same destination.

Parameter Settings: The random exploration parameter ϵ is set to 0.1, which indicates a higher probability of exploiting current behavior policy and a lower probability of exploring more actions. In order to approximate the optimum point

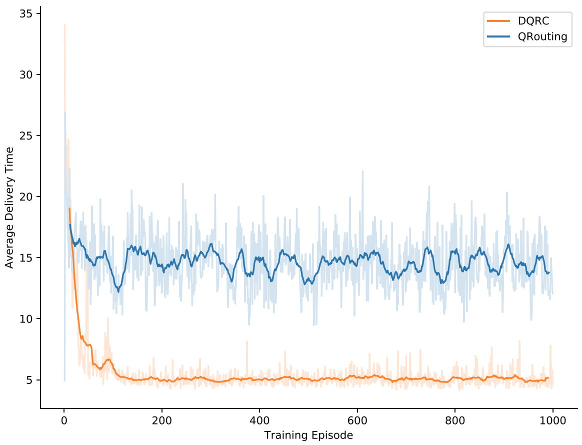


Fig. 3. Offline training speed comparison.

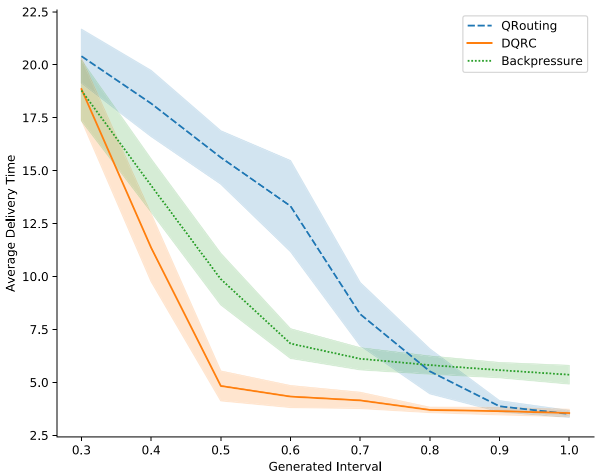


Fig. 4. Offline test with different network loads.

quickly, we set the learning rate α to 0.01 during the first ten iterations. And then for better convergence, the learning rate decrease with time until it reaches 0.0001 at iteration 1000. Since packet routing can be viewed as an episodic task [1] which ends with finite time, we set the discount factor γ to 1.0, which means the agents will treat the immediate reward and future rewards equally.

B. *Offline-Training Online-Test Result*

In offline experiments, we generated a fixed packet series containing 1000 packets as the training set on which we trained the neural network and Q-table separately. Instead of ending with all the packets routed to their destinations, each training episode stops in 100 ms, in order to meet the characteristic of endless packet in online environment. After 1000 training episodes, we restored the well-trained models to compare their performance in a new test environment where packets were generated at the corresponding network load level but different source-destination pairs from the training set. Note that Backpressure does not need training and can be applied to test directly.

Training Speed: With the fixed packet sequence whose generated interval is 0.5 ms and distribution ratio is 70%, we trained DQRC and Q-routing and compared their training speeds. Fig. 3 shows the variation trend of average packet delivery time along with the training episode. The solid line is the smoothed result of the true value depicted in the shadowed part. Though performing worse in the first few episodes, DQRC convergences quickly and keeps stable at a lower delay level. On the contrary, Q-routing fluctuates violently from time to time and never converges. From the perspective of either convergence speed or final training result, DQRC outperforms Q-routing in our simulation.

Network Load: In terms of the distribution ratio at 70%, we carried out online tests with various packet generated intervals ranging from 0.3 ms to 1.0 ms and recorded 50 sets of result for the first 100 ms under each parameter. The average value and standard deviation of these resulting data are depicted with curve and shadow separately in Fig. 4. As expected, we can clearly see the following.

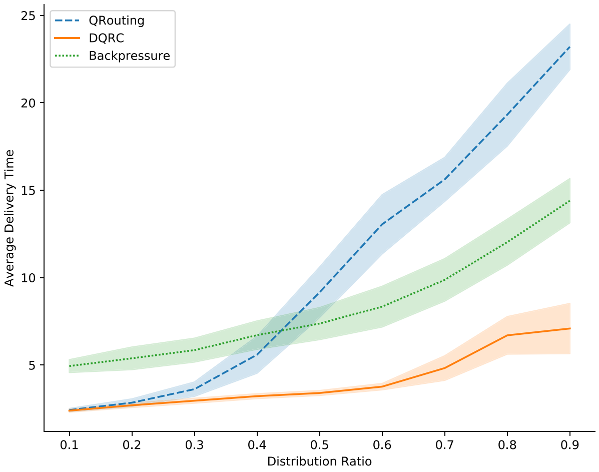


Fig. 5. Offline test with different distribution ratios.

1) All three algorithms present a rising tendency on average delivery time as the load becomes heavier. DQRC outperforms the other two algorithms at all network load levels.
2) Q-routing behave equivalently with DQRC when the generated interval is between 0.9 ms and 1.0 ms (low network load). Conversely, Backpressure has fairly good performance when the generated interval is between 0.4 ms and 0.8 ms (high network load). This interesting phenomenon may reveal the fact that DQRC has the merits of Q-routing and Backpressure while discarding their shortcomings.
3) The width of the shadow band of DQRC is the narrowest, meaning that our proposed algorithm performs more stable during the test and is robust to randomly generated packet set.

Distribution Ratio: We collected another 50 sets of results from online tests by adjusting the distribution ratio from 10% to 90% and fixing the generated interval at 0.5 ms. Similarly, each test ends in 100 ms. From the result diagram shown in Fig. 5, we can obtain more appealing discoveries.

1) As the distribution ratio continues increasing, the average delivery time of all three algorithms is prolonged because larger distribution ratio indicates more pressure on the links connnetting node 0 and node 8.
2) The distribution ratio for Q-routing and Backpressure with good performance is 10%–40% and 50%–90%, respectively, but DQRC works well in either case. This result draws the same conclusion that DQRC possesses the features of Q-routing and Backpressure as that stated in the above part.
3) Considering DQRC, there is almost no shadow area around the solid line before the distribution ratio reaches 60%, indicating the good robustness to the randomness of packet generation.
4) The slope of Q-routing is the largest, showing that Q-routing has higher sensitivity to the change of distribution ratio than the other two algorithms. By contrary, DQRC and Backpressure are more robust to this kind of traffic change.

In summary, no matter in which kind of test environment, DQRC integrates the merits of both Q-routing and Backpressure and performs better in the following three aspects.

1) The average delivery time of DQRC is the shortest regardless of generated interval or distribution ratio.
2) DQRC is more adaptable and resistant to the change of network load and spatial distribution of packet generation.
3) DQRC frees itself from random errors caused by the network disturbances.

C. Online-Training Online-Test Result

In online simulation environment, packets are generated all the time following the regulations described in Section IV-A. The simulation timeline is split into intervals of 100 ms and for each interval the average delivery time of transmitted packets is recorded. In order to test each algorithm's adaptability to the change of traffic pattern, we initially set the generated interval of packets to 1.0 ms and suddenly change it to 0.7 ms at time 4000 ms and reset it to 1.0 ms at time 8000 ms. Before put into online test, DQRC and Q-routing are well-trained in offline training environment with generated interval set to 1.0 ms, and during the simulation the parameters of neural networks and the value of the Q-table are updated from time to time. Note that Backpressure does not need training.

Fig. 6 plots the average result of 50 online tests with different packets sets. We can clearly see the following.

1) Q-routing have comparable performance with DQRC in the first and the third stage. However, when the network load begin to increase at 4000 ms, Q-routing find it hard to converge to a new routing policy to address the growing number of packets, leading to larger transmission delay and unstable performance.
2) Though behaving fairly well in the second stage, Backpressure has an inferior performance compared to DQRC and Q-routing in lightly loaded network due to insufficient pressure to push packets to their destinations.

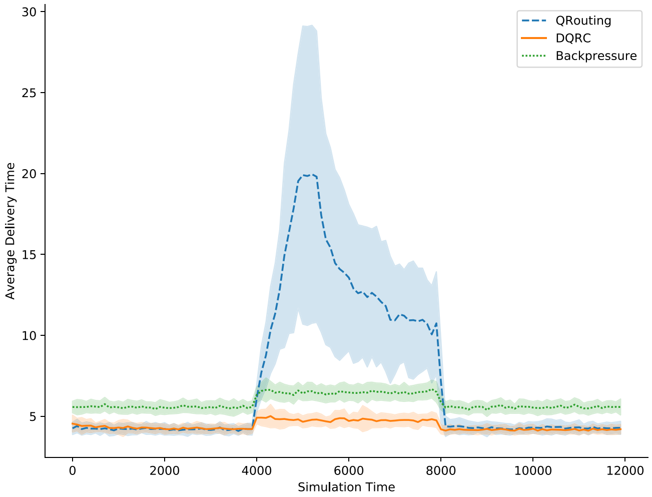


Fig. 6. Online test result with changing network loads.

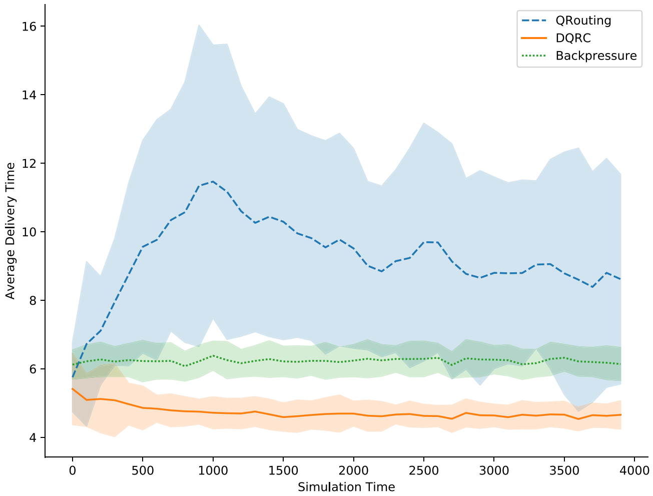


Fig. 7. Online test result with changing network loads.

3) DQRC achieves the shortest delivery time and the lowest fluctuations in all three stages.

Furthermore, we conduct another experiment where the generated interval varies with time and is uniformly distributed on (0.5, 1.0) ms. From the simulation result shown in Fig. 7, we find that DQRC performs the best all the time in terms of transmission delay, and therefore is more adaptable to dynamic changes of network load.

D. Complex Topology

In the above experiments, the 3 × 3 topology in Fig. 1 is symmetrically designed and each node has no more than four neighbor nodes. To test the scalability of our proposed algorithm, we expand the network scale to a 25-nodes and 56-edges topology depicted in Fig. 8. The topology is taken from topology zoo [38] and represents real AT&T North America network. Each vertex represents a city, for example node 0 is New York, and each edge is a bidirectional link. The connection of routers becomes complex and each router has more choices to make, thus increasing the difficulty of decision making. The attributes of the new topology are the

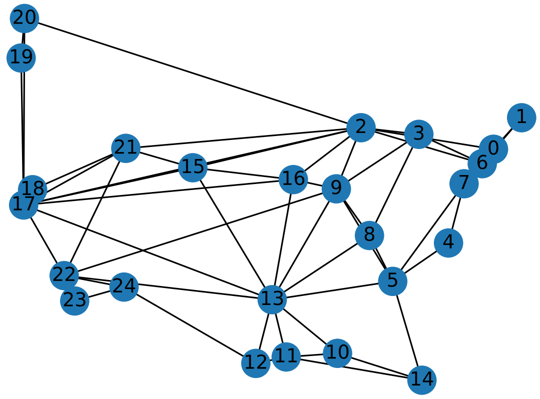


Fig. 8. AT&T North America topology.

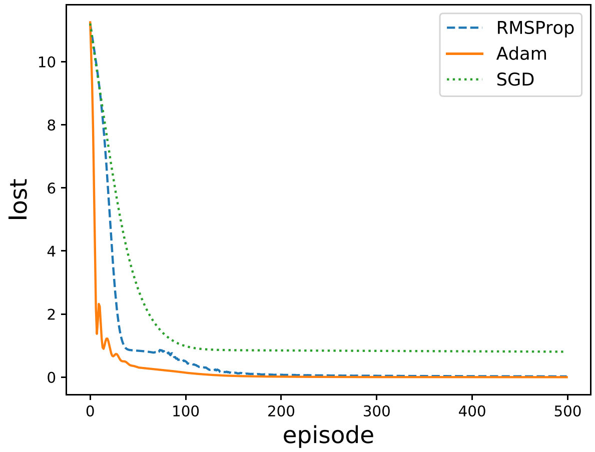


Fig. 9. Pretraining result with different optimization algorithms.

same as those in Section IV-A except that node 17 is viewed as the busy ingress-router and node 8 is viewed as the busy egress-router.

In multiagent DRL, the addition of more agents will increase the convergence difficulty of neural networks. To address this problem, we introduce the technique of pretraining. In the neural network designed in Section III-C, the value of each output neuron represents the estimated remaining delivery time after choosing the corresponding neighbor node. We take shortest-path algorithm as an initial guide to grasp the topology information. In detail, first we calculate the lengths of the shortest path between each source-destination pair; then we treat these lengths as labels and craft states by editing only the current destination and setting the other three part of input information to zeros; finally, we perform supervised learning to train the neural network before putting it into the process of RL.

We conducted pretraining with three optimization algorithms: stochastic gradient descent (SGD), Adam, and RMSProp. Fig. 9 depicts the average lost change of all the agents with respect to the episode of pretraining. We can clearly see that, with Adam or RMSProp as the optimization algorithm, the average lost decrease sharply in the first few episodes, and after about 200 episodes, the neural network of each nodes converges with average lost close to zero. However, pretraining with SGD optimization converges to a higher level with slower speed. The main reason is that Adam is an adaptive learning rate optimization algorithm that computes individual learning rates for different parameters. Adam can be viewed as a combination of SGD with momentum which uses moving average of the gradient and RMSProp which scales the learning rate by squared gradients. It shows great performance gains in terms of training speed and convergence point, and therefore we choose Adam as the final optimization algorithm in the pretraining process.

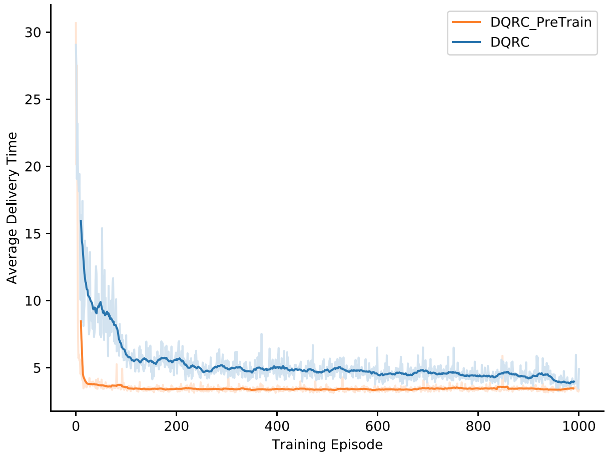


Fig. 10. Comparison between DQRC with and without pretraining.

Fig. 10 shows the performance gap between DQRC with and without pretraining during the RL training process. We find that, in such a complex topology with 25 nodes, DQRC finds itself takes a very long period converge with random initialization. However, after the implementation of pretraining, DQRC reaches a fairly good performance in the initial state and converges quickly to a low delay level.

In line with the experiment setting in Section IV-B, we execute online-tests in two cases: 1) fixing the distribution ratio and changing the network load level with different packet generated intervals and 2) fixing the generated intervals and changing the distribution ratio. The simulation results are illustrated in Figs. 11 and 12, respectively. Unsurprisingly, in this complicated topology, the variation trends of the average delivery time with respect to the network load level and distribution ratio have fairly consistency with those in the 3×3 topology. The similar results in both topologies demonstrate good robustness of DQRC.

E. Performance Analysis

With the results in both offline-training online-test and online-training online-test tests with different topologies, DQRC can be regarded as a combination of Q-routing and Backpressure, achieving not only shorter average delivery time but also greater stability in dynamically changing networks. We will clarify the main reasons for this result from the perspective of the routing policy learned by each algorithm. To simplify our interpretation, the following analysis is based on the 3×3 topology (Fig. 1).

Why Does DQRC Possess the Features of Q-Routing and Backpressure? From Figs. 4 and 5, we can see that in both

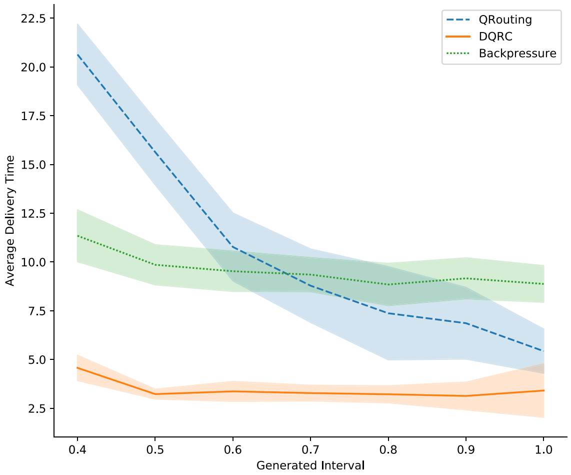


Fig. 11. Offline test with different network loads in AT&T network.

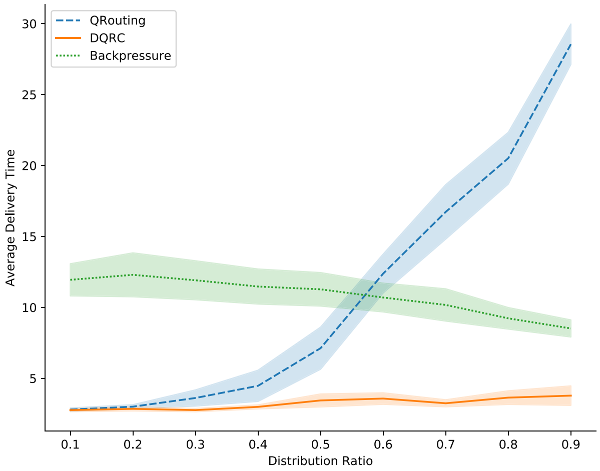


Fig. 12. Offline test with different distribution ratios in AT&T network.

cases DQRC resembles Q-routing in light network load and Backpressure in heavy network load. This is an inevitable result owing to the neural network structure. In the anterior description of neural network design in Section III-C, the deep RNN of DQRC is constructed on the basis of Q-table in Q-routing, which is used to store Q-values for each state-action pair. Therefore, DQRC reserves the ability to estimate the remaining delivery time for all possible decisions. When the network is relatively light-loaded, both Q-table and neural network can give the agent an accurate estimate of Q-value to help with decision making. But this consonance will be disturbed by the rise of network load, in which case the traits of Backpressure become useful. Different from Q-routing, agents in DQRC are no longer isolated without communication with each other, and in contrast, they will exchange their queue information with adjacent agents. With this extra information, agents in DQRC can be aware of the pressure in the queue of neighbor nodes, thus avoiding conveying too many packets to busy nodes. DQRC has grasped the core idea of Backpressure which has been proved its excellent performance especially in heavy traffic pattern.

What Are the Advantages of DQRC? In addition to time delay estimation with neural network and information sharing between neighbor nodes, we also introduce new techniques into DQRC: 1) as described in Section III-C, we take full advantage of the representation ability of the neural network by expanding the input dimension. With the introduction of history actions and future destinations, DQRC can make more accurate estimate of the Q-value and 2) with the LSTM layer in neural network, DQRC can capture invisible connections among input elements and temporal features of sequential decisions.

Whenever the agents of Q-routing choose an action, the only information they can utilize is the destination of the current packet, leading to the same routing decision for packets with the same destination and a static path between each source-destination pair. During the training process of Q-routing, when congestion occurs on some link, the Q-value of the corresponding action will decrease, and agents will choose another action with higher Q-value. The congestion will be lessened on the previous link, but shortly afterwards, the newly selected link becomes congested. Accordingly, the congestion alternation among links causes severe performance fluctuation of Q-routing in Fig. 3. As for DQRC, with additional information as the network input, the agents of DQRC can execute different but effective routing policy for each packet despite the same destination. The flooding of packets will be shared by different links, leading to stable training performance.

More precisely, we evaluate the case where the generated interval and distribution ratio are set to 0.1 ms and 70%, respectively, in online test. The network load is so heavy that a large number of packets are waiting in the queue of node 0 to be transmitted to node 8. For these packets, the agent of node 0 has two choices: sending them to node 1 or node 3. The well-trained agent of node 0 of Q-routing follows the shortest path and therefore all those packets will be sent to node 1. Under this strategy, serious congestion will occur unavoidably in the links 0→1, 1→2, and 1→4, which eventually lead to longer delivery time. However, DQRC can overcome this difficulty cleverly. Before making decisions for every packet destined for node 8 at node 0, the agent will first collect the information about the actions the last five packets have taken and the nodes the next five packets are destined for and then check the queue length of node 1 and node 3 with communication. For example, when the last five packets were sent to node 1, the agent will decide to send the current packet to node 3 to avoid long queuing delay. Similarly, after some packets were sent to node 3, the agent will change its policy and decide to transfer the packet through node 1 again. Therefore, the great packet burden can be shared by node 1 and node 3 in DQRC instead of being born only by node 0 in Q-routing.

Of course, it is possible that different kinds of information will lead to various decisions, so agents need to train themselves to judge which kind of information is more crucial and needs more attention. After the self-adaptation in training process, the astute agents have the ability to grasp the dynamic changes in the network and adjust their routing policies accordingly, which, shown in our test result, can gain shorter average delivery time and a better performance.

TABLE I
TEST WITH DIFFERENT NUMBER OF HIDDEN LAYERS

Number of hidden layers (between fc1 and LSTM layer)	Average delivery time
0	4.28
1	4.17
2	4.11
3	4.33
random	4.24

TABLE II
TEST WITH DIFFERENT NUMBER OF NEURONS

Number of neurons	Average delivery time
32	5.26
64	4.96
128	4.11
256	4.71
random	4.82

TABLE III
TEST WITH DIFFERENT INTERVALS OF INFORMATION SHARING

Intervals of information sharing (ms)	Average delivery time
0	4.11
1	4.32
2	4.75
3	4.89
4	5.19
5	5.35

F. Deep Dive Into DQRC

In this part, we are committed to exploring the intrinsic disciplines within DQRC by evaluating its performance under the following situations: 1) we change the architecture of the neural network with different quantities of neurons and dense layers; 2) we modify the communication interval of information sharing among agents; 3) we rule out a certain element of DQRC, including extra information, shared information, and LSTM layer; and 4) we set the network topology with random labels. Note that the following experiments are conducted with setting generated interval at 0.5 s and distribution ratio at 70%,

Neural Network Architecture: Starting with the original architecture (Fig. 2), we altered a range of neural network parameters to understand their impact on the performance of DQRC. First, after fixing the number of neurons in each hidden layer at 128, we varied the number of hidden layers between the first hidden layer (fc1) and the LSTM layer in the architecture of DQRC. The comparison results are shown in Table I. We find that the default setting yields the best performance, but the delay gap between it and the others is very small, showing DRQC's great robustness to the depth of neural network. Then with the default layer number, we varied the number of neurons in each hidden layer and LSTM layer. The changes of these parameters are synchronous, i.e., when 64 neurons are used, the neuron number of each sublayer in the first hidden layer is 16. Results from this alteration are presented in Table II. We can see that the number of neurons have higher influence on DQRC's performance. Too few or too many neurons will lead to performance degradation. In conclusion, carefully choosing hyper-parameters of neural network is needed for better estimation.

In our previous design, all the neural networks share the same structure. In order to test the stability of DQRC, we allow agents possess different neural network architectures. We allocate each agent with random number of hidden layers or random number of neurons in each hidden layer and LSTM layer. The results are shown in Tables I and II, respectively. We can find that, though each agent is equipped with different neural networks, the performance of DQRC will degrade slightly compared with the optimal network setting but is fairly superior to the benchmarks. We can clarify this result from the perspective of fully distributed learning. Each agent possesses an individual neural network for decision making, and the parameters will not be shared among agents. The learning process of agents will not be influenced by each other. Besides, the experiment result above shows that DQRC has fairly good robustness against the structure of the neural network. Accordingly, agents with different neural networks can still learn accurate mapping relationship between the input and output.

Communication Interval: In our original design, each agent can collect queue lengths of neighbor nodes immediately whenever making a decision. However, when applying DQRC to a network with delayed feedback, the time delay during the transmission of shared information may impede the adoption of DQRC in practice. Therefore, we modified the communication interval of information sharing among agents to test its influence on DQRC's performance. With communication interval ranging from 1 ms to 5 ms, the test result is shown in Table III. We find that the average delivery time steadily rises as we increase the communication interval. Despite slight performance degradation, the average delivery time of DQRC is still less than half of that of Backpressure, showing DQRC's feasibility in non real-time network.

Element in DQRC: In order to identifies the rationality of each element in DQRC, we put forward another three algorithms on the basis of DQRC: 1) DQRC without communication: we forbid the communication process in DQRC and eliminates shared information in the state space; 2) DQRC without LSTM: we delete the LSTM layer in the original neural network (Fig. 2); and 3) DQR: we include only current destination into the input information.

The comparison result with different algorithms is shown in Fig. 14. We can see that the lack of extra information or shared information will lead to remarkable performance degradation. If not equipped with LSTM layer, the average delivery time of DQRC will increase by 10%. As for DQR and Q-routing, the

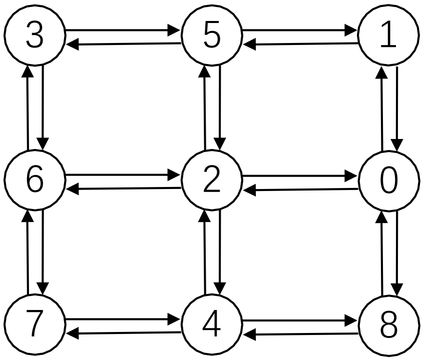


Fig. 13. 3×3 network topology with random labels.

only difference between these two algorithms is the representation of Q-value but the learning algorithms are identical. The neural network which is merely the approximation of Q-table would not help with the estimate of the accurate transmission time between the source and termination of packets. As a result, DQR and Q-routing who share the same input have comparable performance.

One-Hot Encoding: Since we encode the node ID with one-hot encoding method, all the elements in the one-hot vector are "0" except one "1." As shown in Fig. 13, we set the original network topology with random labels which are entirely different from the ones in Fig. 1. With the other environment settings unchanged, we run DQRC in Figs. 1 and 13 separately, and the result is shown in Fig. 14. We find that the performance gap is negligible and therefore the change of node labels makes little impact on the performance of DQRC. This phenomenon can be clarified from the perspective of neural network architecture. As stated in Section III-C, the number of input neurons is the multiple of the total number of nodes in the network topology, and one input neuron corresponds to one node. The change of the routers' labels will only lead to the exchange of positions of input neurons in the neural network, while the independence of input features after one-hot encoding will not be disturbed. Consequently, the mapping relationship between the input and output remains the same, and therefore the system is robust against the change of labels.

V. Discussion

In this section, we put forward our research plan and ensuing challenges in several directions, deriving from some limitations of the current work.

Other DRL Algorithms: The routing algorithm we propose is based on DQN [25], which is a classical but simple form of DRL. Thanks to the tremendous contribution researchers in the community of DRL have made, more effective DRL algorithms can be leveraged in packet routing. For example, as the optimization of the policy gradient-based RL algorithm, TRPO [12] is combined with the neural network in continuous control domain to ensure monotonic performance [13]. Besides, based on DPG [15], an off-policy actor-critic algorithm DDPG [16] uses the neural network as a differentiable function approximator to estimate action-value function, and then updates the policy parameters in the direction of the deterministic policy gradient.

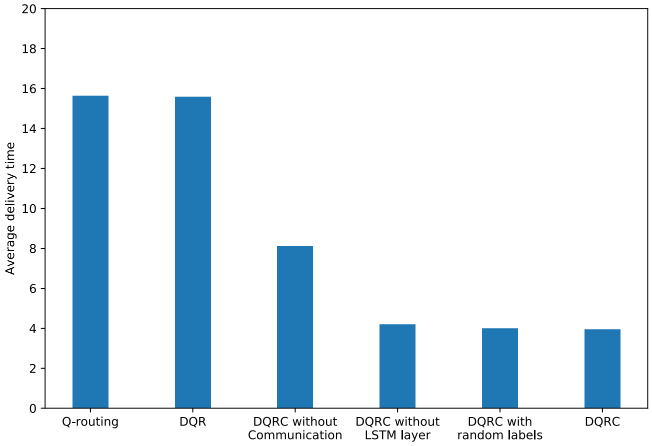


Fig. 14. Test with different algorithms.

Realistic Simulation Environment: In this article, the experiments we have conducted in the simulation environment are based on some restrictive conditions, as described in Section IV-A, which would impede the adoption of our proposed routing algorithms in the realistic network with complex traffic patterns. In future work, we will consider a more general network setting, such as routers with finite queue and packets with varied sizes. A couple of uncertain elements like link breakage or node failure can also be taken into account. Furthermore, NS-3 network simulator [2] can be utilized as the standard platform to test the performance of routing algorithms. When implementing our proposed algorithm in real-world networks with high link rate, reducing the time consumed by decision making plays a critical part in achieving real-time inference. The design of sparse LSTM network [46] and the implementation of specialized FPGAs [47] are promising candidate solutions to this challenge.

Multiagent Learning: The packet routing system in our current implementation is built on the multiagent model, where each router is treated as an independent agent and learns asynchronously. However, in multiagent environment, the inherent nonstationarity problem [18] during the learning process will be magnified after the application of DNN. Besides the information-sharing mechanism stated in our paper, importance sampling and fingerprint [29] provide alternative solutions to address this problem. Moreover, witnessing the benefits of cooperative learning [28], we will analyse the performance boost that the local coordination of DNNs (e.g., parameter sharing and memory sharing [28]) can yield in our future study.

VI. Conclusion

We presented a fully distributed packet routing algorithm DQRC based on multiagent DRL. We designed a deep RNN with proper input to make more accurate estimation for the transmission delay. With information sharing mechanism, each agent in DQRC can learn adaptive routing policy through the interaction with environment. From the preliminary experiment results, we find that, compared with Q-routing and Backpressure, DQRC can reduce the average packet delivery time to a considerable extent in different traffic patterns and topologies.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[2] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.

[3] Z. Lin and M. van der Schaar, "Autonomic and distributed joint routing and power control for delay-sensitive applications in multi-hop wireless networks," *IEEE Trans. Wireless Commun.*, vol. 10, no. 1, pp. 102–113, Jan. 2011.

[4] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," 2018. [Online]. Available: arXiv:1810.07862.

[5] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in Neural Information Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann, 1994, pp. 671–678.

[6] D. Mukhutdinov, A. Filchenkov, A. Shalyto, and V. Vyatkin, "Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system," *Future Gener. Comput. Syst.*, vol. 94, pp. 587–600, May 2019.

[7] S. P. Choi and D.-Y. Yeung, "Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control," in *Advances in Neural Information Processing Systems*. San Mateo, CA, USA: Kaufmann, 1996, pp. 945–951.

[8] S. Kumar and R. Miikkulainen, "Dual reinforcement Q-routing: An on-line adaptive routing algorithm," in *Proc. Artif. Neural Netw. Eng. Conf.*, 1997, pp. 231–238.

[9] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," Internet Eng. Task Forece, RFC 3561, 2003.

[10] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, pp. 234–244, Oct. 1994.

[11] H. A. Al-Rawi, M. A. Ng, and K.-L. A. Yau, "Application of reinforcement learning to routing in distributed wireless networks: A review," *Artif. Intell. Rev.*, vol. 43, no. 3, pp. 381–416, 2015.

[12] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[13] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.

[14] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[15] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mech Learn. (ICML)*, 2014, pp. 387–395.

[16] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, [Online]. Available: arXiv:1509.02971.

[17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," 2017. [Online]. Available: arXiv:1708.05866.

[18] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," 2017. [Online]. Available: arXiv:1707.09183.

[19] Y. Xu, Z.-G. Wu, Y.-J. Pan, C. K. Ahn, and H. Yan, "Consensus of linear multiagent systems with input-based triggering condition," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 11, pp. 2308–2317, Nov. 2019.

[20] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proc. 16th ACM Workshop Hot Topics Netw.*, 2017, pp. 185–191.

[21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[22] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Spec. Interest Group Data Commun.*, 2017, pp. 197–210.

[23] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, 2017, pp. 1–6.

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

[25] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] R. Yang, H. Zhang, G. Feng, H. Yan, and Z. Wang, "Robust cooperative output regulation of multi-agent systems via adaptive event-triggered control," *Automatica*, vol. 102, pp. 129–136, Apr. 2019.

[27] Y. Zhang, H. Li, J. Sun, and W. He, "Cooperative adaptive event-triggered control for multiagent systems with actuator failures," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 9, pp. 1759–1768, Sep. 2019.

[28] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2016, pp. 2137–2145.

[29] J. Foerster *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn. Vol. 70*, 2017, pp. 1146–1155.

[30] G. E. Monahan, "State of the art—A survey of partially observable Markov decision processes: Theory, models, and algorithms," *Manag. Sci.*, vol. 28, no. 1, pp. 1–16, 1982.

[31] Z.-G. Wu, Y. Xu, Y.-J. Pan, P. Shi, and Q. Wang, "Event-triggered pinning control for consensus of multiagent systems with quantized information," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 11, pp. 1929–1938, Nov. 2018.

[32] R. Yang, H. Zhang, G. Feng, and H. Yan, "Distributed event-triggered adaptive control for cooperative output regulation of heterogeneous multiagent systems under switching topology," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 9, pp. 4347–4358, Sep. 2018.

[33] B. Xia, M. H. Wahab, Y. Yang, Z. Fan, and M. Sooriyabandara, "Reinforcement learning based spectrum-aware routing in multi-hop cognitive radio networks," in *Proc. 4th Int. Conf. Cogn. Radio Orient. Wireless Netw. Commun.*, Hannover, Germany, 2009, pp. 1–5.

[34] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," in *Proc. 29th IEEE Conf. Decis. Control*, Honolulu, HI, USA, 1990, pp. 2130–2132.

[35] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Series*, 2015, pp. 29–37.

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc.*, 2014, pp. 338–342.

[38] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[39] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958.

[40] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[41] X. Zhao, H. Yang, and G. Zong, "Adaptive neural hierarchical sliding mode control of nonstrict-feedback nonlinear systems and an application to electronic circuits," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 7, pp. 1394–1404, Jul. 2017.

[42] M. Chen, "Constrained control allocation for overactuated aircraft using a neurodynamic model," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 12, pp. 1630–1641, Dec. 2016.

[43] Y. Wang, X. Yang, and H. Yan, "Reliable fuzzy tracking control of near-space hypersonic vehicle using aperiodic measurement information," *IEEE Trans. Ind. Electron.*, vol. 66, no. 12, pp. 9439–9447, Dec. 2019.

[44] Y. Wang, H. R. Karimi, H.-K. Lam, and H. Yan, "Fuzzy output tracking control and filtering for nonlinear discrete-time descriptor systems under unreliable communication links," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2369–2379, Jun. 2020.

[45] Y. Wang, W. Zhou, J. Luo, H. Yan, H. Pu, and Y. Peng, "Reliable intelligent path following control for a robotic airship against sensor faults," *IEEE/ASME Trans. Mechatronics*, vol. 24, no. 6, pp. 2572–2582, Dec. 2019.

[46] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2017, pp. 75–84.

[47] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," Microsoft Res., Redmond, WA, USA, White Paper, 2015.

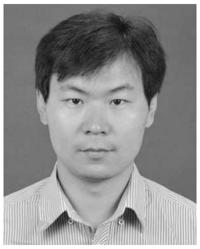
[48] X. You, X. Li, Y. Xu, H. Feng, and J. Zhao, "Toward packet routing with fully-distributed multi-agent deep reinforcement learning," in *Proc. IEEE Int. Symp. Model. Optim. Mobile Ad Hoc Wireless Netw. (WiOPT)*, Avignon, France, 2019, pp. 1–8.

Xinyu You received the B.Eng. degree in electronic and information engineering from the School of Electronic Information, Wuhan University, Wuhan, China, in 2018. He is currently pursuing the M.S. degree in circuits and systems with Fudan University, Shanghai, China.

His research interests include reinforcement learning, applications of machine learning for computer networks, and edge computing.

Xuanjie Li received the B.Eng. degree in electronic and information science and technology from the School of Information Science and Technology, Fudan University, Shanghai, China, in 2020.

Her research interests include network routing, optimization and reinforcement learning, and especially the application of nonconvex optimization in computer networks. She is awarded the Fudan Junzheng Scholar in 2019.



Yuedong Xu received the B.S. degree in automation from Anhui University, Hefei, China, in 2001, the M.S. degree in automatic control from the Huazhong University of Science and Technology, Wuhan, China, in 2004, and the Ph.D. degree in computer science from the Chinese University of Hong Kong, Hong Kong, in 2009.

He is a Tenured Associate Professor with the School of Information Science and Technology, Fudan University, Shanghai, China. From 2009 to 2012, he was a Postdoctoral Researcher with INRIA, Sophia Antipolis, France, and Université d'Avignon, Avignon, France. His areas of interests include performance evaluation, optimization, security, data analytics and economic analysis of communication networks, and mobile computing.

Hui Feng (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in electronic engineering from Fudan University, Shanghai, China, in 2003, 2006, and 2014, respectively.

He is currently an Associate Professor with the Department of Electronic Engineering, Fudan University. His research focuses on signal processing and robotics.

Jin Zhao (Senior Member, IEEE) received the B.Eng. degree in computer communications from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2001, and the Ph.D. degree in computer science from Nanjing University, Nanjing, in 2006.

He is currently an Associate Professor with Fudan University, Shanghai, China. In 2014, he stayed with the University of Massachusetts Amherst, Amherst, MA, USA, as a Visiting Scholar for one year. His research interests include software-defined networking and network function virtualization.

Huaicheng Yan (Member, IEEE) received the B.Sc. degree in automatic control from the Wuhan University of Technology, Wuhan, China, in 2001, and the Ph.D. degree in control theory and control engineering from the Huazhong University of Science and Technology, Wuhan, in 2007.

In 2011, he was a Research Fellow with the University of Hong Kong, Hong Kong, for three months, and also a Research Fellow with the City University of Hong Kong, Hong Kong, in 2012, for six months. He is currently a Professor with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. His research interests include networked control systems, multiagent systems, and robotics.

Prof. Yan is an Associate Editor of the *International Journal of Robotics and Automation* and the IEEE OPEN JOURNAL OF CIRCUITS AND SYSTEMS.