Relational Deep Reinforcement Learning for Generalizable Routing in Wireless Networks

Victoria Manfredi, Alicia P. Wolfe, Cheonjin Park, *Student Member, IEEE,* Xiaolan Zhang, *Member, IEEE,* Sushirdeep Narayana, Dongjin Song, *Member, IEEE,* Bing Wang, *Senior Member, IEEE*

***Abstract*—While routing in wireless networks has been studied extensively, existing protocols are typically designed for a specific set of network conditions and so do not easily accommodate changes in those conditions. In this paper, we develop a distributed routing approach based on deep reinforcement learning. In this approach, although the routing policies are trained offline in a centralized setting, they run in a fully distributed manner during testing and are able to generalize to network conditions unseen during training, including diverse network sizes, topologies, traffic patterns, congestion levels, and link dynamics. We make the following key innovations in our design: (i) the use of *relational features* as inputs to the deep neural network approximating the decision space, which enables our algorithm to generalize to unseen network conditions; (ii) the use of *packet-centric decisions* to transform the routing problem into an episodic task by viewing *packets*, rather than wireless devices, as reinforcement learning agents, which provides a natural way to propagate and model rewards accurately during learning, and allows for decoupling of forwarding decisions among agents; (iii) the use of *extended-time actions* to model the time spent by a packet waiting in a queue, which reduces the amount of training data needed and allows the learning algorithm to converge more quickly; and (iv) the use of *sampling-based training* to further improve training efficiency. We evaluate our routing algorithm using a packet-level simulator and show that our policies trained on a grid topology generalize to other grid topologies, as well as to a real-world network topology and associated traffic matrices. In addition, our policies outperform both classical and RL-based algorithms during testing in terms of packet delivery ratio, delay, and forwards per packet.**

***Index Terms*—Wireless networks, routing, reinforcement learning, deep neural networks**

I. Introduction

Efficient routing in multi-hop wireless networks is challenging due to network dynamics caused by interference from the environment, the shared nature of the wireless medium, variability in the propagation of wireless signals, device and link failures, and the lack of a centralized coordinator. While many routing algorithms have been developed for wireless networks (see §III), they typically assume operation under very specific network conditions.

In this paper, we ask the following question: *Can we design a generalizable routing algorithm that seamlessly adapts to different network conditions?* In other words, we seek to design a routing algorithm that works regardless of traffic pattern, congestion level, network size, connectivity, or link dynamics. Such an algorithm is desirable since the optimal routing algorithm may be very different depending on the network conditions. For example, consider a wireless network that fluctuates between periods of low and high congestion. In this scenario, backpressure routing [2] can achieve throughput-optimal performance during the high congestion periods, but leads to performance inferior to many other algorithms during the low congestion periods.

One way to design an adaptive routing algorithm is to determine the set of target network conditions to handle, identify the appropriate routing algorithm for each, and then switch algorithms as needed. This approach risks instability if network conditions change frequently or if a routing algorithm takes a long time to converge [3]. Reinforcement learning (RL) [4] allows for an alternate approach in which an RL agent trained on a set of conditions learns to make routing decisions in an uncertain and time-varying environment. Q-routing [5] is the first RL-based routing algorithm. Since then, many more have been proposed (see surveys [6], [7] and §III)). Recently, advances in deep reinforcement learning (DRL), which uses deep neural networks (DNNs) to approximate the decision space, have motivated the design of DRL-based routing algorithms (see references in [8]–[10]).

While training a DRL algorithm in one scenario and testing it in the same scenario often leads to excellent performance, the problem is significantly more challenging when training and testing scenarios differ [11]. For instance, existing DRL-based routing strategies do not easily generalize to other scenarios because they encode assumptions about the network topology and the number of possible actions in the DNN used to make routing decisions (see §III). In this work, we develop a novel DRL-based routing approach, *Relational DRL (RDRL)*, that uses *relational features* to enable generalizion to diverse network conditions.

We make the following contributions to DRL-based routing design for multi-hop wireless networks:

• *Centralized training and distributed execution (CTDE).* We introduce a novel approach to *offline* centralized training but *online* distributed execution for packet routing. Our use of offline centralized training combines packet information from all nodes to train a single DNN model in a sequence of rounds, and then, once all rounds have finished, each node in the network uses the trained model in a fully distributed manner

Manuscript received xx xx, 2024.

V. Manfredi, A. P. Wolfe, and S. Narayana are with Wesleyan University. X. Zhang is with Fordham University. C. Park, D. Song and B. Wang are with the University of Connecticut. A preliminary version of this paper [1] appeared in IEEE WoWMoM 2021. This research was supported in part by the National Science Foundation under grants 2154191 and 2154190. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

to make routing decisions during testing. Our methodology differs from the approaches in recent related work [12]–[14] (see details in §III). It also differs from those for cooperative games [15]–[18] in that multi-hop routing involves multiple nodes to deliver a packet from a source to a destination, but the nodes do not work cooperatively to achieve a common goal.

• *Relational features.* To enable our algorithm to scale to larger networks and generalize to diverse network conditions, we input relational features to the DNN model used to approximate the routing decision space. This not only allows data from all nodes in the network to be used to train a single DNN model that encodes the routing policy, but also allows for distributed execution during testing where each packet independently uses the local copy of the DNN at each node for routing decisions.

• *Packet-centric decisions.* We transform the routing problem into an episodic task [1] by viewing *packets*, rather than devices, as the DRL agents that must learn a routing policy. This packet-centric approach provides a more natural way to propagate and model rewards accurately and allows for decoupling of routing decisions among agents.

• *Extended-time actions.* We use extended-time actions, also known as options [19], to model the time spent by a packet waiting in a queue, which reduces the amount of training data needed and allows learning to converge more quickly.

• *Sampling-based training.* Extending our basic offline training approach [1], we develop a sampling-based training approach, to significantly improve training efficiency (up to $5\times$ less computation), while not impacting routing performance during testing.

We evaluate our approach using a packet-level network simulator, with our source code available at https://gitlab.com/vmanfred/drl_routing_stationary. Using two models trained in a grid topology with 64 nodes, we show that they are generalizable to grid topologies with 16 to 81 nodes, in the presence or absence of link dynamics. We further demonstrate that these two grid-trained models work well on a real-world network topology, the GÉANT network [20] and associated real-world traffic matrices [21]. Our approach significantly outperforms shortest path routing [22], backpressure routing [2], and an online RL-based approach [23] with respect to fraction of packets delivered, packet delivery delay, and forwards per packet.

The rest of this paper is organized as follows. In §II, we present the problem setting and give background on RL and DRL. In §III, we describe related work on routing in wireless networks. In §IV, we present our RDRL algorithm. In §V, we describe our offline centralized training and online decentralized testing methodology. Our training and evaluation settings and results are presented in §VI and §VII, respectively. Finally, in Section VIII, we provide conclusions and future work.

[1]In RL, an episodic task has a clear start and end, while a continuing task does not. In a network setting, from the viewpoint of a node, routing is a continuing task, but from the viewpoint of a packet, routing is an episodic tasks that starts with the creation of a packet and terminates when that packet is delivered or dropped.

II. Problem Setting and Background

A. Routing in Multi-hop Wireless Networks

Consider a multi-hop wireless network with a set of stationary nodes, i.e., wireless devices, V. Let $N = |V|$ denote the number of nodes in the network. Each node transmits via a wireless channel. Two nodes that are within transmission range can communicate with each other. Even though nodes are stationary, due to interference and possible link dynamics, the neighbors of a node can still change over time.

Each node can buffer a maximum of B packets. A node can originate, receive, or relay packets for other nodes in the network. The routing decision at a node v is to choose a neighbor to which to send a packet from v's queue. We assume each node makes routing decisions in fixed time intervals, referred to as *time steps* [2]. At the beginning of each time step, a node is aware of all of its neighbors (i.e., all nodes within its transmission range), which can be achieved using a neighbor discovery mechanism.

Three potential goals of routing in multi-hop wireless networks are: (i) *maximize throughput*, i.e., the packet delivery rate, (ii) *minimize delay*, i.e., the time from a packet being generated at its source to being delivered to its destination, and (iii) *minimize the number of forwards* made to deliver packets to reduce the amount of resource usage overhead (e.g., energy consumption) of the network. Each of these goals makes different performance trade-offs. For instance, more aggressive forwarding can lead to lower delay for delivered packets, but also more forwards and hence higher resource usage. In certain cases, e.g., when the network is already congested, more aggressive forwarding may cause even more congestion, and thus even higher delay and lower delivery rate (due to packet drops when node buffers are full).

A *static* routing strategy that ignores congestion can lead to poor performance when a network's traffic load changes, or the network's topology changes in such a way as to cause changes in the traffic load. For instance, shortest path routing, which selects the path between a source and destination solely based on the number of hops, leads to low throughput when the shortest path is congested.

In contrast, an *adaptive* routing algorithm can lead to either high throughput with large delay, or low throughput with small delay. One example is backpressure routing [2], which routes packets dynamically based on the amount of congestion in the network. When per-node buffer size, B, is large, backpressure routing leads to high throughput at the cost of large delay, since packets flow through the network following the lowest congestion gradient at each node, and may take many hops to reach the destination. When B is small, backpressure leads to lower delay at the cost of low throughput since many packets can be dropped. In general, it is difficult to achieve both high throughput and low latency simultaneously under dynamic traffic conditions.

[2]The time steps of the nodes do not need to be synchronized. Rather, each node can make its own routing decisions independently. In §VII, for simplicity only, we assume that all nodes make routing decisions at the same time steps. Our scheme can be used in asynchronized settings.

B. Background: RL and DRL

The goal of reinforcement learning (RL) [4] is to derive an optimal policy of the underlying Markov decision process (MDP). A Markov decision process is a fully observable, stochastic environment with a Markovian transition model and additive rewards. An MDP comprises a set of states $\mathcal{S}$, a per state set of actions $\mathcal{A}(s)$, $\forall s \in \mathcal{S}$, a reward function, and a state transition function. State transitions are assumed to be Markovian: the probability of the next state $s' \in \mathcal{S}$ depends only on the current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$. In RL, the agents are modeled as an MDP, but now the agents lack knowledge of both the transition function and the reward function. Instead, it is assumed that agents have access to an environment. In this environment, agents balance *exploration* with *exploitation* to estimate an optimal policy based on interactions with the environment. Agents receive rewards as feedback in response to actions taken, and transition to different next states in the environment.

Let $Q(s,a)$ be the Q-value for state-action pair, (s,a), which estimates the expected future return for an agent when starting in state s and taking action a. Intuitively, $Q(s,a)$ represents how good an action a is for state s. To learn the Q-value function, the agent observes (s,a,r,s') at each time step, where r is the immediate reward. The Q-value function is then updated through Q-learning [24] as

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \cdot \max_{a' \in \mathcal{A}(s')} Q(s',a') - Q(s,a) \right] \quad (1)$$

where $0 \leq \gamma \leq 1$ is a *discount factor*, which indicates the relative value of immediate vs. future rewards, and $0 < \alpha \leq 1$ is the *learning rate*. Both the discount factor and the learning rate can be set to be a constant or vary over time [4]. Once learned, the *optimal* action in state s is the one with the highest Q-value.

In the update of $Q(s,a)$ in (1), define

$$y \triangleq r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s',a') \,, \quad (2)$$

where y is the predicted Q-value, referred to as the *Temporal Difference (TD) target*. Then the update to $Q(s,a)$ in (1) is equivalent to a weighted sum:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha y \,. \quad (3)$$

DRL. When the MDP has a small number of states and actions, an RL agent can learn a Q-value function directly as shown above. When the state space is too large for exact computation of the Q-values, function approximation may be used to find approximate Q-values. In particular, deep learning approaches, e.g., deep neural networks (DNNs) [25], can be used for function approximation. In this case, the resultant RL model is then referred to as a DRL model.

III. Related Work

A. Traditional Routing in Wireless Networks

The literature on routing strategies for wireless networks is extensive, ranging from early protocols such as DSR [26], AODV [27], and OLSR [28], to backpressure routing [2], to protocols suited for resource constrained sensor [29] and IoT networks [30]. Most of these protocols, however, are designed for specific scenarios and so do not generalize well to scenarios with different network conditions. For instance, DSR, AODV, and OLSR are designed for mobile ad hoc networks, with no specific treatment of network congestion, while backpressure routing is particularly suitable for networks with congestion, and routing strategies for sensor and IoT networks must often manage unique traffic patterns, such as data being sent from many source devices to one sink device.

In contrast, in this paper, we focus on designing a DRL-based routing algorithm with the explicit goal of generalizing the learned routing strategy to network scenarios unseen in training. In §VII, we compare our strategy with two representative traditional strategies, backpressure routing and shortest path routing, which are respectively optimal in terms of throughput and delay depending on the level of network congestion. We also compare our approach with an an online RL-based approach [23] described in the next section.

B. RL and DRL-based Routing

We classify existing approaches that use RL or DRL for routing into multiple categories based on key design decisions. We also describe into which categories our work falls and the relationship of our work with other works.

Centralized vs. distributed decision-making. While routing inherently involves multiple entities interacting (i.e., the individual nodes in a network), the decision-making by these entities can be either centralized or distributed. In centralized decision-making, a central entity determines the routing decisions for every node in the network, while in distributed decision making, each node makes routing decisions autonomously using its own local model or algorithm.

As an example, centralized decision-making is often used in software-defined networks (SDNs), where an SDN controller gathers the information of the entire network and determines the forwarding table for each switch (e.g., [31]–[35]). In such cases, the input to the decision may involve global network information, such as the traffic matrix of the entire network or per link statistics (loss, delay, and load), which are difficult to obtain in distributed settings such as a multi-hop wireless network. In distributed decision-making, each node makes decisions independently using its own routing model (e.g., [12], [13], [36]). Such strategies are more suitable for multi-hop wireless networks since no centralized controller exists in these networks.

In this paper, our RDRL algorithms use *distributed decision-making*. That is, each node autonomously uses a local copy of the DRL model to make routing decisions in a fully distributed manner.

Online vs. offline training. An RL or DRL model can be trained online, i.e., training is done directly while operating in a deployed network, or offline, i.e., training is done beforehand and then the resulting trained model is used in the deployed network without further training. Q-routing [5], the first RL-based routing protocol, uses online training, as

do its subsequent variants (see a summary of the variants in [23]). In Q-routing specifically, each node maintains a table of estimated Q-values and uses Q-learning to update the table after receiving updates from one of its neighbors. A recent study [23] points out that Q-routing and its subsequent variants are algorithms (not network protocols), and so develops and evaluates a protocol based on Q-routing related approaches. The study in [12] extends Q-routing by introducing a Deep Q-Network (DQN) [37] and evaluates its performance in both routing and baggage handling systems.

Online training has the advantage that no prior training in the network environment is needed, but it requires sufficient stationarity in the network topology and traffic for the routing protocol to converge, and that convergence takes time. Thus, when the network environment changes in terms of topology or traffic, reconvergence is needed. Some approaches therefore incorporate an offline pre-training phase, e.g., the study in [12] uses the link-state routing protocol in pre-training, before performing online training of the DRL model. Similarly, a supervised-learning based pre-training phase is used in [13]. Therefore, these approaches can be viewed as hybrid approaches that utilize both offline and online training.

In this paper, *we use offline training in a packet-driven simulator to train a DRL model, and then use the trained model directly in testing without further training*, following the CTDE [17], [18], [38] paradigm. Offline training allows us to easily adopt *centralized training* by logging the packet information of the entire network into a file (see §IV), and use parameter sharing to train a single DRL model, which is then copied to each node to be used locally in a distributed manner during testing[3].

Differences in training vs. testing scenarios. While the CTDE paradigm has been used by state-of-the-art multi-agent DRL studies [17], [18], [38], this paradigm has not been extended to settings where training and testing are performed on different scenarios. The recent studies in [12]–[14] that use DRL for distributed packet routing also use the same scenarios for both training and testing.

In this paper, our RDRL algorithms use *relational features to achieve generalizability to scenarios unseen in training* and thereby address a key challenge for the CTDE paradigm. That is, we train a DRL model offline on one scenario in such a way that it can be used in new testing scenarios unseen in training, including different numbers of nodes, topologies, link dynamics, and traffic conditions.

Most closely related work. The studies that are closest to our work are those using RL or DRL for distributed packet routing, which makes routing decisions for every packet (instead of flows) using only local information. These studies include Q-routing, variants of Q-routing (see earlier), and more recent studies that use a much larger state space [12]–[14]. Like our study, these more recent studies also use DRL, but as mentioned earlier, they use the same scenarios for both training and testing, while the primary goal of our study is developing a DRL-based approach that can generalize to unseen testing scenarios. Indeed, the features to their DNNs include node IDs and other network specific values. Therefore, a model trained in one network cannot be used for another network that has a different size or topology, since node IDs do not generalize well.

We are not aware of any existing work on RL or DRL-based distributed packet routing that shares the same goal as ours, i.e., training a model beforehand and then using it in unseen testing scenarios without any further training. In §VII, we compare our RDRL schemes with two traditional routing algorithms (they are optimal in congested and light-traffic scenarios, respectively), along with an RL-based routing protocol. This RL protocol is based on Q-routing and its variants, inspired by [23], and we refer to it as *Q-protocol*. Specifically, Q-protocol uses Distance Vector-like message exchange among neighbors, which provides much more up-to-date information for decision-making and significantly outperforms Q-routing in our network scenarios (see more details in §VI-C). Q-protocol uses online training, and hence can be used in unseen testing scenarios, comparable to our approach.

Enhancements to our prior work. A preliminary version of this work is [1]. We have substantially enhanced the preliminary version by developing an efficient sampling-based training methodology that further improves the scalability of our approach. In addition, we have included results in real-world network scenarios (GÉANT network with a set of traffic matrices) as well as a comparison with an RL-based routing algorithm.

IV. RDRL Routing Strategies

In this section, we detail our RDRL solution for routing in wireless networks. We start with a high-level overview and then describe the main design decisions.

A. High-level Overview

We use *offline centralized training* and *online distributed execution* for routing decisions in this work. The centralized training is computationally expensive, and hence takes place offline before any testing is done. For scalability, we use *parameter sharing*, i.e., we train a single model with model parameters shared by all agents; a copy of the trained model is then used locally at each node in the network for decision-making.

Specifically, we train a *packet agent*-based DRL model that is able to be used locally at each node by packets at that node (see §IV-B). This is possible because we use *relational features* (see §IV-C). Relational features allow us to aggregate all of the data collected from nodes in the network into a single training set, enabling us to *train the DRL model efficiently* (see §V). Relational features also allow us to train a model for one network scenario, but then use it for another different network scenario, thus achieving *better generalization*. Once the routing decision model has been trained, during testing

[3]The studies in [39], [40] state that the centralized training in our preliminary version [1] is done online at a central node and leads to a large amount of communication overhead. While our training is centralized, it is done offline, so there is no training communication overhead and no risk of a single-point of failure. See more details in §IV.

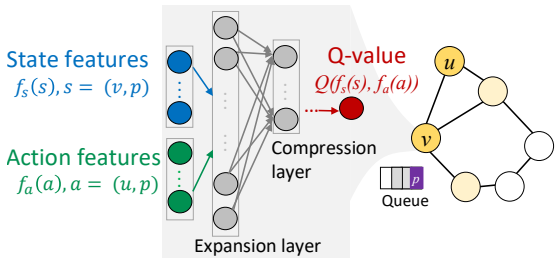


Fig. 1. Illustration of packet-centric decision with the input features, DNN, and output. Packet p is considering the action of moving from node v to node u at time t, so feeds the associated features into the DNN to obtain the Q-value for taking action u.

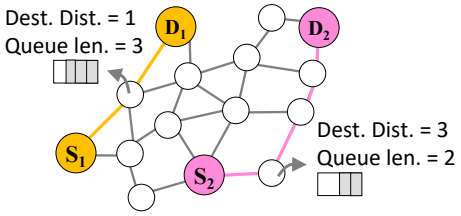


Fig. 2. Illustration of two relational features, destination distance and queue length. Packets can have different destinations, D_1 and D_2 respectively, but still use the same relational features, just with different values. This example illustrates two flows, from S_1 to D_1 and from S_2 to D_2; the features marked at a node are for the first packet in the queue of that node.

the DRL model is copied to each node for fast distributed decision-making. That is, the parameters of the model are fixed during testing, though the feature inputs to the model may vary.

We use a feedforward DNN model with fully connected layers for function approximation in our DRL approach, see Fig. 1. The inputs to our DNN are the relational feature vectors for a state-action pair. The output is a single neuron which estimates the Q-value for that pair. We obtain the input features and output of the DNN as follows. We define two functions, $f_s(\cdot)$ and $f_a(\cdot)$, that translate a state and an action into, respectively, a set of state features and a set of action features. Consider state s and action a, which are mapped to the corresponding features, $f_s(s)$ and $f_a(a)$. These two sets of features are then used as input to the DNN, to produce as output an approximate Q-function, $\widehat{Q}(f_s(s), f_a(a))$.

When training the DNN on a state s, action a, reward r, and next state s' transition, we use $f_s(s), f_a(a)$ as input and the TD target y as output (sampling from $\widehat{Q}$). Specifically, in our setting, y is obtained as

$$y = r + \gamma \cdot \max_{a' \in \mathcal{A}(s')} \widehat{Q}(f_s(s'), f_a(a')). \tag{4}$$

When using the DNN for prediction, i.e., to select the next hop forwarding action, the output of the DNN is the Q-value for taking action a in state s, $Q(f_s(s), f_a(a))$. When links are dynamic, only those neighbors for which there is currently a link are considered as possible actions. Specifically, we first obtain the Q-value, $Q(f_s(s), f_a(a))$, for each action, $a \in \mathcal{A}(s)$. Then, during testing, we select the neighbor a with the highest Q-value as the next hop. During training, we use ϵ-greedy to select the next hop, where with probability ϵ we choose a random neighbor as the next hop and with probability $1 - \epsilon$ we choose the neighbor with the highest Q-value. The above design thus provides us the flexibility to handle an arbitrary and varying number of possible next hop actions and neighbors.

B. Packet Agents vs. Node Agents

An agent's experience consists of (s, a, r) tuples that are chained together into time sequences with the next state, s'. There are two natural ways to do this. In *node-centric decisions*, each node (i.e., router) is an agent and independently makes a decision about where to forward the packet at the front of the queue. In *packet-centric decisions*, each packet that travels through the network is an agent and independently makes a decision when it reaches the front of a node's queue; the packet may choose to stay at the current node or move to a neighbor. In both cases, the state and action features are the same. The difference is in how states and actions are chained together in time sequences: either all experiences from the same packet form a sequence, or all experiences from the same node form a sequence.

While the node-centric approach may seem more intuitive since we typically think of wireless nodes as making routing decisions, here we use the packet-centric approach, as it provides a more natural way to propagate reward, which is defined based on packet states, back to the previous time steps and actions that helped deliver the packet (or not). Fig. 1 shows an example, where packet p at the head of the queue of node v calculates its Q-value to go to neighbor u as part of its decision making.

Packet-centric decision-making can be viewed as a multi-agent problem, as each packet interacts with others while attempting to greedily optimize its own travel time. To reduce computational complexity, however, we do not use a global cooperative reward function. Instead, we have each node queue enforce fairness among its packets: i.e., only the packet at the front of the queue gets to choose to move to another node at each time step. This allows for decoupled decision-making and avoids the need for explicit cooperation among packets or nodes for next hop decisions.

C. State and Action Features

We define relational features to be those that are independent of the network topology and traffic on which the DNN is trained. In our work, all packets in the network use the same set of relational features for routing decisions, though the values of specific features will differ for different packets.

A simple example of relational features is given in Fig. 2, where the distance to a destination characterizes the distance from a packet's current node location to the packet's destination node, while queue length represents the number of packets currently at that node. Both of these features can be defined for an arbitrary node and packet, with only their values depending on the specific network topology and traffic. We do not use any information specific to a node or packet as a feature, such as node ID or packet destination ID, as doing so would hinder generalization.

In our evaluation in §VII we use a finite set of easy-to-calculate state and action features for the per-packet state s and associated action set $\mathcal{A}(s)$, defined by the following functions.

State features. For a packet p at node v at time t, with one-hop neighbor set $Nbr(v)$, the state features are a function, $f_s(v,p,t)$, comprised of three sets of features:

$$f_s(v,p,t) \quad = \quad f_{packet}(\cdot) \cup f_{node}(\cdot) \cup f_{neighbor}(\cdot)$$

where $f_{packet}(\cdot)$, $f_{node}(\cdot)$, and $f_{neighbor}(\cdot)$ represent packet, node, and neighbor features, respectively, as described below.

1) *Packet features,* $f_{packet}(v,p,t)$. These are features derived from the packet p itself. Currently, our only packet feature is packet p's time-to-live (TTL) field. A packet's TTL field prevents the packet from looping forever in the network: it is set to an initial value and is decremented by one after traversing a hop; when the TTL field reaches zero, the packet is dropped. We use this feature since intuitively a packet with a larger TTL value may need to be treated differently from one with a lower TTL value. a lower TTL value.
2) *Node features,* $f_{node}(v,p,t)$. These are features derived from the node v at which the packet is currently located. We use i) the estimated distance (in terms of hop count) from node v to the packet's destination, $dest(p)$, ii) v's queue length, iii) v's queue length considering only packets destined to $dest(p)$, and iv) v's node degree. These features are intuitively helpful for routing decisions. Distance to destination is important state information that is typically estimated by routing algorithms. The use of queue length and per-destination queue length as features is inspired by the backpressure routing algorithm [2], which uses differences in per-destination queue lengths to choose next hops for packets and is throughput optimal under certain conditions. Node degree is useful for characterizing the connectivity of neighboring nodes as possible next hops: e.g., more well-connected next hops are more likely to have a short path to a given destination.
3) *Aggregated neighbor features,* $f_{neighbor}(Nbr(v),p,t)$. These are features aggregated over all neighbors of node v. We first compute the node features, $f_{node}(u,p)$, for each neighbor $u \in Nbr(v)$. Then we compute the minimum, mean, and maximum of these features. This is similar in spirit to the aggregation function in a GNN [41]. These features allow us to compare the feature values of a particular action at a node to an aggregation of the feature values for all possible actions in the node's neighborhood. This comparison then gives insight into how a given action under consideration compares to the possible actions in the node's neighborhood as a whole.

Like our aggregated neighbor features, we note that several studies [42]–[44] leverage the generalization capability of Graph Neural networks (GNNs) for routing so that the learned strategies are generalizable to other topologies and traffic intensities. The study in [42], however, relies on supervised learning. The studies in [43], [44] combine DRL and GNNs for centralized routing; designing such models for a distributed setting is much more challenging.

Feature normalization. Since the above feature values can be in dramatically different ranges, we normalize them to be in similar ranges. Specifically, let f_i be the value of feature i and f_i^{max} be its maximum value. We use the normalized features $f_i = (f_i+1)/(f_i^{max}+1)$ as the input to the DNN, where we add 1 to both the numerator and denominator so that no feature has value 0. In the rest of the paper, all features refer to normalized features.

Action features. Each action at time t selects a next hop for the packet p that is at the front of node v's queue. A packet can choose either to stay at its current node or transition to one of the neighboring nodes. Let $\mathcal{A}(v) = Nbr(v) \cup \{v\}$ be the set of possible actions at node v. Actions are represented via relational features that abstractly represent these choices. For packet p considering moving from v to $u \in \mathcal{A}(v)$, the features for action u are given by $f_a(u,p,t)$, which corresponds to the node features of u:

$$f_a(u,p,t) \quad = \quad f_{node}(u,p,t).$$

D. Reward Function

An RL agent learns to maximize the expected future reward for each state-action pair. We divide states into three categories: (1) *delivery* states, in which a packet is delivered to its destination, (2) *drop* states, in which a packet is dropped, and (3) *transition* states, in which a packet either stays at its current node or is transmitted to a neighbor that is not its destination. Our reward function, r, is then:

$$r_{delivery} = 0, r_{transition} = -1, r_{drop} = r_{transition}/(1-\gamma)$$

where $\gamma \in [0,1]$, as described earlier, is the RL discount factor. In §IV-B, we described a packet-centric view that formulates routing as an episodic task that terminates when a packet is delivered or dropped. The drop reward r_{drop} is thus defined to be equivalent to receiving $r_{transition}$ for infinite time steps. As such, since $\gamma < 1$, dropping a packet carries a large penalty to discourage packet loss. We explain the above specific value of r_{drop} in §IV-E.

E. Actions vs. Options

Actions in Q-learning typically take only one time step to complete. Routing actions, however, often involve multiple time steps: e.g., after a packet arrives at a new node, it waits for some time in the node's queue, with no opportunity to make a routing decision. This scenario is a natural case for extended-time actions, or options [19], which take a variable amount of time. The time interval that the packet waits is treated as a single option. This approach requires less data to be collected, and allows Q-learning to proceed more quickly.

The sample estimate of expected return (i.e., the sum of discounted future rewards) for an option that starts at time step t_i and ends at time step t_j is (see [19]):

$$y = \sum_{k=t_i}^{t_j-1} \gamma^{k-t_i} \cdot r_k + \quad \left[\gamma^{(t_j-t_i)} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a') \right]$$

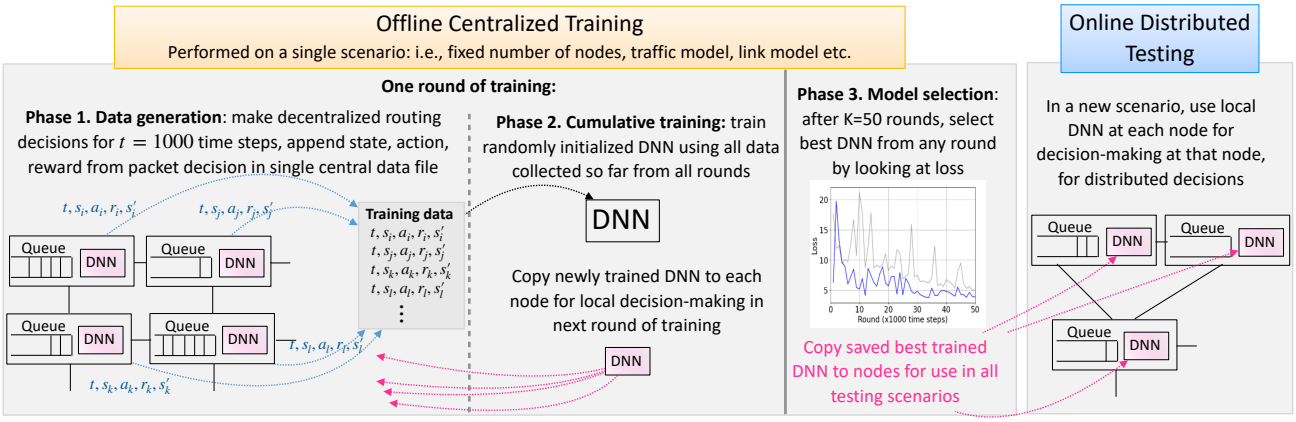


Fig. 3. RDRL: overview of our offline centralized training and online distributed testing methodology. While we show just a single training round here, in practice we execute D training rounds in a single training simulation, all on the same fixed network scenario and traffic pattern. Once training completes, we look at the training vs. validation loss for each of the D rounds to determine the round with the "best" trained model, denoted as M_{θ^*}. Then this model is used in all testing scenarios, including on network scenarios and traffic patterns that were unseen in training.

where r_k is the reward at time step k, and s_{t_j} is the state encountered at time t_j, with $\mathcal{A}(s_{t_j})$ its actions. We use this as the output target y for the neural network, replacing Eq. (4).

Here, we consider only reward functions that are constant for every time step over the life of the option (see §IV-D), so that all $r_k = r_c$, where r_c is one of our three reward types. The sample return for an option starting at time t_i and ending at time t_j with constant per-time step reward r_c is then:

$$y = R(t_j - t_i, r_c) + \gamma^{(t_j - t_i)} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a') \quad (5)$$

where

$$R(t_j - t_i, r_c) = r_c \cdot \frac{1 - \gamma^{(t_j - t_i)}}{1 - \gamma}.$$

There are two types of options in this domain: *terminal* (packet delivery or drop) and *non-terminal* (transitions from one node to another). On packet delivery, the option takes only a single time step and the next state s_{t_j} is the terminal state. The sample return for delivery is:

$$y = R(t_j - t_i, r_{delivery}) + \gamma^{(t_j - t_i)} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a')$$
$$= r_{delivery} \cdot \frac{1 - \gamma^1}{1 - \gamma} + 0 = r_{delivery}.$$

For a packet drop, although it also only takes a single time step, we model it as a transition for infinite time steps (i.e., the return is equivalent to never being delivered). As such, since $\gamma < 1$, we have

$$y = R(\infty, r_{transition}) + \gamma^{\infty} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a')$$
$$= \frac{r_{transition}}{1 - \gamma} + 0 = \frac{r_{transition}}{1 - \gamma}.$$

On non-terminal transitions the sample return is:

$$y = R(t_j - t_i, r_{transition}) + \gamma^{(t_j - t_i)} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a').$$

Every packet that remains in a node queue at the end of a training round (described in the beginning of §V) has an *unfinished* option. We remove such options from the data. As more data accumulates, including the end of the option, the newly finished options are used. Henceforth, to be consistent with §IV-C, we use "actions" rather than "options" to refer to extended-time actions.

F. Action Selection

Different numbers of actions are available to packets at different nodes. Thus, when packet p makes a decision at node v, the feature set $(f_s(\cdot), f_a(\cdot))$ is fed into the DNN as input for each $u \in v \cup Nbr(v)$ to obtain a list of estimated $Q(f_s(p, v, t), f_a(u, p, t))$ values, one Q-value for each possible next hop action u including staying at node v. These Q-values can then be fed into any action selection mechanism; here, we use ϵ-greedy. That is, the agent chooses a random action with probability ϵ; otherwise, the agent chooses the action u with the highest Q-value.

V. RDRL Routing Training vs. Execution

In this section, we describe our offline centralized training methodology for RDRL (§V-A). The testing is performed online, in a fully distributed manner (§V-B).

A. Offline Centralized Training

For a given network scenario and traffic setting, we divide training of the DNN model encoding the RDRL's agent's routing policy into three phases: (1) data generation, (2) cumulative training, and (3) model selection. This is illustrated in Fig. 3. We further divide training of the DNN model into rounds, where each round comprises 1000 time steps and is repeated D times. Phases 1 and 2 are done every training round; only once all D training rounds are complete does Phase 3 occur. At the end of round d, we define M_{θ_d} to be the newly trained RDRL agent using all of the data through round d, for $d = 1, \ldots, D$. This model is then used for decision making in round $d + 1$. In the first round (i.e., $d = 1$), the parameters for the DNN model, θ_0, used for decision-making are randomly initialized since no previous trained DNN model exists yet.

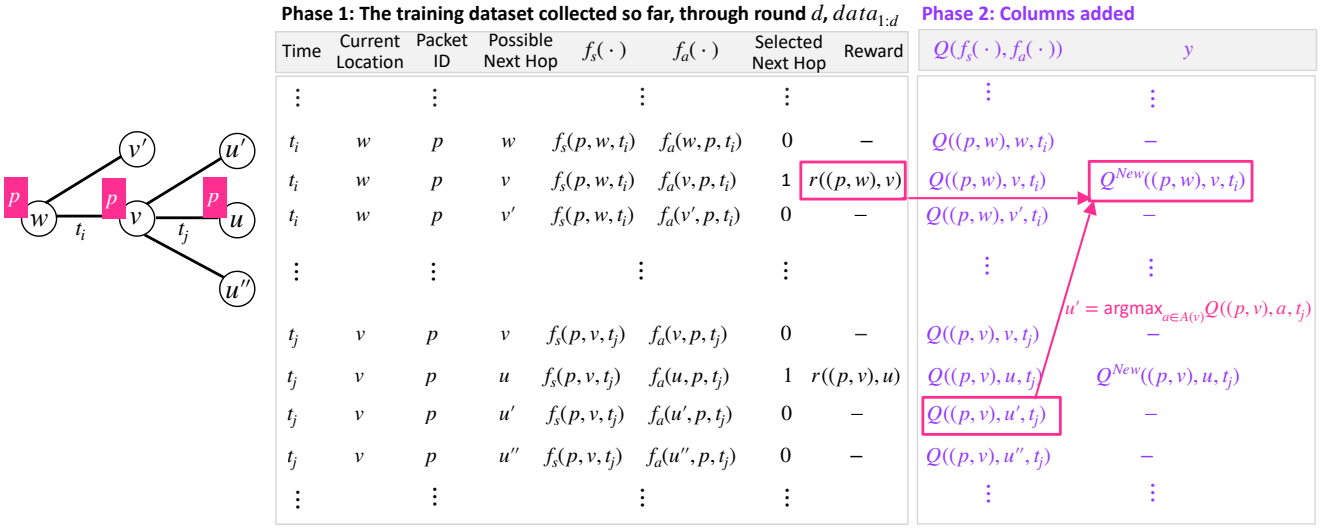


Fig. 4. Illustration of how target Q-values are computed from the training data generated during rounds 1 through d, referred to as $data_{1:d}$, for the training of the DNN model M_{θ_d} at the end of the d-th round. The details of the computation of the Q-values, $Q^{New}((p,w),v,t_i)$, are described in Algorithm 1.

Phase 1: Data Generation. We describe data generation from the viewpoint of a single round, d. Let $data_{1:d}$ be a table representing all of the data generated from packet decisions so far through round d; this table is illustrated in Fig. 4 (see the columns marked in black). Each row of the table corresponds to a packet action that could have or did happen at some node in the network at a given time in some round $i \leq d$. For instance, in Fig. 4, three rows are associated with time t_i, representing a potential forwarding of packet p from node w back to itself, to neighbor node v, or to neighbor node v'. Similarly, four rows are associated with time t_j, representing the four possible forwarding choices for p at its new node location of v.

In round d, to select one of the possible forwarding actions for packet p, the state features $f_s(\cdot)$, and action features, $f_a(\cdot)$ for that row are input into $M_{\theta_{d-1}}$ (i.e., the model that is used for decision making for round d), and the resulting Q-value is obtained. Then the next hop action with the best Q-value is selected. In each row, a binary flag, "selected next hop", indicates this selected next hop choice: this flag is set to 1 for the packet's chosen next hop action, and 0 otherwise. In the example in Fig. 4, we see that at time t_i, packet p at node w chooses to move to neighbor node v; then at a later time $t_j > t_i$, packet p at node v chooses to move to neighbor node u. As shown in Fig. 4, a packet's chosen action is associated with some resulting reward (as defined in §IV-D), and this is also included in the row added to the table.

Phase 2: Cumulative Training. At the end of the d-th training round, after the data generation phase has been completed, $data_{1:d}$ represents the data that has been generated by all nodes through round d. This dataset is centrally pooled to train a new DNN model, M_{θ_d}, for use in decision-making in the next round, $d+1$. The centralized pooling of $data_{1:d}$ from all nodes (instead of using only data from a single node) reduces the time needed to generate training data and thus reduces the overall time needed to train a DNN model. This periodic update of the DNN model also helps to avoid instability caused by "chasing a nonstationary target" [37].

Algorithm 1 Training of DNN model at the end of round d.

Input: data generated from round 1 through round d, $data_{1:d}$

Create a new randomly initialized DNN, M_{θ_d}
for k = *1 to # of RL iterations* **do**
  Compute $Q(f_s(\cdot), f_a(\cdot))$ column in Fig. 4 by feeding $f_s(\cdot)$ and $f_a(\cdot)$ into M_{θ_d} for each row in $data_{1:d}$
  Compute $subset_{1:d}$, by selecting the subset of rows in $data_{1:d}$ for which "selected next hop" = 1
  for *every row in* $subset_{1:d}$*, compute target* y **do**
    Let p be the packet associated with the row
    Let node v be the next hop that p selects when at p's current node w at time t_i
    Let $r((p,w),v)$ be the resulting reward for p selecting v
    Consider all rows in $data_{1:d}$ for p at v at time t_j to obtain the set of potential next state actions $\mathcal{A}(v)$ for p
    Take max over p's next state actions to obtain $Q_v^{\max} \triangleq \max_{a \in \mathcal{A}(v)} Q((p,v),a,t_j)$
    Record $y = R(t_j - t_i, r((p,w),v)) + \gamma^{t_j - t_i} Q_v^{\max}$ as the new target, $Q^{New}((p,w),v,t_i)$ (see Fig. 4), for packet p's decision at node w at time t_i
  end
  Train M_{θ_d} to fit target column y given as input $f_s(\cdot)$ and $f_a(\cdot)$ columns.
end

Algorithm 1 summarizes the steps for training M_{θ_d} on $data_{1:d}$ at the end of training round d. We first randomly initialize a new DNN model, M_{θ_d}. Then we work on constructing the dataset $\mathcal{S}$ from $data_{1:d}$ that will be used to fit M_{θ_d}. This is done as follows. The outer loop of RL iterations in Algorithm 1 performs the RL backups by fitting M_{θ_d} for a given target y and state and action inputs, $f_s(\cdot)$ and $f_a(\cdot)$, while the inner loop computes the updated target y with better Q-value estimates from the last RL iteration fitting M_{θ_d}.

For ease of notation, we introduce the following shorthand notation in Fig. 4. For packet p at node v considering moving to node u at time t, the Q-function is $Q(f_s(p,v,t), f_a(u,p,t))$,

which we shorten to

$$Q((p,v),u,t) \quad \triangleq \quad Q(f_s(p,v,t), f_a(u,p,t)), \tag{6}$$

where (p,v) represents the state (i.e., of packet p at node v), u represents the potential next hop action, and t represents the time. In each RL iteration, as shown in Fig. 4, we first calculate the Q-value for each row of $data_{1:d}$ using the current M_{θ_d}. Then we select out the subset of rows in $data_{1:d}$ that correspond to next hop actions actually taken by packets and store these as $subset_{1:d}$.

For each row in $subset_{1:d}$, we consider the packet p that made a next hop selection in that row. Suppose p is at node w and makes a forwarding decision at a time t_i, selecting v as its next hop. The immediate reward for taking action w is $r((p,w),v)$, corresponding to one of $r_{delivery}$, $r_{transition}$, or r_{drop} as in §IV-D. We then look at all possible actions available to p at its new node v, $a \in \mathcal{A}(v)$ at time t_j: each of these possible actions a is represented as one row in $data_{1:d}$ and has a corresponding $Q((p,v),a,t_j)$ already computed at the start of the Algorithm 1. [4] We can thus obtain $Q_v^{\max} \triangleq \max_{a\in\mathcal{A}(v)} Q((p,v),a,t_j)$. After that, we calculate the target value for p's row in $subset_{1:d}$ following Eq. (5) as

$$y = R(t_j - t_i, r((p,w),v)) + \gamma^{t_j - t_i} Q_v^{\max}.$$

Following this process, we obtain the training dataset used to fit the DNN model, M_{θ_d}. For each row in $subset_{1:d}$ we have the sample $(f_s(p,w,t_i), f_a(v,p,t_i), y)$, where $f_s(p,w,t_i)$ and $f_a(v,p,t_i)$ are the state and action inputs to M_{θ_d}, and y is the associated target. These rows then form our training dataset $\mathcal{S}$ and $|\mathcal{S}|$ is the same as the number of rows in $subset_{1:d}$. We then use $\mathcal{S}$ to fit M_{θ_d} for this RL iteration of round d.

Importantly, the parameters of each newly trained DNN model at the end of each round d, θ_d, are shared among nodes. For instance, during training round $d+1$ each node has a local copy of the same DNN model, M_{θ_d}, for packets to use to make next-hop forwarding decisions when at that node.

Sampling. To reduce the running time of training during each round, we sample a subset of $\mathcal{S}$, and only fit M_{θ_d} to this subset (not shown in Algorithm 1). To ensure that representative samples are selected, we select the samples based on flow ID. Specifically, we associate each flow with a globally unique ID: this is possible since we do this only during the training process, when we have global knowledge of the network. As an example, we associate monotonically increasing IDs to flows based on generation time. For the samples in $\mathcal{S}$, let $\mathcal{F}$ be the set of unique flows, and n_f be the number of samples in flow $f \in \mathcal{F}$. Then for a sampling rate of β, we take βn_f samples from flow f. We round the number of samples to the closest integer, except that if $\beta n_f < 1$, we set it to one despite the actual value (so as to take at least one sample from that flow). We further reduce the computational overhead by sampling before the Q-value calculation step, thereby avoiding the computation of targets for data that will not be used in training.

[4]Note that the action that leads to $Q_v^{\max}$ may not be the action that was actually taken; one example of this is shown in Fig. 4, where at time t_j, the routing decision is to forward to u, while the action that leads to $Q_v^{\max}$ is to forward to u'. This is because Q-learning is an off-policy RL algorithm: the policy being learned is independent of the policy that was used for decision-making.

TABLE I
DNN SETTINGS USED IN OUR SIMULATIONS.

Setting	Value
# of input features	$F = \lvert f_s(\cdot)\rvert + \lvert f_a(\cdot)\rvert$
Size of expansion layer	$10F$
Size of compression layer	$F/2$
Size of output layer	1
Learning rate	0.0001
Activation function (all layers)	ReLU
Optimizer (all layers)	Adam
Loss	Mean squared error
Batch size	32
# of Epochs	10
Training validation split	0.2

Phase 3: Model Selection. Once all D rounds of training are complete, we compare the training vs. validation loss for each of the DNN models M_{θ_d}, $d = 1, \ldots, D$, fit during each of the D rounds of training. From all of these rounds, we select the model, M_{θ^*}, with the minimum joint loss for both training and validation to avoid overfitting: this is what we call the "best" model and it is this model that we will later use during testing.

B. Online Distributed Testing

Testing is illustrated in the right part of Fig. 3. Once training is complete, as in Fig. 3, we copy M_{θ^*} to each node in the network for *online distributed testing*. During testing, the performance of the learned routing strategy, as encoded by M_{θ^*}, is evaluated on a variety of network scenarios, primarily scenarios that were unseen during training.

VI. Training and Evaluation Settings

We develop a discrete-time packet-level network simulator in Python and use it train our RDRL agents, and compare their performance with other algorithms. Table I lists the settings of the DNN for our RDRL routing strategy. The feed-forward, fully connected DNN architecture contains 4 layers: input, expansion, compression, and output. Let $F = |f_s(\cdot)| + |f_a(\cdot)|$ be the number of input features and thus the size of the input layer. The expansion layer has $10F$ neurons and the compression layer has $F/2$ neurons. The implementation uses Keras v.2.3.1 [45] and Tensorflow v.1.14.0 [46].

We aim to test how well an RDRL agent trained on one scenario is able to generalize its learned routing policy to unseen scenarios. Therefore, we explore a wide range of scenarios that differ in topology, link dynamics, and traffic.

A. Topology and Link Dynamics

We consider both synthetic and real-world topologies. The synthetic topologies are square grids (lattices) with N nodes, where N ranges from 16 to 81. The real-world topology is the GÉANT network with 23 nodes and 37 links [20]. We simulate the GÉANT network as a multi-hop wireless backbone network, since we are not aware of any large-scale multi-hop wireless network topology in the public domain.

For each topology, we consider *static* scenarios where links are up all the time, and *dynamic* scenarios where links may be up or down, modeled by a 2-state Markov model. Specifically, a link can be in either the up or down state in a time step. If it is in the up state, it stays up in the next time step with probability p_{up} (i.e., transitions to the down state with probability $1-p_{up}$). If it is in the down state, it stays down in the next time step with probability p_{down}. For a given topology, we initialize the up and down states of links based on the steady-state link probability for this 2-state model, with the link up probability being $\pi = (1-p_{down})/(2-p_{up}-p_{down})$. For grid topologies, the 2-state Markov model uses $p_{up} = 0.5$ and $p_{down} = 0.4$, so that the network has poor connectivity. For GÉANT topology, since we regard it as a wireless backbone network, we use $p_{up} = 0.9$ and $p_{down} = 0.2$ for better connectivity.

In §VII, we only use two scenarios (grid topology, $N = 64$, with either static or dynamic links) to train two RDRL models. We then test them in all remaining scenarios (including grid topologies of different sizes and the GÉANT topology).

B. Traffic Generation and Packet Forwarding

Grid topologies. For grid topologies, we generate traffic flows (where each flow is between a source and destination pair selected uniformly randomly) according to a Poisson distribution with parameter λ_F; flow durations are generated by sampling an exponential distribution with parameter λ_D. Packet arrivals in a flow are generated according to a Poisson distribution with λ_P as the average number of new packets generated per time step. A simulation starts with $\lambda_F\lambda_D$ initial flows. We set $\lambda_F = 0.002N/25$, $\lambda_D = 5000$, and $\lambda_P = 0.2$. That is, on average $2N/25$ new packets are generated in the network per time step, leading to more traffic in larger networks.

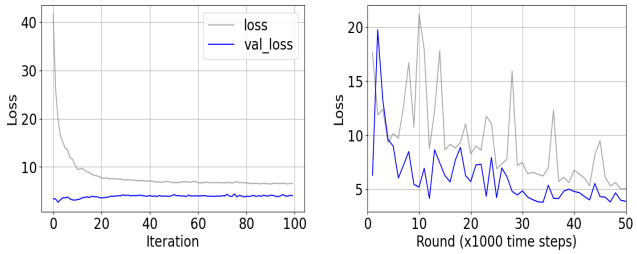
In each time step, we loop through all nodes in random order and allow each node to transmit a single packet that was previously received or generated. For simplicity, packet forwarding at each node follows a First Come First Serve order. That is, the first packet in the buffer will be routed first. If the routing decision is that the packet stays in the buffer, then this packet will be placed at the end of the queue, and routing will be applied to the next packet. This process repeats until one packet is routed to a neighbor, or all packets in the buffer end up staying at the current node. Each node maintains a packet queue with a maximum buffer size, B, beyond which additional packets are dropped.

GÉANT topology. In the GÉANT topology, traffic generation follows the measured traffic matrices [21]. Packet forwarding follows the GÉANT link capacity (see §VII-C). Again each node has a maximum buffer size B.

C. Comparison Routing Algorithms

We compare our RDRL models with three other routing algorithms: one based on RL along with two classical routing algorithms.

• **Q-protocol.** This protocol is inspired by [23], which develops a protocol based on Q-routing related approaches and uses a Distance Vector-like protocol to exchange messages among neighbors periodically (every 10 time units). In Q-protocol, we reduce the update interval to every time step so that nodes have even more up-to-date information for their decision-making. This is in contrast to Q-routing, where a node x only obtains an update from a neighbor y after x sends a packet to y, which motivated subsequent studies to address the issue of out-of-date information (see summary of these studies in [23]). We verify that Q-protocol substantially outperforms the original Q-routing algorithm on all of our performance metrics.



(a) (b)

Fig. 5. Loss and validation loss for one training scenario (64-node static grid topology): (a) loss versus iteration in one training round; (b) loss versus training round (each training round is 1000 time steps).

• **Shortest path routing (SP).** We implement shortest path routing as a distance vector algorithm using hop count as cost. Because devices are stationary though links may transition up or down, we assume that once a device determines that a link is possible to a neighbor device, that link continues to be present in the distance calculations. However, only those neighbors for which there are actual links present are considered as possible next hops when a routing decision is made.

• **Backpressure routing (BP).** Consider an arbitrary device v. Let b_d^v be the number of packets destined to device d in the queue at device v. For every destination d of a packet in v's queue, v computes $b_d^v - b_d^u$ for the neighbors $u \in Nbr(v)$ currently available. Then v finds the optimal destination d^* and corresponding neighbor u^*, such that $b_{d^*}^v - b_{d^*}^{u^*}$ is the largest among all destinations (breaking ties arbitrarily), i.e., BP routes packets in the direction that maximizes the *differential backlog* between neighboring devices. If $b_{d^*}^v - b_{d^*}^{u^*} > 0$, then v sends a packet with destination d^* to u^*; otherwise v does not send any packet. Unlike other algorithms, which forward the packet at the front of a device's queue, BP chooses the best packet from anywhere in the queue to forward.

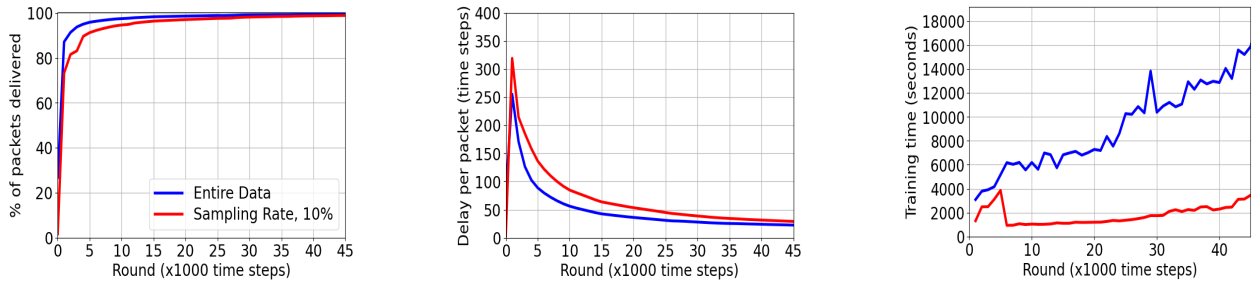
Performance metrics. Corresponding to the three goals of routing in multi-hop wireless networks (see §II-A), we use three performance metrics: delivery rate, and average delay and number of forwards for delivering packets from source to destination.

VII. Training and Evaluation Results

A. RDRL Training Results

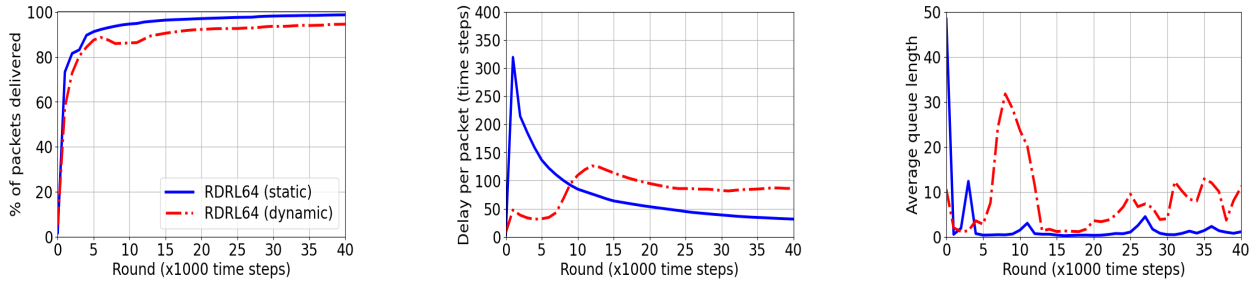
We train two RDRL models following the methodology in §V. Both models are trained in grid topology with $N = 64$[5].

[5]We also trained two models in a small network size ($N = 25$) and found that the models do not generalize as well as those trained when $N = 64$.



(a) Delivery rate. (b) Delay. (c) Computation time.

Fig. 6. Comparison of training performance for a 10% sampling rate vs. no sampling (i.e., using the entire dataset) for a 64-node static lattice topology with high traffic load.



(a) % of packets delivered. (b) Delay per delivered packet. (c) Average queue length.

Fig. 7. Training performance of the two RDRL agents that are trained for the two synthetic network scenarios. For both scenarios, the network size is $N = 64$.

They are trained in static and dynamic settings, respectively, referred to henceforth as *RDRL64 (static)* and *RDRL64 (dynamic)*.

During the training, we use the ϵ-greedy algorithm for exploration with $\epsilon_{train} = 0.1$. The RL discount rate, γ, is set to 0.99. The number of training time steps T_{train} is set to $50,000$ for both the static and dynamic settings. As mentioned earlier, training is divided into training rounds, with each round comprising $T_{round} = 1000$ time steps. To evaluate the packet delivery rate, we add a cool-down period at the end of the simulation where no more packets are generated, allowing queues to drain when needed.

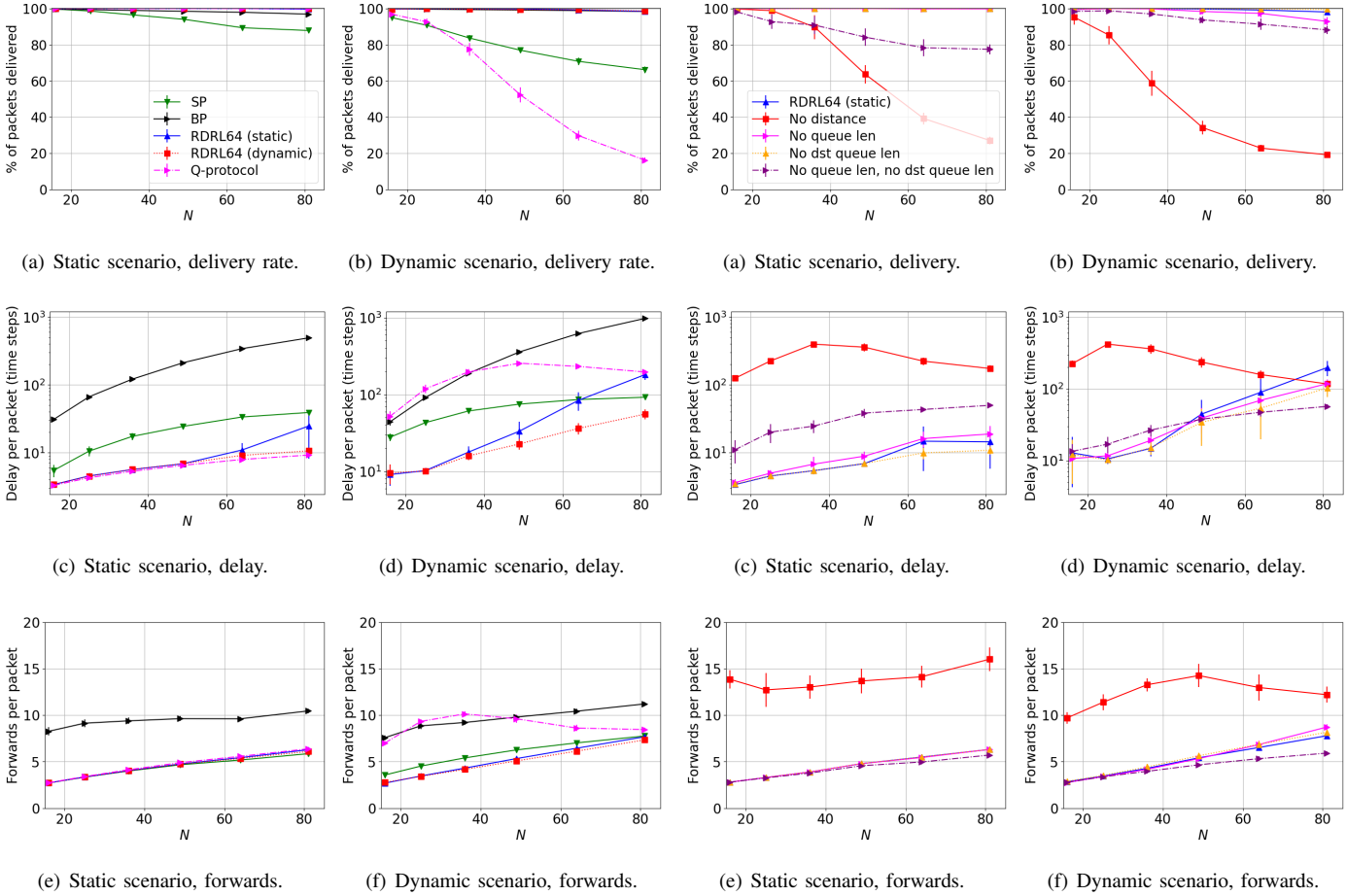
A RDRL agent estimates the distance feature using the distributed distance vector algorithm. Because packets may take very long paths while the RDRL agent is learning a policy during training, we set the initial TTL to $L = 200$, much larger than the expected path length, to prevent packets from always being dropped before the RDRL agent has had sufficient time to learn. Other features and feature normalization are described in §IV-C. Specifically, for node features, when normalizing the distance to destination, queue length, and node degree, we set the maximum destination distance to N, the maximum queue length to $B = 50$, and the maximum degree to N. For the packet TTL, we set the maximum TTL to $L = 200$.

Training and validation losses. Fig. 5(a) shows an example of training and validation losses over the number of iterations in one training round in one setting (static grid topology, 64 nodes). It shows that the model converges within 100 iterations. Fig. 5(b) shows loss and validation loss versus training rounds for the same setting. It shows that loss in general reduces over the training rounds. We choose a model in a round that has both low loss and validation loss, which is round 32 for this setting.

Sampling rate. We experimented with sampling rates ranging from 5% to 30% in various settings and found that a sampling rate of 10% achieves the best trade-off in reducing computational overhead while still maintaining a performance similar to that when using all of the data. Fig. 6 shows the results during training for the static case. Fig. 6(a) and (b) show two performance metrics: delivery rate and delay, while Fig. 6(c) plots the computational overhead. We see that the 10% sampling rate leads to similar delivery rate and delay as the model with no sampling, while reducing the computation time by up to 7.9× (the average reduction is 5.5×). We therefore use 10% sampling rate while training models.

Training convergence. Fig. 7 plots the training results for both the static and dynamic models. In Fig. 7(a), we see that the agent trained in the static scenario takes a shorter amount of time to converge than the agent trained in the dynamic scenario. This is not surprising since the static scenario is less complex than the dynamic scenario.

Fig. 7(b) shows delay per delivered packet. We observe both models converge to a steady delay over time. Fig. 7(c) plots average queue length over time. We again observe convergence behavior where the average queue length converges to a low value for both models.



(a) Static scenario, delivery rate. (b) Dynamic scenario, delivery rate.

(c) Static scenario, delay. (d) Dynamic scenario, delay.

(e) Static scenario, forwards. (f) Dynamic scenario, forwards.

Fig. 8. Testing generalization of the two RDRL agents on the grid topologies; connectivity decreases as N increases.

(a) Static scenario, delivery. (b) Dynamic scenario, delivery.

(c) Static scenario, delay. (d) Dynamic scenario, delay.

(e) Static scenario, forwards. (f) Dynamic scenario, forwards.

Fig. 9. Ablation study comparing the performance of the full-fledged RDRL agent (trained on the static scenario, 64 nodes) with RDRL agents trained with some features removed.

B. Testing Results for Grid Topologies

We now apply the two trained RDRL models to the grid topologies in a total of $6 \times 2 = 12$ scenarios: six network sizes ($N = 16$, 25, 36, 49, 64, or 81), and two types of link dynamics (static or dynamic). *Our focus is on generalizability, that is, can these two RDRL models perform well in testing scenarios that differ from their training scenarios?* During testing, we run each simulation for $T_{test} = 50,000$ time steps. The maximum buffer of each node, B, is set to 50 for all the schemes, except for BP, which uses $B = 50N$ since it uses per-destination queues. Again, the initial TTL is set to 200.

Fig. 8 plots testing results on grid topologies varying in size. The three rows of Fig. 8 show the packet delivery rate, delay per delivered packet, and number of forwards per delivered packet. For each setting, the average results from 30 simulation runs along with the 95% confidence interval are plotted.

The first column of Fig. 8 shows testing results for the static scenario. We see that the two RDRL agents deliver all the packets. Q-protocol leads to similar performance as our models. SP delivers significantly fewer packets, and leads to a higher delay than the two RDRL agents since SP chooses path only based on the number of hops, regardless of the congestion levels. BP is able to build an effective congestion gradient and delivers more packets than SP, but with much higher delay than the two RDRL agents. In addition, BP does not achieve 100% delivery rate for large networks ($N = 81$).

The second column of Fig. 8 shows the testing results for the dynamic scenario, where the network is predominantly disconnected. We see that the two RDRL agents significantly outperform Q-protocol in all three performance metrics. This is because in dynamic settings, the topology and traffic lack sufficient periods of stability for Q-protocol to converge to useful routing decisions. Our RDRL agents also outperform SP in all three metrics (delay for SP is only lower than that of RDRL64 when SP does not deliver all packets when $N = 81$). Finally, our RDRL agents also outperform BP, achieving a significantly lower delay with a fewer number of forwards.

In summary, our two RDRL agents generalize well to test scenarios with different network sizes and link dynamics. In static scenarios, the RDRL agent trained in the dynamic setting performs similarly to the agent trained in the static setting, while the latter performs similarly to the former in the dynamic scenarios except with higher delay for large networks.

Ablation study. We now conduct an ablation study to explore the impact of the various features. Specifically, we explore four variants: (1) no distance feature, (2) no queue length feature, (3) no destination queue length feature, and (4) no destination queue length and no queue length features. We train these four variants in the static 64-node setting. Fig. 9 compares the

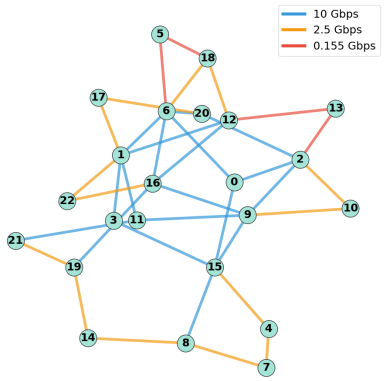


Fig. 10. Illustration of the GÉANT topology. Three types of link capacity are illustrated in color.

performance of these four variants with the full model, i.e., RDRL64 (static). The two columns of Fig. 9 show the testing results in the static and dynamic settings, respectively. In both columns, the number of nodes varies from 16 to 81. We see that for all testing scenarios, the variant that is trained without the distance feature performs poorly on all three metrics, indicating the importance of the distance feature. The variant without the queue length and destination queue length features has low delivery rate and high delay. The variant without queue length (but with distance and destination queue length features) leads to low delivery rate in some dynamic settings. Finally, the variant without destination queue length (but with distance and queue length features) performs similarly to the full model, with lower delay in some cases.

The above results indicate that the distance feature is important for routing decisions. The queue length feature is an important complementary feature to distance. With both distance and queue length features, adding destination queue length feature does not lead to benefits in the above scenarios that we explore. We keep it since it is used in BP and may be beneficial for highly congested scenarios.

C. Testing Results for GÉANT Network

We now evaluate the performance of the two RDRL models in the GÉANT topology. We use the link capacity as specified in [20], where 19, 14 and 4 links have capacity 10, 2.5, 0.155 Gbps, respectively. Fig. 10 shows the topology, where the link capacities are marked in color. As mentioned in §VI, we consider both static scenarios where the links are up all of the time, and dynamic scenarios with $p_{up} = 0.9$ and $p_{down} = 0.2$.

We generate traffic following the publicly available intra-domain traffic matrices [21]. They are for more than three months, from January to April 2005. Each matrix specifies the average amount of traffic (in Kbps) between a source and destination pair over 15 minutes. Since the minimum link capacity is 155 Mbps, we generate packets by treating 155 Mb as a unit. In other words, if the amount of traffic generated in one second is r Mbps, we treat it as $r/155$ packets (rounded to the closest integer). This is equivalent to scaling all link capacities by a factor of 155 to reduce running time. For a given rate, we generate packets following a Poisson process. For all schemes, we set the buffer size to $50N = 1150$ packets at each node. For the links with capacity 155 Mbps, only a single packet can be transmitted in one time step along that link. For a link with capacity 2.5 (vs. 10) Gbps, up to 16 (vs. 64) packets can be transmitted in one time step.

We consider all of the traffic matrices. From these, we identify 10 days that have the highest amount of traffic (in terms of average amount of traffic across all source and destination pairs). We found that even for these days, the network is lightly loaded, and all of the algorithms perform well in static scenarios (figures omitted). In dynamic scenarios, our two RDRL models perform better than other algorithms. Fig. 11 shows the results for one day (March 17, 2005). For clarity, only the results for the first 12 hours are shown in the figure. While all algorithms lead to high delivery rate, only our RDRL models and SP lead to 100% delivery rate. Our models outperform SP in terms of lower delay and number of forwards.

To simulate congested network scenarios, we scale up the amount of traffic of each source and destination pair by $3\times$. Fig. 12 shows the results for the first 12 hours of March 17, 2005 after increasing the load as above. We see that BP delivers all packets since it is optimal in terms of throughput for congested networks. Our RDRL64 (dynamic) model also delivers all packets and has lower delay and number of forwards than BP. Our RDRL64 (static) model delivers all packets in almost all cases; for those that it does not deliver all packets, it still has higher delivery rate than Q-protocol and SP, with lower or comparable delay and number of forwards.

The above results demonstrate that our two RDRL algorithms, even though trained in 64-node grid topologies, generalize well to the GÉANT network which differs significantly from the 64-node grid training scenarios in terms of topology, link capacity, and traffic.

VIII. Conclusions and Future Work

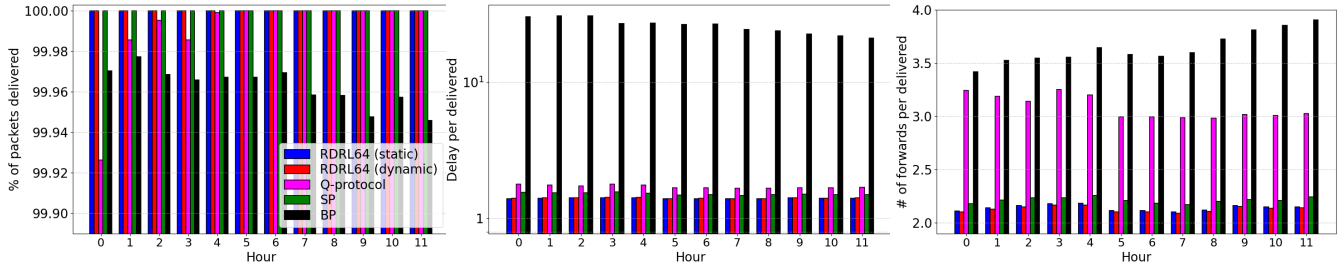
In this work, we have designed a novel distributed packet routing algorithm using relational deep reinforcement learning. Our algorithm generalizes to diverse network scenarios through the use of relational features, packet-centric decision-making, and extended-time actions, and outperforms shortest path routing, backpressure routing, and an RL-based routing approach with respect to packets delivered and delay per packet. There are a number of directions for future work, including extending our design to consider mobile devices, increasing flexibility in choice of packet to send, and further exploring generalization.

Acknowledgements

Results presented in this paper were obtained in part using CloudBank, which is partially supported by the NSF under award #2154190.

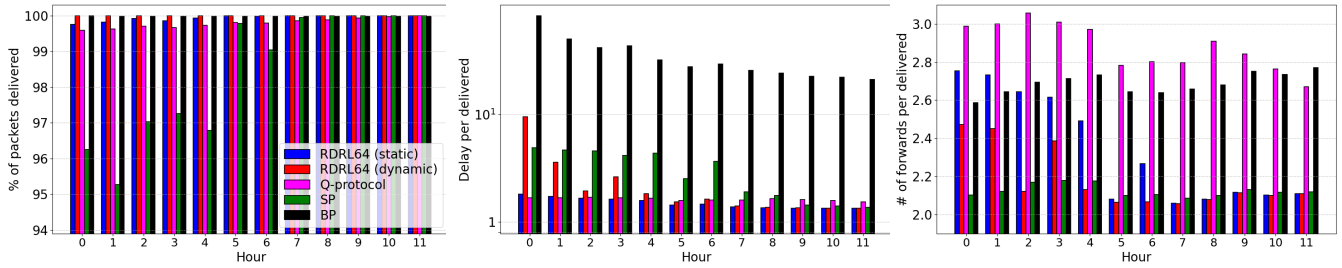
References

[1] V. Manfredi, A. P. Wolfe, B. Wang, and X. Zhang, "Relational deep reinforcement learning for routing in wireless networks," in *Proceedings of IEEE WoWMoM*, June 2021.



(a) Delivery rate. (b) Delay. (c) # of forwards.

Fig. 11. Results for GÉANT network, dynamic scenario ($p_{up} = 0.9$, $p_{down} = 0.2$). Each bar represents the request for one hour using the traffic matrices for that hour. The results are for the first 12 hours on March 17, 2005.



(a) Delivery rate. (b) Delay. (c) # of forwards.

Fig. 12. Results for GÉANT network, static scenario. The results are obtained using the traffic matrices for the first 12 hours on March 17, 2005, with the rate scaled up by $3\times$ to simulate higher network load.

[2] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," in *IEEE Conference on Decision and Control*, 1992.

[3] V. Manfredi, M. Crovella, and J. Kurose, "Understanding stateful vs stateless communication strategies for ad hoc networks," in *MobiCom*, 2011, pp. 313–324.

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

[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*, 1994, pp. 671–678.

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

[7] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, 2019.

[8] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019, fourth Quarter.

[9] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2595–2621, 2018, fourth Quarter.

[10] T. Li, K. Zhu, N. C. Luong, D. Niyato, Q. Wu, Y. Zhang, and B. Chen, "Applications of multi-agent reinforcement learning in future internet: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 2, pp. 1246–1278, 2022, second Quarter.

[11] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of zero-shot generalisation in deep reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 76, pp. 201–264, 2023.

[12] 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 Generation Computer Systems*, vol. 94, pp. 587–600, May 2019.

[13] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward packet routing with fully distributed multiagent deep reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 2, pp. 855–868, Feb. 2022.

[14] L. Chen, B. Hu, Z. Guan, L. Zhao, and X. Shen, "Multiagent meta-reinforcement learning for adaptive multipath routing optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5374–5387, Oct. 2022.

[15] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 6379–6390.

[16] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2018.

[17] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of PPO in cooperative multi-agent games," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, 2022.

[18] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Applied Intelligence*, vol. 53, no. 11, pp. 13 677–13 722, Jun. 2023.

[19] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[20] P. T. Kirstein, "European international academic networking: A 20 year perspective," in *One step ahead, The 20th Trans European Research and Education Networking Conference, June 7-10, 2004, Rhodes, Greece, Selected Papers*. TERENA, 2004.

[21] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.

[22] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[23] A. Bitaillou, B. Parrein, and G. Andrieux, "Q-routing: from the algorithm to the routing protocol," in *Second IFIP International Conference on Machine Learning for Networking*, Dec. 2019.

[24] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[25] Y. Bengio, *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[26] D. Johnson and D. Maltz, "Dynamic source routing in ad hoc wireless network," in *Mobile Computing*, T. Imielinski and H. Korth, Eds. Kluwer Academic Publishers, 1996, ch. 5, pp. 153–181.

[27] C. Perkins and E. Royer, "Ad hoc on-demand distance vector routing," in *IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

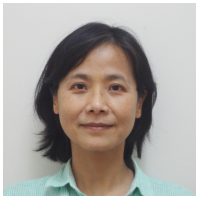
[28] T. Clausen and P. Jacquet, "Optimized link state routing protocol (olsr)," *RFC 3626*, 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3626

[29] K. Sha, J. Gehlot, and R. Greve, "Multipath routing techniques in wireless sensor networks: A survey," *Wireless personal communications*, vol. 70, no. 2, pp. 807–829, 2013.
[30] O. Bello and S. Zeadally, "Intelligent device-to-device communication in the Internet of things," *IEEE Systems Journal*, vol. 10, no. 3, pp. 1172–1182, 2014.
[31] G. Stampa, M. Arias, D. Sanchez-Charles, V. Munts-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," in *CoNEXT Student Workshop*, 2017, arXiv preprint arXiv:1709.07080.
[32] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route with deep RL," in *NIPS Deep Reinforcement Learning Symposium*, 2017.
[33] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: A comprehensive overview," *IET Networks*, vol. 8, no. 2, pp. 79–99, Mar. 2019.
[34] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Access*, vol. 9, pp. 11 465–11 479, Jan. 2021.
[35] K. Chiu, C. Liu, and L. Chou, "Reinforcement learning-based service-oriented dynamic multipath routing in SDN," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–15, Jun. 2022. [Online]. Available: https://doi.org/10.1155/2022/9805372
[36] S. Kaviani, B. Ryu, E. Ahmed, K. A. Larson, A. Le, A. Yahja, and J. H. Kim, "Robust and Scalable Routing with Multi-Agent Deep Reinforcement Learning for MANETs," *arXiv:2101.03273*, 2021.
[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[38] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, Feb. 2022.
[39] R. A. Alliche, T. da Silva Barros, R. Aparicio-Pardo, and L. Sassatelli, "Impact evaluation of control signalling onto distributed learning-based packet routing," in *Proc. of International Teletraffic Congress (ITC)*, Sep. 2022.
[40] R. A. Alliche, R. A. Pardo, and L. Sassatelli, "O-DQR: A multi-agent deep reinforcement learning for multihop routing in overlay networks," *IEEE Transactions on Network and Service Management*, vol. 22, no. 1, pp. 439–455, Feb. 2025.
[41] Peter Battaglia et al., "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf
[42] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Big-DAMA*, 2018.
[43] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in SDN," in *Proc. of SOSR*, 2019.
[44] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case," 2020, https://arxiv.org/abs/1910.07421.
[45] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras
[46] Martín Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

Victoria Manfredi is currently an Associate Professor of Computer Science in the Department of Mathematics and Computer Science at Wesleyan University. She was previously a network scientist at Raytheon BBN Technologies. She received her Ph.D. in Computer Science from the University of Massachusetts, Amherst in 2009. Her research interests are in wireless and mobile ad hoc networks and the application of reinforcement learning techniques to decision problems in computer networks

Alicia P Wolfe received her Ph.D. in Computer Science from the University of Massachusetts, Amherst in 2010.

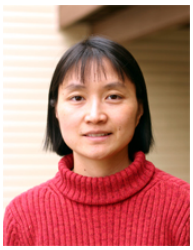
Cheonjin Park received the B.S. degree in Computer Science and Engineering from the University of Connecticut, Storrs, CT, USA, in 2018. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering at the University of Connecticut.



Xiaolan Zhang is currently an associate professor of the Computer and Information Sciences Department at Fordham University. She received her Ph.D. in Computer Science from the University of Massachusetts, Amherst in 2007.

Sushirdeep Narayana is currently a Postdoctoral Fellow in Computer Science at Wesleyan University. He received his Ph.D. in Computer Science from the University of Illinois at Chicago in 2024. Before his Ph.D., he received an M.S. degree in Electrical Engineering from Texas A&M University, College Station, TX. His research interests are in artificial intelligence, computer networks, algorithmic game theory, and reinforcement learning.

Dongjin Song Dongjin Song is currently an associate professor in the School of Computing at the University of Connecticut. He was previously a research staff member at NEC Labs America in Princeton, NJ. He received his Ph.D. degree in the ECE Department from the University of California San Diego in 2016. His research interests include machine learning, data science, deep learning, and related applications for time series data analysis.



Bing Wang is currently a Professor of the School of Computing at the University of Connecticut. She received her Ph.D. in Computer Science from the University of Massachusetts, Amherst in 2005. Her research interests are in Computer Networks, Multimedia, and Distributed Systems. She received the NSF CAREER award in 2008.