Relational Deep Reinforcement Learning for Routing in Wireless Networks

Victoria Manfredi
Wesleyan University
Middletown, CT, USA
vumanfredi@wesleyan.edu

Alicia P Wolfe
Wesleyan University
Middletown, CT, USA
pwolfe@wesleyan.edu

Bing Wang
University of Connecticut
Storrs, CT, USA
bing@uconn.edu

Xiaolan Zhang
Fordham University
Bronx, NY, USA
xzhang@fordham.edu

***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. For instance, protocols that assume network connectivity cannot be easily applied to disconnected networks. In this paper, we develop a distributed routing strategy based on deep reinforcement learning that generalizes to diverse traffic patterns, congestion levels, network connectivity, 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 diverse 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 (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. We evaluate our routing algorithm using a packet-level simulator and show that the policy our algorithm learns during training is able to generalize to larger and more congested networks, different topologies, and diverse link dynamics. Our algorithm outperforms shortest path and backpressure routing with respect to packets delivered and delay per packet.**

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

I. Introduction

The problem of routing in wireless networks is much more challenging than that in wired networks: the shared nature of the wireless medium reduces per-device bandwidth, while the variability of wireless signal propagation and device mobility introduce topology uncertainty. While many routing algorithms have been developed for wireless networks (see §III), they typically assume operation under very specific network conditions. For example, routing algorithms developed for ad hoc networks assume an always connected network, with a focus on finding an optimal connected path from a source to a destination; conversely, routing strategies for delay or disruption tolerant networks assume a mostly disconnected network, and hence the focus is to determine, after encountering another device, whether to forward a packet to that device, so as to optimize some performance criteria.

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 connectivity, or link dynamics. Such a strategy is desirable since the optimal routing algorithm may be very different depending on the network conditions. For example, consider a wireless network that is initially connected and then becomes disconnected due to device failures or link dynamics. In this scenario, a routing algorithm whose goal is to find a connected end-to-end path (e.g., through route discovery requests as in DSR [1] or AODV [2]) will initially succeed, but will eventually fail to deliver any packets when the network becomes disconnected. Alternatively, consider a wireless network that fluctuates between periods of low and high congestion. In this scenario, the use of backpressure routing [3] can achieve 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 identify a set of target network conditions to handle, identify the appropriate routing strategy for each, and then switch strategies as needed. This approach risks instability if network conditions change frequently or algorithms take a long time to converge. Reinforcement learning (RL) [4] allows for an alternate approach in which an RL agent trained on a set of target network conditions learns to make routing decisions even 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 [6] and the references within. 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 §III). However, existing DRL-based routing strategies do not easily generalize to other scenarios because they encode assumptions about the network topology and possible actions into the DNN used to make routing decisions.

In this work, we develop a novel DRL-based routing strategy that is able to generalize to different network conditions. We focus on distributed routing, allowing individual agents to make routing decisions: this supports scalability and provides redundancy in the case of network dynamics. We make the following contributions to DRL-based routing design:

1) *Relational features.* To enable our algorithm to scale to larger networks and generalize to other network conditions, we input relational features to the DNN used to approximate the routing decision space. This not only

978-1-6654-2263-5/21/$31.00

allows data from all DRL agents to be used to train the DNN, but also allows each agent to independently use the same DNN for decision-making during testing.

2) *Packet-centric decisions.* We transform the routing problem into an episodic task 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.
3) *Extended-time actions.* We use extended-time actions or options [7] 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.

We evaluate our approach using a packet-level network simulator. Extensive results demonstrate that our approach is both generalizable and scalable, and significantly outperforms shortest path routing [8] and backpressure routing [3] with respect to packets delivered and delay. The rest of this paper is organized as follows. §II gives background on routing, §III describes related work, §IV overviews our DRL algorithm, §V provides simulation results, and §VI gives our conclusions.

II. Routing in Multi-hop Wireless Networks

Consider a wireless network with a set of nodes, i.e., wireless devices, V. Let $N = |V|$ denote the number of devices in the network. Each device transmits via a wireless channel. Two devices that are within transmission range can communicate with each other. Let E_t denote the set of wireless links present at time t. Due to interference and possible link dynamics, E_t can vary over time. We assume that devices are stationary; device mobility leads to other interesting challenges, which we leave to future work. All devices are capable of receiving and forwarding packets as well as serving as a source or destination. Each device $v \in V$ has a finite queue which can buffer a maximum of B packets. The routing decision at a device v is to choose the neighbor to which to send a packet from v's queue (typically the packet at the front of the queue). A packet carries a time-to-live (TTL) field, which is decremented by one at each intermediate device that forwards the packet. A packet is forwarded until it either reaches its destination, its TTL becomes zero, or it is dropped upon arrival at a device whose queue is full.

Two goals of routing are to (i) maximize throughput, i.e., the packet delivery rate, while (ii) minimizing delay, i.e., the time from a packet being generated at the source to being delivered to its destination. When a network's traffic load changes, or topology changes in the network itself cause traffic load changes, a static routing strategy that ignores congestion can lead to poor performance. 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 typically leads to either high throughput with large delay, or low throughput with small delay. One example is backpressure routing [3], which routes packets dynamically based on the amount of congestion in the network. As we shall see in §V, when the per device queue size, B, is large, it leads to high throughput at the cost of large delay, since packets flow through the network following the lowest congestion gradient at each device, and may take many hops to reach the destination. When B is small, it leads to lower delay at the cost of low throughput since many packets are dropped. In general, it is difficult to achieve both high throughput and low latency simultaneously under dynamic traffic conditions.

III. Related Work

Routing in wireless networks. The literature on routing strategies for wireless networks is extensive, ranging from early protocols such as DSR [1] and AODV [2] for ad-hoc networks, various protocols for delay and disruption tolerant networks (DTNs) [9], to protocols suited for resource constrained sensor [10] and IoT networks [11]. Most of these protocols, however, are designed for specific scenarios and so do not generalize well to new scenarios. For instance, ad hoc network protocols assume a connected network despite possible device mobility, while DTN protocols assume a disconnected network. Sensor networks often have unique traffic patterns, such as data being sent from many source devices to one sink device, which introduce additional routing requirements. In comparison, our DRL-based routing strategy is able to generalize to scenarios with different levels of network connectivity and congestion.

RL and DRL-based routing. Q-routing [5] is the first RL-based routing protocol with subsequent work adding memory [12] and confidence estimates [13]. More recent work considers a wider variety of network types and focuses on DRL [6], using DNNs [14] to approximate the RL policy. In centralized settings, one DRL agent makes decisions for all devices using one DNN, such as for traffic engineering or software defined networks [15]–[18]. In distributed settings, each DRL agent makes decisions independently using its own DNN, which is advantageous when centralized control is not practical, such as in a wireless network. However, existing work on distributed routing using DRL has limitations. For instance, in [19] a DNN is trained for each device, using device specific information such as the last k actions taken and the next m destinations of packets in queues, which limits scalability (the largest network that they test contains 25 devices and 56 links). The work in [20] found that experience from different agents is incompatible and leads to ineffective learning. Recent work [21] considers device mobility, but focuses on scenarios with a few fixed flows and up to 50 devices.

While we also focus on distributed routing, our study is the first that applies relational DRL to this problem. The use of relational features allows our approach to scale to much larger networks (e.g., 100 devices and $>$100 links) and to more diverse traffic scenarios than existing studies, by removing the need to use device IDs in training (such as for choosing a packet's next hop). As a consequence, we are able to train a DNN using data from all DRL agents, while also allowing each DRL agent to independently use the DNN to make decisions during testing. Our packet-centric approach also allows us to transform the problem into an episodic task, and our use of extended-time actions allows for faster learning.

GNN-based routing. Our use of relational features, in particular how we aggregate neighbor features, is similar in spirit to the aggregation function in a graph neural network (GNN) [22]. Several studies leverage the generalization capability of GNNs for routing so that the learned strategies are generalizable to other topology and traffic intensity. The study in [23] relies on supervised learning. The studies in [24], [25] combine DRL and GNNs for centralized routing; designing such models for a distributed setting is much more challenging.

IV. A Reinforcement Learning Model for Routing

The goal of reinforcement learning (RL) [4] is to learn to choose actions to maximize expected future reward. RL uses a Markov decision process (MDP) to describe an agent's environment. An MDP comprises a set of states (S), a per state set of actions ($\mathcal{A}(s)$), a reward function, and a state transition function. State transitions are assumed to be Markovian: the probability of the next state $s' \in S$ depends only on the current state $s \in S$ and action $a \in \mathcal{A}(s)$. RL assumes that these state transition probabilities are *not* known, but that samples from the environment can be generated. The Q-value for each (s, a) pair estimates the expected future reward for an agent, when starting in state s and taking action a. To learn, the agent observes (s, a, r, s') at each time step, where r is the immediate reward. The Q-value function is then updated via:

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

where $0 \leq \gamma \leq 1$ is a discount factor which indicates the relative value of present and future rewards, and $0 < \alpha \leq 1$ is a learning rate. Once learned, the optimal action in state s is the one with the highest Q-value.

When the MDP has a small number of states and actions, an RL agent can learn a Q-value function using Q-learning (see [26]). 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 this work, we use DNNs [14] for function approximation. Each state s and action a are translated into a set of features via the functions $f_s(s)$ and $f_a(a)$. These features are then used as input to the DNN, to produce as output an approximate Q-function $\hat{Q}(f_s(s), f_a(a))$. When training the DNN for each observation (s, a, r, s'), we use $f_s(s), f_a(a)$ as input and y, defined as follows, as output (sampling from $\hat{Q}$):

$$y = r + \gamma \cdot \max_{a' \in \mathcal{A}(s')} \hat{Q}(f_s(s'), f_a(a')). \quad (1)$$

A. Formulating an MDP for Routing

Consider a wireless device v and packet p at the front of v's queue. State features are derived from the packet p, device v, and neighbors of v. Action features are derived from the neighbor u under consideration for the role of packet p's next hop. When links are dynamic, only those neighbors for which there is currently a link are considered as possible actions. The specific features we use are relational (see §IV-B). The next hop for p is found by inputting the features into a trained DNN (see §IV-F). The output of the DNN is a Q-value indicating the expected future reward of choosing neighbor u as packet p's next hop given the state and action features.

Learning the decision model that will be applied for the packet agent at each device is computationally expensive, so training takes place offline. Once the decision model has been learned, it is copied to each device and "frozen." During use the parameters of the model are fixed, though the feature inputs vary. This allows fast decision making at each device.

B. Relational State and Action Features

Relational features are independent of the network topology and traffic on which the DNN is trained. All packets in the network use the same relational features for routing decisions, though the values of those features for individual packets may differ. A simple example of such features is given in Fig. 1(a). We omit features like device ID, packet destination ID, and any other features containing identifying information specific to a device or packet that would prevent generalization.

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

State features. For a packet p at device v at time t, with one-hop neighbor set $Nbr(v)$, the state features are a function, $f_s(p, v, t)$ computed from the following packet, device, and neighbor features at time t.

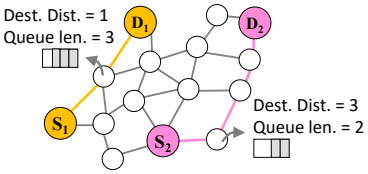
1) *Packet features,* $f_{pkt}(p, v, t)$. These are features derived from the packet p itself. We use i) p's TTL field and ii) p's location in device v's queue.
2) *Local device features,* $f_{device}(v, p, t)$. These are features derived from the device v at which the packet is currently located. We use i) the estimated distance from device 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 degree.
3) *Aggregated neighbor features,* $f_{nbr}(Nbr(v), p, t)$. These are features aggregated over all neighbors of device v. We first compute the local features, $f_{device}(u, p, t)$, 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.

The state features then comprise the three sets of features:

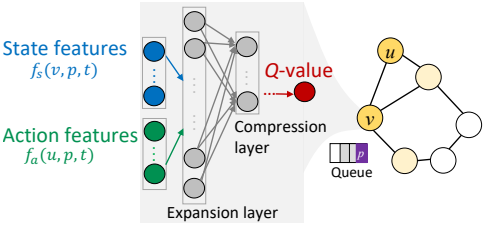
$$f_s(p, v, t) \quad = \quad f_{pkt}(\cdot) \cup f_{device}(\cdot) \cup f_{nbr}(\cdot).$$

Let f_i be the value of feature i and f_i^{max} be its maximum. 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, $f_s(\cdot), f_{pkt}(\cdot), f_{device}(\cdot), f_{nbr}(\cdot)$ all refer to the normalized features. We set the maximum destination distance to N, the maximum queue length to B, the maximum degree to N, and the maximum TTL to L, see Table II.

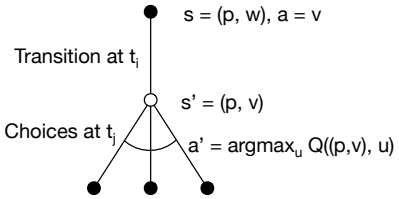
Action features. Each action at time t selects a next hop for the packet p that is at the front of device v's queue. A packet can choose either to stay at its current device or transition to one



(a) Packets can have different destinations, D_1 and D_2 respectively, but still use the same relational features, destination distance and queue length, just with different values.



(b) Using a DNN to make routing decisions. Packet p is considering the action of moving from device v to u at time t, so feeds the associated features into the DNN to obtain the Q-value of action u.



(c) The Q-learning backup diagram for a packet p which goes from device w to v at timestep t_i and then must choose among neighboring devices $u \in \mathcal{A}(v)$ at timestep t_j.

Fig. 1: Diagrams illustrating our algorithm operation.

of the neighboring devices. Let $\mathcal{A}(v) = Nbr(v) \cup \{v\}$. 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 local device features of u:

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

C. Reward Function

RL agents optimize a reward function by prediction of the expected future reward for each state s, action a 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 device 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-D, we describe 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 defined to be equivalent to receiving $r_{transition}$ for infinite timesteps.

D. Packet Agents vs. Device 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 *device-centric decisions*, each device 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 device's queue; it may choose to stay at the current device 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 device form a sequence. While the device-centric approach may seem more natural since we typically think of wireless devices as making routing decisions, here we use the packet-centric approach (see Fig. 1(b)) 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).

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 device queue enforce fairness among its packets: e.g., only the packet at the front of the queue gets to choose to move to another device at each time step.

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 device, it waits for some time in the device's queue, with no opportunity to make a routing decision. This scenario is a natural case for extended-time actions, or options [7], 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 for an option that starts at time step t_i and ends at time step t_j is (see [7]):

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

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. (1).

Here, we consider only reward functions that are constant for every timestep over the life of the option (see §IV-C), 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:

$$y = R(t_j - t_i, r_c) + \gamma^{(t_j-t_i)} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a') \qquad (2)$$

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 option in this domain: terminal (packet delivery or drop) and non-terminal (transitions from one device to another). On packet delivery or drop, the option takes only

TABLE I: Information in row of data for (state, action) pair.

Symbol	Meaning
$id(p)$	id of packet p
$id(v)$	id of device v
t_i	time that packet arrives at device v
t_j	time that packet departs device v
$f_s(p, v, t_j)$	state features at t_j for device v
$f_a(u, p, t_j)$	action features at t_j for neighbor u
$r(p, v)$	observed reward
b	flag: whether u was selected at t_j
Calculated columns	
$Q((p, v), u, t_j)$	estimated value of going from v to u
$\max_{u \in \mathcal{A}(v)} Q((p, v), u, t_j)$	max value over all u rows for (p, v)
Joined columns	
$f_s(p, w, t_i)$	state features at t_i for packet p device w
$f_a(v, p, t_i)$	action features at t_i for neighbor v

a single time step and the next state s_{t_j} is the terminal state. The sample of return for delivery is:

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

On non-terminal transitions the sample of 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 device queue at the end of a training round 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.

From this point on, to be consistent with §IV-B, we use "actions" rather than "options" to refer to extended-time actions.

F. Function approximation

We use a DNN [14] to approximate the value function. Since we assume all devices use the same DNN to make decisions, data from all devices can be pooled into a single large training set. During testing, each device independently uses its own copy of the DNN.

Our DNN architecture has 4 layers as in Fig. 1(b): 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 input to our DNN is a state, action pair (s, a) represented through their feature vectors. The output is a single neuron which estimates the Q-value of (s, a). For packet p at device v considering moving to device u at time t, this takes the form of a Q-function with feature inputs: $Q(f_s(p, v, t), f_a(u, p, t))$. We use the following shortened notation:

$$Q((p, v), u, t) \quad = \quad Q(f_s(p, v, t), f_a(u, p, t)).$$

Action selection. Different numbers of actions are available to packets at different devices. Thus, when packet p makes a decision at device v, the feature set $(f_s(\cdot), f_a(\cdot))$ for each device $u \in \mathcal{A}(v)$ is fed into the DNN as input to obtain a list of estimated $Q((p, v), u)$ values, one Q-value for each possible action. These Q-values can then be fed into any action selection mechanism. Here, we use ϵ-greedy with ϵ set as in Table II.

Algorithm 1 Steps executed for one round of training.

Input:
nn: a new randomly initialized DNN for this round
$data$: collected from time 0 up to and including this round
for *k = 1 to # of Q-learning iterations* **do**
  Calculate target y
  Estimate $Q((p, v), u, t_j)$ for all rows using nn and add column to $data$
  Take max over actions: $\max_{u \in \mathcal{A}(v)} Q((p, v), u, t_j)$, add column
  Add $y = R(t_j - t_i, r(p, v)) + \gamma^{t_j - t_i} \max_{u \in \mathcal{A}(v)} Q((p, v), u, t_j)$
  Find previous device w **and filter**
  Filter to select those rows with $b = 1$ (only actions chosen)
  Join $data$ with itself on $id(p)$ and t_i matched with t_j, labelling the t_i device w, and t_j device v
  Remove rows with unfinished actions
  Fit nn
  Improve estimate of $Q((p, w), v, t_i)$ using: input $f_s(p, w, t_i), f_a(v, p, t_i)$, target y
end

Data collection. In each training round, we gather a sequence of experience tuples $\cdots, (s, a, r), (s', a', r'), \cdots$, using the current DNN to choose actions. We record data only for packets that make a decision, i.e., the packet p at the front of the queue at each device v. For every possible action $u \in \mathcal{A}(v)$ available to p at v at time step t_j, we record the information in Table I as a row in our data. The row that contains the action selected by p has flag b set to one. The columns marked Calculated and Joined are added later by the algorithm.

Training. For each packet p which transitions from device w to device v, arriving at v at time step t_i and departing at t_j, we use Algorithm 1 to do the "backup" shown in Fig. 1(c). This update improves the estimate of $Q((p, w), v)$. Non-relational columns are used to find the device w that packet p departed at time t_i (to chain together (s, a, r, s')). However, only $f_s(p, v, t_j)$ and $f_a(u, p, t_j)$ are used as input to the DNN to estimate $Q((p, v), u, t_j)$ for each $u \in \mathcal{A}(v)$ when calculating the target values. We use Eq. (2) to calculate the training target.

V. Performance Evaluation

We evaluate our approach using a discrete-time packet-level network simulator that we have implemented in Python3. This simulator provides the environment in which the DRL agents are trained and all routing algorithms are tested. Tables II to IV show our simulation parameters. We use Keras v.2.3.1 [27] and Tensorflow v.1.14.0 [28] to implement the DNN. Training and testing was done using the MIT SuperCloud and Lincoln Laboratory Supercomputing Center [29].

A. Simulation Settings

Our goal is to identify the wireless scenarios for which our DRL approach performs well (i.e., delivers the most packets with low delay) and to test how well a DRL agent trained on one scenario is able to generalize its learned routing policy to unseen scenarios. We thus explore a wide range of scenarios that differ in topology, link dynamics, and traffic.

Topologies. As shown in Fig. 4, we consider two types of network topologies: i) a square grid lattice, and ii) a geometric

TABLE II: Simulation parameters

Symbol	Meaning	Value
N	# of network devices	9 to 100
ϵ_{train}	Training exploration rate	.1
ϵ_{test}	Testing exploration rate	0
γ	RL discount rate	0.99
$r_{transition}$	Transition reward	-1
r_{drop}	Drop reward	$r_{transition}/(1-\gamma)$
$r_{delivery}$	Delivery reward	0
L	Packet time-to-live	200
B	Maximum queue size	50 or $50N$
T_{train}	# of training timesteps	30,000 or 49,000
T_{test}	# of testing timesteps	100,000
T_{round}	# of timesteps per round	1000

random graph where devices are randomly placed in a unit square, and two devices are connected by a link if they are within a given transmission radius.

Link dynamics. To model link dynamics, we use a 2-state Markov model. Links are i.i.d and stay up from one time step to the next with probability p_{up} (and transition down with probability $1-p_{up}$), and stay down from one timestep to the next with probability p_{down} (and transition up with probability $1-p_{down}$). 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, $\pi = (1-p_{up})/(2-p_{up}-p_{down})$.

Medium access control. On each timestep, we loop through all devices in random order and allow each device to transmit a single packet that was received or generated in a previous timestep. Doing this imposes per-device capacity constraints.

Traffic generation. Sources and destinations are selected uniformly randomly, with the constraint that a source never equals its associated destination. The flows present then change over time: new flow arrivals are generated according to a Poisson distribution with parameter λ_F; flow durations are generated by sampling an exponential distribution with parameter λ_D. Packet arrivals are generated according to a Poisson distribution with parameter λ_P, where λ_P is the average number of new packets generated per timestep on a given flow. A simulation starts with $\lambda_F\lambda_D$ initial flows.

Queue size. Each device in the network maintains a packet queue with a maximum buffer size, B, beyond which additional packets are dropped.

B. Training and Testing Scenarios

We organize the wireless network scenarios that we consider along three dimensions: connectivity, size, and congestion.

1) Network connectivity: As in [30], network connectivity influences the kind of routing strategy that is appropriate. Ad hoc routing strategies work well in networks that are mostly well-connected, while delay tolerant routing strategies work well in networks that are mostly disconnected. To measure network connectivity we use *algebraic connectivity*, defined as the second-smallest eigenvalue of the normalized Laplacian matrix of a graph [31]. The larger the value, the more well-connected is the network topology. When algebraic connectivity equals 0, the network is disconnected. Here, we vary network connectivity (and thus algebraic connectivity) in terms of i) *link dynamics*, and ii) *path redundancy*.

TABLE III: Network topologies and link dynamics

Network scenario	radius	p_{up}	p_{down}	π
Static lattice topology	-	1.0	0.0	1.0
Dynamic lattice topology	-	0.8	0.2	0.8
Delay tolerant lattice topology	-	0.5	0.4	0.55
Static random topology	0.5	1.0	0.0	1.0
Delay tolerant random topology	0.3	0.5	0.4	0.55

TABLE IV: Network traffic scenarios.

Traffic scenario	λ_F	λ_D	λ_P
Low traffic congestion	$.002N/25$	5000	.05
High traffic congestion	$.002N/25$	5000	.2

To vary *link dynamics* we vary the values of p_{up} and p_{down} for our 2-state link model in §V-A. For large p_{up} and small p_{down}, the network is connected; as p_{up} decreases and p_{down} increases, the probability that a contemporaneous end-end path exists between two devices decreases. The different link dynamics we use are shown in Table III. The special case with $p_{up}=1$ and $p_{down}=0$ has no link dynamics, and is referred to as *static*. For certain settings of p_{up} and p_{down} (e.g., when $p_{up}=0.5$ and $p_{down}=0.4$), the network is mostly disconnected and is referred to as *delay tolerant*. When there are some link dynamics, but the network is nonetheless mostly connected, is referred to as *dynamic*.

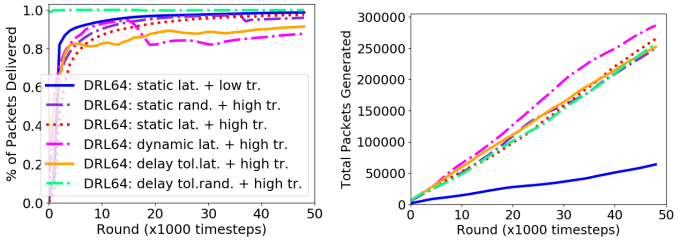
To vary *path redundancy* we vary the topology. Topologies with high redundancy should have shorter paths, and can better handle congestion. The lattice topology is relatively sparsely connected, while the random topology is densely connected. As in Table III, we consider two random topologies, one with a transmission radius of 0.5, and one with a radius of 0.3.

2) Network size: Varying the network size, N, affects connectivity differently depending on the underlying topology. For the lattice, increasing N decreases connectivity. For the random geometric topology, since devices are always distributed within the unit square, increasing N increases connectivity. In our testing results, we vary N from 9 to 100; for training, we use two sizes, $N=25$ and $N=64$. The testing results shown use the $N=64$ training results as the $N=25$ training results do not generalize as well to larger networks.

3) Network congestion: We consider two traffic scenarios, low and $high$, shown in Table IV. For both scenarios, the amount of traffic generated varies over time due to the Poisson distributed arrivals of flows and packets. During periods of increased traffic, there is correspondingly increased congestion.

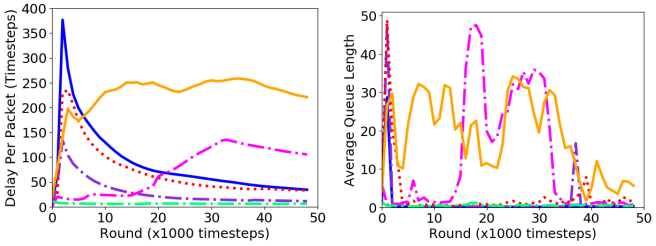
When these traffic scenarios are used with other topologies, link dynamics, and values of N, the amount of traffic congestion they generate will vary. For instance, congestion scales super-linearly with N in a lattice topology when traffic is uniformly random. Similarly, for the same traffic scenario on the same topology, the introduction of link dynamics will decrease the available bandwidth and increase congestion.

4) Training and testing scenarios: We vary connectivity (which varies the destination distance and neighbor features, see Fig. 3(a)) and congestion level (which varies the queue length feature, see Figs. 3(b) to (d)). We obtain the following six scenarios, which use the parameter settings in Tables III and



(a) % of Packets delivered

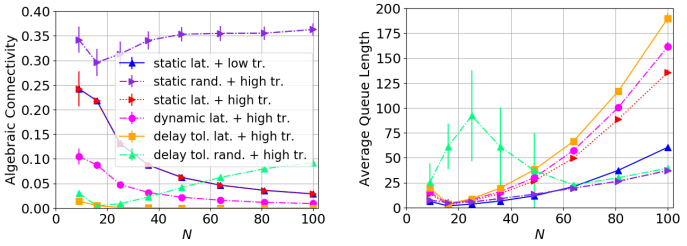
(b) Total packets generated



(c) Delay per packet

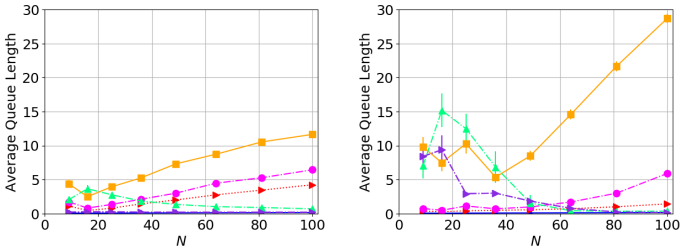
(d) Average queue length

Fig. 2: Training performance of DRL agents, for $N = 64$. The legend indicates the training conditions.



(a) Algebraic connectivity of scenarios. Algebraic connectivity is a topology measure, so is independent of traffic.

(b) BP queue lengths during testing. BP uses $B = 50N$, so sees larger queue lengths than the other algorithms which use $B = 50$.



(c) SP queue lengths during testing.

(d) Queue lengths seen for a DRL64 agent trained on the delay tol. lat. + high traffic and then tested on all scenarios.

Fig. 3: Example network connectivity and congestion levels.

IV: i) *static lattice + low traffic*, ii) *static lattice + high traffic*, iii) *dynamic lattice + high traffic*, iv) *delay tolerant lattice + high traffic*, v) *static random + high traffic*, and vi) *delay tolerant random + high traffic*. We use only one low traffic scenario as the high traffic scenarios are more challenging.

C. Routing Algorithms

We compare the performance of the following three routing algorithms. All algorithms use only local information (such as distance or destination queue length) obtained from a device's neighbors during testing. We do not model control packets since local (1-hop) traffic typically causes relatively little congestion compared to non-local ($>$ 1-hop) traffic [32].

1) Shortest path routing (SP): We implement shortest path routing as a distance vector algorithm using hop count as cost. We modify the algorithm slightly to accommodate link dynamics. Because devices are stationary though the links present may change, we assume that once a device has a link to a neighbor device, that link continues to be present in the distance calculations. Because link changes are i.i.d., this means the distance vector algorithm we use converges on the true shortest path distance in terms of hop count. However, only those neighbors for which there are links present are considered as possible next hops when a routing decision is made. For each packet, then, the next hop for the shortest path that is currently available is chosen. If no next hop is available, then the packet stays at the device.

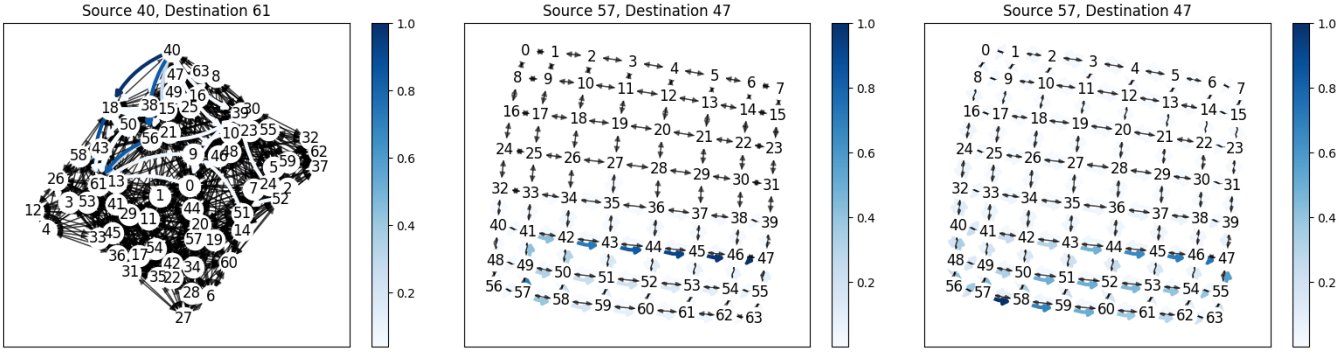
2) Deep reinforcement learning (DRL): To train, our algorithm can use either simulated network data (as we do here) or historical data recorded from a network of interest. A DRL agent estimates the distance feature using the same distributed distance vector algorithm that is used by shortest path routing. Because packets may take very long paths while the DRL agent is learning a policy during training, we use the relatively high TTL value of $L = 200$ compared to the expected path length, to prevent packets from always being dropped before the DRL agent has had sufficient time to learn.

3) 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. BP thus relies on network congestion to route well, but is the throughput optimal strategy for certain conditions.

Unlike SP or our DRL approach which forward the packet at the front of a device's queue, BP chooses the best packet from anywhere in the queue to forward. Thus, BP requires the use of large queues to ensure packets are never dropped due to a queue being full. We set the maximum queue size for BP to be $B = 50N$, which allows each device to (virtually) keep a separate queue for every destination, compared to $B = 50$ for the other algorithms. We further evaluate BP with $B = 50$ and find that it delivers many fewer packets than the other algorithms (results omitted in the interest of space).

D. Results

We first overview our DRL agent training performance in §V-D1. Then we evaluate how well the trained DRL agents generalize their learned policies by testing their performance on the lattice (§V-D2), and the random scenarios (§V-D3). In our simulation results, we plot the following metrics. Let D_t



(a) Static random + high traffic (b) Static lattice + high traffic (c) Delay tolerant lattice + high traffic

Fig. 4: Example learned policies of DRL64 agents from training data. Plots show the number of times each action (link) is selected for the specified flow, normalized by the maximum number of times any link is selected for the flow.

(G_t) be the total number of packets delivered (generated) by round t, and let Y_t be the total delay of packets delivered by round t. Then we compute (i) the *% of packets delivered* by round t with D_t/G_t; (ii) the *delay per packet* by round t with Y_t/D_t; (iii) *average queue length* at round t by averaging over all device queue lengths on the last timestep in the round; and (iv) *algebraic connectivity* at round t on the network topology on the last timestep in the round.

1) Learning curves and learned policies: We train DRL agents for each scenario in §V-B4 for $N = 64$, labeled DRL64. Training is divided into T_{train}/T_{round} rounds, see Table II; we use 10 epochs and a batch size of 32.

Fig. 2(a) shows that when training on the static scenarios, the DRL64 agents quickly learn policies that deliver all packets. The DRL64 agent trained on the delay tolerant random scenario also converges quickly, due to high connectivity (giving many possible paths) combined with high traffic (providing feedback on which paths not to use). In comparison, the DRL64 agents trained on the dynamic and delay tolerant lattice scenarios show fluctuation in packets delivered as the number of flows varies over time. While the mean number of flows is given by λ_F, the actual number of flows at any given time can be higher (or lower) and cause queues to build up. Because of the extra time needed for the DRL64 agents to converge when trained on the dynamic and delay tolerant lattice scenarios, for testing we train these DRL64 agents with $T_{train} = 49,000$ timesteps; all other DRL64 agents used are trained with $T_{train} = 30,000$ timesteps.

Fig. 4 shows example learned policies. In the delay tolerant lattice + high traffic scenario in Fig. 4(c), the DRL64 agent learns to distribute traffic over additional paths to the destination, which has similarities with the row-first column-next policy for optimal shortest path routing in a lattice [33].

2) Lattice generalization: This section shows results for when DRL64 agent training and testing are done on a lattice scenario, see Fig. 5. The top row of Fig. 5 shows the packet delivery rate while the bottom row shows packet delay for the four lattice testing scenarios (as marked in the captions), each using all four lattice training scenarios (as marked in the legend). Each point in Fig. 5 (and Figs. 3 and 6) shows the 95% confidence interval computed over 50 simulation runs.

Figs. 5(a) and (e) show testing on the *static lattice + low traffic* scenario. Due to low traffic, queues are mostly empty (see Fig. 3), and SP is optimal. Indeed, both the DRL64 agents and SP deliver all packets. BP, however, delivers significantly fewer packets as network size increases, because there is insufficient traffic for it to effectively build a congestion gradient.

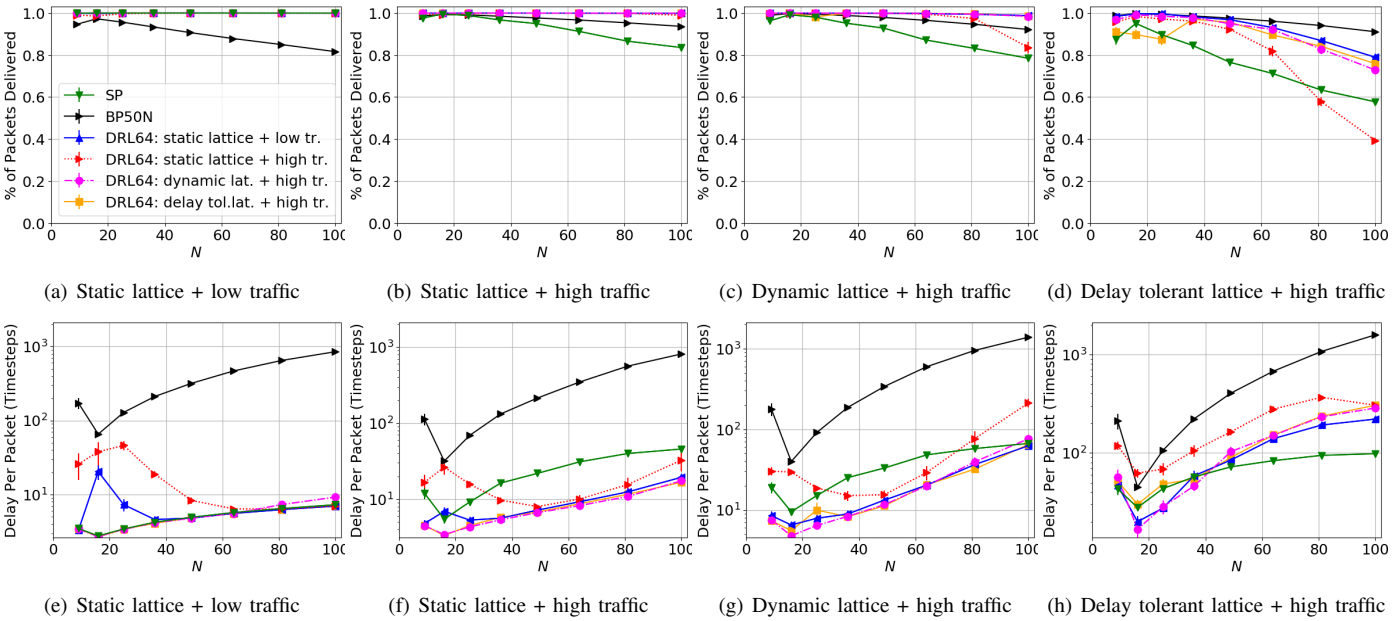
Figs. 5(b) and (f) show testing on the *static lattice + high traffic* scenario. Despite the increased traffic, the DRL64 agents still deliver all packets. SP, however, delivers significantly fewer packets. Conversely, BP is now able to build an effective congestion gradient and delivers more packets than SP but with much higher packet delay than the DRL64 agents.

Figs. 5(c) and (g) show testing on the *dynamic lattice + high traffic* scenario. Although the topology and traffic are the same as those in Figs. 5(b) and (f), due to the addition of link dynamics, the available bandwidth is reduced. Now, all DRL64 agents deliver all packets except that trained on the static lattice + high traffic. Neither SP nor BP are able to deliver all packets once the network size is sufficiently large.

Finally, Figs. 5(d) and (h) show testing on the *delay tolerant lattice + high traffic* scenario. This is the only lattice scenario in which the network is predominantly disconnected (see Fig. 3(a)). Due to the increased link dynamics, as N increases the network becomes sufficiently congested that not all packets can be delivered. BP now delivers the most packets in the larger network sizes because it can choose any packet in the queue to send rather than only the one at the front. The DRL64 agents, except that trained on the static lattice + high traffic, deliver close to 80% of traffic for $N = 100$, despite being restricted to choosing the packet at the front of the queue to send, and using the much smaller queue size of $B = 50$.

3) Random geometric generalization: Fig. 6 shows results when training DRL64 agents on lattice or random scenarios, and then testing on random scenarios. This evaluates how well DRL64 agents generalize to diversely connected scenarios.

Figs. 6(a) and (c) show testing on the *static random + high traffic* scenario. As shown in Fig. 4(a), the static random



(a) Static lattice + low traffic (b) Static lattice + high traffic (c) Dynamic lattice + high traffic (d) Delay tolerant lattice + high traffic

(e) Static lattice + low traffic (f) Static lattice + high traffic (g) Dynamic lattice + high traffic (h) Delay tolerant lattice + high traffic

Fig. 5: Testing generalization of DRL64 agents on the lattice topologies; connectivity decreases as N increases. The training conditions (using $N = 64$) are shown in the legend and the testing conditions are shown in the figure captions.

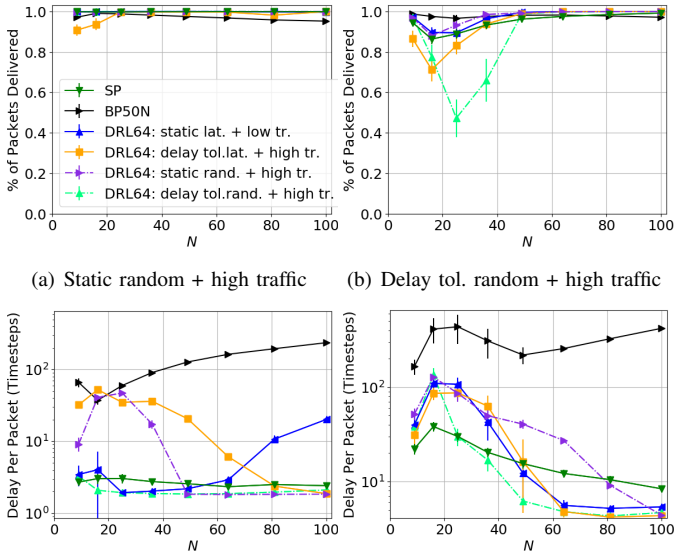
topology has significantly higher and more variable connectivity than does the lattice, leading to many more actions to consider for each packet and varying the number of neighbors over which features are computed. Figs. 6(a) shows that SP and DRL64 agents deliver all packets for the larger network sizes, which are more connected. For the smaller, less connected network sizes, all but the DRL64 agent trained on the delay tolerant lattice + high traffic scenario are able to deliver most packets. Due to the decreased traffic congestion as N increases, however, BP does not deliver all packets for the larger network sizes. BP also typically has much higher packet delay.

Figs. 6(b) and (d) show testing on the *delay tolerant random + high traffic* scenario, which is disconnected for small N but connected for large N, see Fig. 3(a). We now see a split in DRL64 agent performance. Agents trained on the delay tolerant scenarios deliver all packets and with the lowest delay for large N but perform the worst of all strategies for small N; these agents have highly optimized their strategies to well-connected topologies with dynamic links, which does not generalize well to poorly connected topologies with dynamic links. Agents trained on the static scenarios also deliver all packets for large N but with higher delay, while for small N they perform as well as or better than SP. SP only delivers all packets for $N = 100$, the most connected scenario. While BP never delivers all packets, it delivers the most packets for small N though with the highest delay, but again BP is aided by its larger queue size and ability to choose any packet in the queue.

E. Discussion

Our simulation results highlight the importance of training DRL agents on scenarios that are sufficiently diverse and cover the testing state space. For example, the DRL64 agent trained on the delay tolerant random + high traffic scenario (in Fig. 6) performs exceedingly well for the $N = 64$ version of the scenario on which it was trained, but does not generalize well to the smaller versions of this scenario which are sparsely connected and highly congested. The reason is that during training, the $N = 64$ version of the scenario lacks the occasional spikes in delay and queue length seen for the other DRL64 agents, see Fig. 2, so cannot generalize its learned policy to these parts of the state space during testing.

Conversely, we have results (figures not shown) for training a DRL64 agent on the static random + high traffic scenario and then testing on all lattice scenarios. This agent generalizes



(a) Static random + high traffic (b) Delay tol. random + high traffic

(c) Static random + high traffic (d) Delay tol. random + high traffic

Fig. 6: Testing generalization of DRL64 agents on the random geometric topologies; connectivity increases as N increases.

well to the lattice scenarios, due to the better coverage of the state space during training. While in this work we trained individual DRL64 agents on different scenarios, to understand the network conditions under which our approach works well, ideally, a single DRL agent should be trained on a diverse set of scenarios that sample the space of target testing conditions.

The flexibility to choose any packet in the queue to send as with BP, rather than only the packet at the front of the queue as with the DRL64 agents and SP, is valuable when links are up infrequently and not all packets can be delivered. For instance, when a link is finally up, the "best" packet in the queue can be chosen for the link. Or in the case of congestion, when all packets cannot be delivered, those packets that will be more quickly delivered can be selected from the queue to be forwarded. Incorporating this kind of flexibility into our DRL agent design would start to merge our packet-centric approach with that of a device-centric approach.

Finally, there is an interesting trade-off between the maximum queue size B and discount factor γ, since the larger B is, the longer the amount of time options may take, which impacts reward. We leave exploring this trade-off to future work.

VI. Conclusions and Future Work

In this work, we have designed a novel distributed routing algorithm using relational deep reinforcement learning. Our algorithm generalizes to diverse network scenarios through the use of relational features, packet-centric decisions, and extended-time actions, and outperforms shortest path routing and backpressure routing 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 and increasing flexibility in choice of packet to send.

Acknowledgements

The authors thank the anonymous reviewers for helpful comments. The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC and consultation resources that have contributed to the research results reported within this paper.

References

[1] 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.

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

[3] 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*, 1990.

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

[7] 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.

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

[9] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in *Proc. of SIGCOMM*, 2004.

[10] 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.

[11] 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.

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

[13] S. Kumar and R. Miikkulainen, "Confidence-based Q-routing: an on-line adaptive network routing algorithm," in *Proc. of Artificial Neural Networks in Engineering*, 1998.

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

[15] 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.

[16] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route with deep RL," in *NIPS Deep Reinforcement Learning Symposium*, 2017.

[17] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM*, 2018, pp. 1871–1879.

[18] J. Suarez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Feature engineering for deep reinforcement learning based routing," in *Proc. of IEEE ICC*, 2019, pp. 1–6.

[19] X. You, X. Li, Y. Xu, H. Feng, and J. Zhao, "Toward packet routing with fully-distributed multi-agent deep reinforcement learning," in *IEEE RAWNET workshop, WiOpt 2019*, Avignon, France, June 2019.

[20] 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, 2019.

[21] 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.

[22] Peter Battaglia et al., "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf

[23] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Big-DAMA*, 2018.

[24] 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.

[25] 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.

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

[27] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[28] Martín Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[29] A. Reuther, J. Kepner, C. Byun, S. Samsi, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell *et al.*, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *High Performance extreme Computing Conference*, 2018.

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

[31] F. R. Chung, "Lectures on spectral graph theory," *CBMS Lectures, Fresno*, vol. 6, pp. 17–21, 1996.

[32] R. Ramanathan, R. Allan, P. Basu, J. Feinberg, G. Jakllari, V. Kawadia, S. Loos, J. Redi, C. Santivanez, and J. Freebersyser, "Scalability of mobile ad hoc networks: Theory vs practice," in *MILCOM*, 2010.

[33] G. Barrenetxea, B. Berefull-Lozano, and M. Vetterli, "Lattice networks: capacity limits, optimal routing, and queueing behavior," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 492–505, 2006.