Learning to Route in Mobile Wireless Networks

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

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

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

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

ABSTRACT
Designing effective routing strategies for mobile wireless networks is challenging due to the need to seamlessly adapt routing behavior to spatially diverse and temporally changing network conditions. In this work, we use deep reinforcement learning (DeepRL) to learn a scalable and generalizable single-copy routing strategy for such networks. We make the following contributions: i) we design a reward function that enables the DeepRL agent to explicitly trade-off competing network goals, such as minimizing delay vs. the number of transmissions per packet; ii) we propose a novel set of relational neighborhood, path, and context features to characterize mobile wireless networks and model device mobility independently of a specific network topology; and iii) we use a flexible training approach that allows us to combine data from all packets and devices into a single offline centralized training set to train a single DeepRL agent. To evaluate generalizeability and scalability, we train our DeepRL agent on one mobile network scenario and then test it on other mobile scenarios, varying the number of devices and transmission ranges. Our results show our learned single-copy routing strategy outperforms all other strategies in terms of delay except for the optimal strategy, even on scenarios on which the DeepRL agent was not trained.

CCS CONCEPTS
• Networks → Routing protocols; Network protocol design; Mobile ad hoc networks; • Computing methodologies → Sequential decision making; Neural networks.

KEYWORDS
routing, mobile networks, reinforcement learning, neural networks

1 INTRODUCTION
Routing in mobile wireless networks is a fundamental problem that has applications in many real-world networks, such as robot or drone networks [22, 38], mobile Internet of Things (IoT) [6], and sensor networks [21]. Designing effective routing strategies for these networks is challenging, however, due to the need to seamlessly adapt routing behavior to spatially diverse and temporally changing network conditions. While many mobile routing strategies have been proposed, these strategies are typically hand-crafted for specific types of mobile networks. For instance, DSR [20], AODV [34], OLSR [13] and DSDV [35] have been developed for ad hoc networks, while epidemic flooding [47] and other more resource-efficient routing schemes like Seek-and-Focus [41, 42] have been developed for delay tolerant networks.

An alternative to using hand-crafted routing strategies for mobile networks is to use Reinforcement Learning (RL) [44] to learn an adaptive routing strategy. RL focuses on the design of intelligent agents: an RL agent interacts with its environment to learn a policy, i.e., which actions to take in different environmental states. Importantly, the same RL agent can be run independently in multiple network locations, but will still react appropriately to the local device state in each location by using the parts of the RL agent policy relevant for that state. Because changes in state lead to changes in RL agent behavior, a single policy can be applied to a state space that includes, for instance, both well-connected and poorly connected parts of a network. By using function approximation like deep neural networks (DNNs) [7] as in deep reinforcement learning (DeepRL) to approximate the policy, the RL agent can learn to generalize from its training experience to unseen network conditions and scale its learned policy to larger networks.

Recent works on designing DeepRL based routing strategies for wireless networks can be divided into two categories: those that consider network topologies with limited link dynamics [11, 14, 30, 31, 43, 48–52], and the many fewer works that consider networks with mobile devices [23, 27]. Those works that do consider device mobility, however, have limitations: for instance, focusing on establishing network connectivity [27] or considering smaller networks with very few flows [23]. Our goal in this work is to use DeepRL to learn a single-copy routing strategy for mobile wireless networks.
that is able to both scale to larger networks, and generalize to different levels of network connectivity. To achieve this goal, we focus on addressing three key challenges to learning how to route in mobile wireless networks. i) How to balance complex performance trade-offs? ii) How to handle the diversity of mobile scenarios that should be included during training? iii) How to design features to effectively model the dynamics of mobile wireless networks? In this work, we address these challenges, making the following contributions.

i) Designing RL reward. In §3.3.3, we show how the reward function used by the DeepRL agent during training can be designed to trade-off competing network performance goals, such as minimizing delay vs. the number of transmissions per packet delivered.

ii) Flexible training. In §3.3.5, we propose the use of relational features, which are independent of a specific network topology, to combine data over time from all packets and devices into a single offline centralized training set to train a single DeepRL agent. The learned strategy is then able to be used independently by individual devices during testing.

iii) Characterizing device mobility. In §4, we develop a novel set of relational features to characterize mobile wireless networks. Our neighborhood features capture network dynamics over time and ensure that the number of features does not change as a function of neighbors. Our path features estimate properties of time-varying paths between one device and another. Our context features give additional context to the other features.

iv) A generalizeable DeepRL agent. In §5, we train our DeepRL agent on one mobile network scenario with 25 devices and a transmission range of 50m and then test it on a number of other mobile scenarios, varying the number of devices from 25 to 100 and the transmission range from 30m to 80m. The DeepRL agent outperforms all other strategies including seek-and-focus routing [41] except for the optimal strategy, even on scenarios on which the DeepRL agent was not trained.

The rest of this paper is organized as follows. In §2, we overview related work. In §3, we describe how we formulate a DeepRL agent for routing. In §4, we overview the features used by the agent. In §5, we present simulation results. Finally, in §6, we summarize our main results and provide directions for future research.

2 RELATED WORK

In this section, we overview related work in two areas: DeepRL-based routing strategies and modeling of network dynamics.

2.1 DeepRL for Routing

Centralized RL-based approaches to routing [43, 48], in which one RL agent makes decisions for all devices using one DNN, are less practical for mobile networks than distributed approaches, in which each RL agent makes decisions independently using its own DNN, due to communication costs and the risk of out-of-date information when gathering global network information in real-time. Indeed, early works [8, 12, 16, 19, 26, 36] use distributed routing but focus on less scalable and less generalizable table-lookup based approaches instead of using DNNs.

Recent works on DNN-based routing can be divided into strategies that consider network topologies with limited link dynamics [11, 14, 30, 31, 43, 48–52], and the many fewer strategies that consider networks with mobile devices [23, 27]. Those approaches that do consider device mobility have limitations: for instance, [27] focuses on choosing which relay to activate to establish network connectivity, rather than routing, while [23] focuses on scenarios with a few fixed flows and up to 50 devices. Our proposed DeepRL approach to routing focuses specifically on mobile networks and we show our approaches can generalize a strategy learned on a network with 25 devices to networks with 100 devices and varying transmission ranges.

2.2 DNNs to Model Network Dynamics

Many approaches [3, 9, 10, 18] have been proposed to model device mobility, but do not provide a unified way to identify the features necessary to generalize routing to very different kinds of mobility and network connectivity. In contrast, DNNs provide a natural way to unify different approaches to modeling mobility: features that work well for one kind of mobility can easily be included along with features that work well for other kinds of mobility. The DNN representation itself supports generalization to unseen types of device mobility and network scenarios. Through training, a DeepRL agent learns the relationship between mobile network features and how best to route, and encodes that in a policy represented using a DNN.

While previous works [29, 40, 46, 53] have used variations of DNNs to model mobility, they are limited due to the use of global information and do not handle significant link dynamics. While several works [1, 17, 37] leverage the generalization capability of Graph Neural Networks [5, 24, 39] for routing or network optimization they again do not handle link dynamics or device mobility. In comparison, our use of relational features, combined with our neighborhood features, allows us to define features independent of a specific network topology, which is critical for modeling mobility. No prior work on routing that we have seen uses relational features except for [30] which focuses on stationary networks rather than the mobile networks we consider here. The use of a DNN to represent the DeepRL agent’s routing strategy, given our state and action features, additionally lets us take advantage of the generalization capabilities of DNNs.
3 LEARNING-BASED MOBILE ROUTING

In this section, we overview how we design a DeepRL agent that learns how to route in a mobile wireless network.

3.1 Reinforcement Learning Background

Reinforcement Learning (RL) [44] is designed to learn to choose actions to maximize expected future reward. In RL, the environment is modeled using a Markov decision process (MDP). An MDP comprises a set of states ($S$), a per state set of actions ($A(s)$), and a reward function. It also includes a Markovian state transition function, in which the probability of the next state $s' \in S$ depends only on the current state $s \in S$ and action $a \in A(s)$. RL assumes that these state transition probabilities are not known, but that samples of transitions of the form $(s, a, r, s')$ can be generated. From these samples, the algorithm learns a $Q$-value for each $(s, a)$ pair. A $Q$-value estimates the expected future reward for an agent, when starting in state $s$ and taking action $a$. Once learned, the optimal action in state $s$ is the one with the highest $Q$-value. Function approximation can be used to find approximate $Q$-values when the state space is too large for exact computation.

Here, we use DNNs for function approximation.

Routing actions often involve multiple time steps: e.g., after a packet arrives at a new device, it may wait for a neighboring device to come into range. Thus, routing is a natural case for hierarchical RL [4, 15, 45]. Using hierarchical RL with periods of time in which no decisions are made requires less data to be collected, and allows reward signals to be backed up over multiple time steps in a single update. Consequently, Q-learning proceeds more quickly.

3.2 Key Challenges for Mobile Routing

Designing an RL strategy for routing in mobile wireless networks requires addressing two key challenges: i) managing competing network goals; and ii) designing a training approach that works for a dynamically changing topology.

Mobile networks typically have competing performance goals, such as to maximize throughput, minimize delay or power, or treat all traffic flows fairly. The use of RL, however, allows the goals of a routing strategy to be directly specified and rewarded, instead of indirectly specified by dictating routing behavior. In §3.3.3, we show how the reward function used by the DeepRL agent during training can be designed to reflect different preferences for network performance.

In a mobile wireless network, changes in the network topology mean the set of neighbors that a device has is changing. As illustrated in Fig. 1, this means that the set of actions a device has is correspondingly changing, making routing a more difficult problem than in static wireless networks. As illustrated in Fig. 1, at time $t$, $v$ has three neighbors, ${u_1, u_2, u_3}$, while at time $t+1$, $v$ has only a single neighbor, $w_1$. Correspondingly, the possible actions for $v$ at time $t$ (i.e., either transmitting a packet to one neighbor in $\{u_1, u_2, u_3\}$ or keeping the packet) differ significantly from its actions at time $t + 1$ (i.e., either transmitting a packet to $w_1$ or keeping the packet). Our use of relational features allows us to be agnostic to topology changes, enabling us combine data from all packets and devices regardless of connectivity over time into a single offline centralized training set to train a single DeepRL agent, while learning a single strategy that can be used independently by individual devices. In §3.3.5, we overview such flexible learning in more detail.

3.3 Mobile Routing MDP Formulation

In this work, we focus on learning a single-copy routing strategy: that is, we do not allow the DeepRL agent to make multiple copies of a packet to reduce delivery delay. This allows us to make the simplifying assumption that there is no constraint on the number of packets forwarded by a device in a given timestep. As a consequence, we do not have to model fairness between flows and can model each packet as an independent agent with only indirect effects on other packets. While [30] also models each packet as an independent agent, they assume that at each timestep only the packet at the front of the queue can be forwarded, rather than our assumption of all packets.

Like in [50] for stationary devices, however, we make packets, rather than devices, the decision-making agents, to more clearly propagate reward back from a delivered packet to the intermediate devices that forwarded the packet.
Mobile network

Figure 2: Overview of how we construct the state $f_s$ and action $f_a$ for use by the DeepRL agent to make routing decisions.

We introduce two other rewards, for actions that lead to a packet being dropped, $r_{\text{drop}} = r_{\text{transmit}} / (1 - \gamma)$, or delivered to its destination, $r_{\text{delivery}} = 0$, where $\gamma \in [0, 1]$ is the RL discount factor. The drop reward is defined to be equivalent to receiving a reward of $r_{\text{transmit}}$ for infinite timesteps. All reward settings that we use are summarized in Table 2.

3.3.4 Decision-making. Fig. 2 overviews how packets use the DNN that has been trained to represent the learned DeepRL routing strategy. A packet at a device makes a routing decision by activating the DNN once for each possible action available at the device, passing in features that describe the current state from the packet’s point of view, and the action under consideration. The DNN then returns a Q-value for each state, action pair, and the packet chooses the action with the best Q-value.

3.3.5 A flexible training approach. During online testing, all features are estimated in a distributed way and all decision-making is done independently by each DeepRL agent. But training a DeepRL agent online is problematic, as doing so is subject to communication and computation constraints. The use of relational features (see §4.2) allows us to combine data from all packets and devices over time as the network changes into a single offline centralized training set to train a single DeepRL agent, while learning a single online distributed policy which generalizes to unseen network scenarios and which can be used independently by individual devices. This approach to training is particularly important when working with mobile networks: even a single mobile network scenario includes network connectivity changing over time and space, making it essential to be able to include such diversity during training.

4 MOBILE NETWORK FEATURES

In this section, we first overview the challenges when designing mobile network features, and then describe our features.

4.1 Key Challenges for Mobile Features

Designing good mobile features requires addressing three key challenges: i) generalizing to different network scenarios, ii) characterizing network dynamics, and iii) managing uncertain and out-of-date information.
To support generalizability, the same routing strategy should be able to be used at different devices and in different mobile network scenarios. To do this, we use relational features, described in §4.2, to represent the states and actions used by the DeepRL agent. Relational features, such as delay to destination, model the relationship between network devices, instead of a specific device. No prior work on routing that we have seen uses relational features except [30] which focuses on designing features for stationary networks rather than the mobile networks we consider here. Using relational features, however, requires they be appropriately normalized, so that similar ranges of feature values in one network scenario have meaning in another network scenario. This requires careful design for mobile networks, see §4.4.1.

A mobile network’s topology can change spatially and temporally due to device mobility and wireless interference. Features should model these changes so that the best action can be selected for a given state of the network. In §4.2, we propose path and context features that incorporate historical information about the structure of the network graph. We use neighborhood features to model the spatial changes.

Given bandwidth constraints in mobile networks, frequent exchange of features with devices multiple hops away is not feasible. Instead, exchange of primarily local feature information should occur when devices meet. This can lead to out-of-date information. In §4.4.2, we describe what local and non-local information devices exchange, and how devices keep track of the freshness of state information received from other devices. Through these exchanges, devices can build a more global view of the state of the network. Devices may also have uncertainty in their feature estimates when computing more complex features. In §4.2.3, we use renewal theory [25] to statistically model this uncertainty and obtain improved delay estimates for our path features.

### 4.2 Mobile Features

In this section, we overview the five classes of relational features, packet, device, path, neighborhood, and context, that we propose, specifically tailored to model mobile networks. For each of these classes, we describe the features that are used by a packet $p$ to choose its next hop. We assume that $p$’s destination is device $d$ and that $p$ is currently located at device $v$ which has neighbors $u \in Nbr(v)$.

#### 4.2.1 Packet features, $f_{packet}(p)$

We define packet features to be features that are computed as a function of information found in packet $p$. Our DeepRL agent currently uses only one packet feature: packet $p$’s time-to-live (TTL). Every packet should contain a TTL field that is decremented when a packet is forwarded to another device.

#### 4.2.2 Device features, $f_{device}(v, d)$

We define device features to be features that are computed as a function of device $v$’s information and destination $d$’s device ID. Our DeepRL agent uses the following device features computed at the current timestep $t$: i) device $v$’s queue length, ii) device $v$’s queue length considering only packets for destination $d$, iii) device $v$’s node degree, and iv) device $v$’s node density, computed as the fraction of neighbors that $v$ has out of the $N$ devices in the network. Even when a network has little congestion, queue length information can still be useful when choosing next hops. For instance, next hop devices that do not have any packets in the same flow as $p$ may be preferred to better distribute traffic.

Our DeepRL agent additionally keeps track of two device features, the $x$ and $y$ location coordinates of device $v$, which are only used to compute the Euclidean distance, a path feature described later in this section. While we assume every device knows its own location, locations for other devices are only obtained when two devices meet and exchange features. Consequently, location (and thus distance) features can be out-of-date. See §4.4.2 for more feature exchange details.

#### 4.2.3 Path features, $f_{path}(v, d)$

We define path features to be features that describe the time-varying path from a device $v$ to a destination device $d$. Unlike the other features discussed so far, path features are computed using not just current device information but also historical information. Consequently, these features have some associated uncertainty. Our DeepRL agent uses the following path features.

i) **One-hop delay** from device $v$ to destination $d$, denoted as $D_{v,d}^{(1)}$. This is the expected delay for device $v$ to directly transmit packet $p$ to device $d$, i.e., in one-hop, without using any intermediate device. Consider the contact trace between two devices $v$ and $d$ as illustrated in Fig. 3, where the black bars represent duration of time when the two devices are within transmission range, and the space in between represents the duration of time when the devices are out of transmission range. Let $T_{v,d}$ and $M_{v,d}$ denote the average inter-meeting time and meeting duration between $v$ and $d$, respectively. For a packet arriving at device $v$ at time $t$, the expected delay for transmitting the packet directly from $v$ to $d$ is

$$
D_{v,d}^{(1)} = 1 \times \Pr(v, d \text{ in contact}) + R_{v,d} \times \Pr(v, d \text{ not in contact})
$$

$$
= 1 \times \frac{M_{v,d}}{T_{v,d} + M_{v,d}} + R_{v,d} \times \frac{T_{v,d}}{T_{v,d} + M_{v,d}},
$$

where $R_{v,d}$ is the expected duration from time $t$ to when the two devices meet each other, given that at time $t$ the
two devices are not in range of each other. Following renewal theory [25], we estimate $R_{od}$ as $R_{od} = T_{od}/2 + \sigma_{od}^2/(2T_{od})$, where $\sigma_{od}^2$ is the estimated variance of intermeeting time. We obtain the estimate of $T_{od}$, $\sigma_{od}^2$ and $M_{od}$ over time in an online manner, and then use Eq. (1) to estimate $D_{od}^{(1)}$.

ii) Two-hop delay from device $v$ to destination $d$, denoted as $D_{od}^{(2)}$. This is the minimum delay for $v$ to transmit a packet to $d$ through any intermediate neighbor $u$. $D_{od}^{(2)}$ is the sum of the delay over the two hops, from $v$ to $u$ and then from $u$ to destination $d$.

\[
D_{od}^{(2)} = \min_{u \in \text{nbr}(v)} \left( D_{v,u}^{(1)} + D_{u,d}^{(1)} \right).
\]

While we could additionally have features for three-hop delay and so on, we expect diminishing benefits as we increase the number of hops past two.

iii) Euclidean distance from device $v$ to destination $d$. This is calculated using $v$’s current $x$ and $y$ location coordinates, and device $v$’s recorded (and mostly out-of-date) location coordinates of destination $d$ (see the device features). Estimating the Euclidean distance to $d$ in addition to the one-hop and two-hop delays distinguishes the situation where $d$ is far away from $v$ from the situation where connectivity between $d$ and $v$ is poor.

4.2.4 Neighborhood features, $f_{nbrhood}(v,d)$. We define neighborhood features to be features that are computed over the neighbors $\text{Nbr}(v)$ of a device $v$ at the current timestep $t$. Because neighborhood features describe the region around a device without considering device IDs, they are relational. Our neighborhood features update as the spatial structure of the network updates.

Our DeepRL agent uses the following neighborhood features: for each device and path feature, $f_i \in f_{device}(u,d) \cup f_{path}(v,d)$, we compute the minimum, maximum, and average over device $v$’s current neighborhood, $\text{Nbr}(v)$ (as in [30]). These features compress the information obtained from a variable number of neighbors into a fixed size vector to use as input to the routing DNN in Fig. 2.

4.2.5 Context features, $f_{context}(p,u)$. We define context features to be features that provide context for other features. In this work, our context features form part of the action features, $f_a(\cdot)$. For a packet $p$ at a device $v$ that has a neighbor $u \in \text{Nbr}(v)$ under consideration as a possible next hop, the context features indicate whether $p$ has recently visited $u$. Specifically, we assume that each packet $p$ stores in its packet header the last $N_{\text{history}}$ device IDs that it visited, where $N_{\text{history}}$ is a predetermined constant. Let $H(p,i)$ be the ID of the device that packet $p$ visited $i$ hops ago, for $0 \leq i < N_{\text{history}}$. When $i = 0$, then $H(p,0)$ is the device at which $p$ is currently located. IDs themselves as features would tie our features to specific devices in a specific network. Instead, to make these features relational, we use a sequence of Boolean features, $b_i$, defined as:

\[
b_i = \begin{cases} 1, & \text{if } u = H(p,i), \\ 0, & \text{otherwise.} \end{cases}
\]

The use of packet history as a context feature helps reduce the number of unnecessary packet transmissions made. For instance, even if a possible next hop device $u$ has promising features for reaching the destination, if packet $p$ recently visited $u$, then $u$ may be a less good next hop than it seems based solely on $u$’s other feature values.

4.3 Computing $f_i(\cdot)$ and $f_a(\cdot)$

The feature sets defined in this section are used in both the state and action descriptions. The state features for a packet $p$ with destination $d$ that is currently located at device $v$ are: $f_i(v,p,d) = f_{\text{packet}}(p) \cup f_{\text{voice}}(v,d) \cup f_{\text{path}}(v,d) \cup f_{\text{nbrhood}}(v,d)$. The action description for a candidate next-hop action $u \in \text{Nbr}(v) \cup \{v\}$ reuses many of the same features as the state description, but is defined in terms of a device $u \in \text{Nbr}(v) \cup \{v\}$ rather than just $p$’s current location of $v$, and includes the context features: $f_a(u,p,d) = f_{\text{device}}(u,d) \cup f_{\text{path}}(u,d) \cup f_{\text{nbrhood}}(u,d) \cup f_{\text{context}}(p,u)$.

4.4 Feature Estimation

4.4.1 Feature normalization. Our goal when normalizing features is to re-scale them into the range of approximately 0 to 1. Mobility makes normalization more challenging, however, as the un-normalized ranges of feature values, such as for node degree, may be very different in different networks. To address this, we make the normalization factor a function of network properties when possible, such as the (approximate) number of devices or the (approximate) size of the area in which devices are moving. In this way, the normalization can better adapt to new network environments.

Table 1 summarizes how we normalize the packet, device, and path features. We use the same normalization constants during both training and testing to keep feature values consistent. The neighborhood features do not need normalization as they are a function of other normalized non-neighborhood features; the context features also do not need normalization as they are Boolean valued.

4.4.2 Feature exchange. We assume a device is able to discover when another device enters or leaves its transmission range through the use of “heartbeat” control messages. Suppose there is a packet $p$ with destination $d$ at device $v$, and suppose $v$ has neighbors $u \in \text{Nbr}(v)$. Then $v$ obtains from each neighbor $u$ the following information: i) the features $f_{\text{device}}(u,d) \cup f_{\text{path}}(u,d) \cup f_{\text{nbrhood}}(u,d)$, ii) $u$’s current $x$ and $y$ location coordinates, timestamped with $u$’s current clock, and iii) the $x$ and $y$ location coordinates for every
Table 1: Feature normalization for packet, device, and path features. For each feature $f_i$, normalization is done using the following equation $(f_i + 1)/(D + 1)$, where 1s are added to avoid zero values for features. See Table 2 for values of parameters.

| Feature                  | Denominator, $D$ |
|--------------------------|------------------|
| Packet TTL               | $TTL_{train}$    |
| Queue length             | 30               |
| Per-destination queue length | 30             |
| Node degree              | 10               |
| Node density             | $N$             |
| One-hop delay            | 1000             |
| Two-hop delay            | 1000             |
| $x$-coordinate location  | $d_x$           |
| $y$-coordinate location  | $d_y$           |
| Euclidean distance       | 2                |

Table 2: Simulation parameters

| Symbol     | Meaning                                      | Value       |
|------------|----------------------------------------------|-------------|
| $N_{train}$| # of devices during training                 | 25, 64      |
| $N$        | # of devices during testing                  | 25, 64, 100 |
| $d_x$      | Mobility area $x$-location max               | 500m        |
| $d_y$      | Mobility area $y$-location max               | 500m        |
| $X_{train}$| Training transmission ranges                 | 30m, 50m    |
| $X_{test}$ | Testing transmission ranges                  | 30m to 80m  |
| -          | Device average speed                         | 3 m/s       |
| -          | Device speed delta                           | 2 m/s       |
| $B$        | Maximum queue size                           | 200         |
| $\lambda_F$| Mean flow arrivals                           | .001$N/25$  |
| $\lambda_D$| Mean flow duration                           | 5000        |
| $\lambda_p$| Mean packet arrivals                         | .01         |
| $TTL_{train}$| Initial packet TTL for training             | 300         |
| $TTL_{test}$| Initial packet TTL for testing               | 3000        |
| $\epsilon_{train}$ | RL training exploration rate                  | .1          |
| $\epsilon_{test}$  | RL testing exploration rate                   | 0           |
| $\gamma$     | RL discount rate                             | 0.99        |
| -             | RL # of iterations                           | 100         |
| $r_{delivery}$| RL delivery reward                           | 0           |
| $r_{stay}$    | RL stay reward                               | -1          |
| $r_{transmit}$| RL transmit reward                           | -1, -2, -10|
| $r_{drop}$    | RL drop reward                               | $r_{transmit} / (1 - \gamma)$ |
| $N_{history}$ | Length of device visit history               | 0, 5        |
| $T_{train}$  | # of training timesteps                      | 90,000      |
| $T_{test}$   | # of testing timesteps                       | 100,000     |
| $T_{cooldown}$| # of timesteps with no traffic              | 10,000      |
| $T_{round}$  | # of timesteps per round                     | 1000        |
| -            | DNN training dropout rate                    | 0.2         |

Other device $w$, which $u$ has either recorded directly from $w$ or received indirectly from another device, along with the recording’s timestamp, i.e., the time on $w$’s clock of when the coordinates were recorded. Device $v$ then uses ii) and iii) to update its recording of the $x$ and $y$ location coordinates for every other device, overwriting older recordings with more recent recordings for a device $w$, comparing the timestamps associated with the recordings. Because these timestamp comparisons always compare timestamps received from the same device $w$, no clock synchronization is needed.

Device $v$ now has all of the information that packet $p$ needs to make a decision about which next hop to choose. To obtain $f_u(v, p, d)$ and $f_a(u, p, d)$ from §3.3.4, the DeepRL agent for packet $p$ computes $f_{packet}(p)$, $f_{device}(v, d)$, and $f_{path}(v, d)$ using local information at device $v$; $p$ computes $f_{nbrhood}(v, d)$, $f_{adevice}(u, d)$, $f_{path}(u, d)$, and $f_{nbrhood}(u, d)$ using the features received by $v$ from $u \in Nbr(v)$; finally, $p$ computes $f_{context}(p, u)$ using only $u$’s ID in addition to the information carried in packet $p$’s header fields.

5 SIMULATION RESULTS

In this section, we first overview our simulation setup, and then describe simulation results. Our simulations are done in a custom discrete-time event simulator written in Python.

5.1 Methodology

5.1.1 Device mobility. We use BonnMotion [2] to generate mobility traces for devices moving under the steady-state random waypoint mobility model [28, 32, 33]. BonnMotion generates a mobility trace as a sequence of waypoints for each device. For example, device $v$’s mobility is a sequence of tuples: $(t_0^v, x_0^v, y_0^v)$, $(t_1^v, x_1^v, y_1^v)$, and so on. Because these waypoint tuples occur and are thus recorded at arbitrary points in time, we must convert the tuples to match our simulation timesteps, where each timestep corresponds to one second. To do this, in our simulations, the location of device $v$ at each timestep $t$ is calculated by first finding the time interval $[t_k^v, t_{k+1}^v]$ in which $t$ falls in, and then calculating the location at time $t$ via a linear interpolation of $(x_k^v, y_k^v)$, $v$’s location at $t_k$, and $(x_{k+1}^v, y_{k+1}^v)$, $v$’s location at $t_{k+1}$.

For training, we use mobility traces of $N_{train} = 25$ devices moving in a 500m x 500m area at an average speed of 3 m/s with a speed delta of 2 m/s and transmission ranges of $X_{train} = 30m$ and $X_{train} = 50m$. Although not shown, we also trained on larger, more dense networks, but found the smaller networks provided more diversity in the training data.

For testing, we generate separate mobility traces for $N = 25, 64, 100$ devices using the same size area and range of speeds. In our results, we explore the impact of varying the transmission range, $X_{test}$ from 30m to 80m on testing performance. Fig. 4 shows how varying the number of devices and transmission ranges varies the network connectivity in the testing scenarios: both very poorly connected and very well connected mobile scenarios are used in testing.

5.1.2 Network traffic. We vary the amount of traffic over time by modeling flow and packet arrivals and flow durations. To generate traffic, we model flow arrivals using a Poisson distribution with parameter $\lambda_F = .001N/25$, scaling the number of flows as a function of the number of devices
and Seek-and-Focus strategies from [41] as state-of-the-art strategies that trade-off delay with number of transmissions.

**Optimal routing.** Given complete information about current and future network connectivity, optimal routing calculates the minimal hop paths that achieve the minimal delivery delay for all packets. Instead of formulating an optimization problem to obtain the optimal strategy, we obtain it through epidemic routing. Epidemic routing creates many copies for a packet and distributes them to the network. For those packets that reach the destination with the minimum latency, we further find the packet that reached the destination with the minimum number of hops. This routing strategy thus minimizes delay, while maintaining a good trade-off in terms of resources. It is, however, not practical since it requires knowing the current and future network topology.

**Direct transmission routing.** This strategy forwards a packet one hop only, directly from the source to the destination. This strategy is optimal when the goal is to minimize the number of transmissions per packet.

**Utility-based routing.** This strategy is the utility-based algorithm found in [41]. Each device $v$ maintains for each device $d$ in the network, a timer, denoted as $\tau_{v}(d)$, which is the time elapsed since device $v$ last met device $d$, as illustrated in Fig 3. We implemented the timer transitivity as defined in [41]: when two devices, $u$ and $v$ encounter each other, if $\tau_{u}(d) < \tau_{v}(d) - t(d_{u,v})$, where $t(d_{u,v})$ is the expected time for a device to move a distance of $d_{u,v}$, then $\tau_{v}(d)$ is set to $\tau_{v}(d) = \tau_{u}(d) + t(d_{u,v})$. The insight is that for many mobility models, a smaller timer value on average implies a smaller distance to the device. The timer evaluates the “utility” of a device in delivering a packet to another device.

A device $v$ chooses the next hop for a packet as follows: $v$ first determines which neighboring device, $u$, has the smallest timer to the packet’s destination, $d$. If $\tau_{v}(d) > \tau_{u}(d) + U_{th}$, i.e., the timer of $v$ to destination $d$ is larger than the timer of $u$ to $d$ by more than the utility threshold, $U_{th}$, the packet is forwarded to device $u$; otherwise, the packet is not forwarded. We ran a set of experiments using utility-based routing, varying the parameter $U_{th}$ over $\{5, 10, 20, 30, 40, 50, 60, 100\}$, optimized for $N = 25$ and $X_{test} = 50$, which are the same settings for which the DeepRL agent used in testing was trained on. We use $U_{th} = 10$ in our simulations as this made a good trade-off between delay and number of transmissions.

**Seek-and-focus routing.** In Seek-and-focus routing [41], utility-based routing (focus phase) and random forwarding (seek phase) are combined. If the smallest timer (among all neighboring devices) to the destination is larger than the focus threshold $U_{f}$, the packet is in seek phase, forwarded to random neighbor with probability $prob$. Otherwise, the packet

---

5.1.3 **MAC protocol.** At each timestep, every device in the network is given an opportunity to transmit packets. Because we focus on single copy routing (see §3.3.4), network congestion is less of a concern (and our results in Fig. 7 show the average queue length is always less than five). Due to this lack of congestion combined with the often sparse network connectivity of mobile networks, we assume unlimited link bandwidth and allow each device to transmit an unlimited number of packets per timestep.

5.1.4 **Routing strategies.** In our simulations, we compare the performance of five routing strategies, including a delay minimizing strategy (optimal) and a transmission minimizing strategy (direct transmission) to give bounds on the performance of the DeepRL agent. We also compare with the Utility

---

**Figure 4:** In (a) and (b) we show that the different numbers of devices and transmission ranges we consider in our simulations encompass the continuum from disconnected to well-connected networks. In (c) and (d), we show the inter-meeting time $T_{u,d}$ and meeting duration $M_{u,d}$ of §4.2.3: these metrics are computed online between each possible pair of devices, and so are independent of the number of devices.

$N$ in the network. We model flow durations using an exponential distribution with parameter $\lambda_{D} = 5000$ and packet arrivals on flows using a Poisson distribution with parameter $\lambda_{P} = 0.01$. A simulation starts with $\lambda_{P} \lambda_{D}$ initial flows.

Each device has a packet queue with a maximum length of $B = 200$, beyond which additional packets are dropped. A packet’s TTL field is initialized to $TTL_{test} = 3000$. We use a large TTL to ensure no packet is dropped. To ensure that normalization of the feature TTL values was consistent in both training and testing for the DeepRL agent, we normalize the TTL feature by $TTL_{train} = 300$.

---

In §5.1.3, we describe the MAC protocol. At each timestep, every device in the network is given an opportunity to transmit packets. Because we focus on single copy routing (see §3.3.4), network congestion is less of a concern (and our results in Fig. 7 show the average queue length is always less than five). Due to this lack of congestion combined with the often sparse network connectivity of mobile networks, we assume unlimited link bandwidth and allow each device to transmit an unlimited number of packets per timestep.

In §5.1.4, we compare the performance of five routing strategies, including a delay minimizing strategy (optimal) and a transmission minimizing strategy (direct transmission) to give bounds on the performance of the DeepRL agent. We also compare with the Utility and Seek-and-Focus strategies from [41] as state-of-the-art strategies that trade-off delay with number of transmissions.

**Optimal routing.** Given complete information about current and future network connectivity, optimal routing calculates the minimal hop paths that achieve the minimal delivery delay for all packets. Instead of formulating an optimization problem to obtain the optimal strategy, we obtain it through epidemic routing. Epidemic routing creates many copies for a packet and distributes them to the network. For those packets that reach the destination with the minimum latency, we further find the packet that reached the destination with the minimum number of hops. This routing strategy thus minimizes delay, while maintaining a good trade-off in terms of resources. It is, however, not practical since it requires knowing the current and future network topology.

**Direct transmission routing.** This strategy forwards a packet one hop only, directly from the source to the destination. This strategy is optimal when the goal is to minimize the number of transmissions per packet.

**Utility-based routing.** This strategy is the utility-based algorithm found in [41]. Each device $v$ maintains for each device $d$ in the network, a timer, denoted as $\tau_{v}(d)$, which is the time elapsed since device $v$ last met device $d$, as illustrated in Fig 3. We implemented the timer transitivity as defined in [41]: when two devices, $u$ and $v$ encounter each other, if $\tau_{u}(d) < \tau_{v}(d) - t(d_{u,v})$, where $t(d_{u,v})$ is the expected time for a device to move a distance of $d_{u,v}$, then $\tau_{v}(d)$ is set to $\tau_{v}(d) = \tau_{u}(d) + t(d_{u,v})$. The insight is that for many mobility models, a smaller timer value on average implies a smaller distance to the device. The timer evaluates the “utility” of a device in delivering a packet to another device.

A device $v$ chooses the next hop for a packet as follows: $v$ first determines which neighboring device, $u$, has the smallest timer to the packet’s destination, $d$. If $\tau_{v}(d) > \tau_{u}(d) + U_{th}$, i.e., the timer of $v$ to destination $d$ is larger than the timer of $u$ to $d$ by more than the utility threshold, $U_{th}$, the packet is forwarded to device $u$; otherwise, the packet is not forwarded. We ran a set of experiments using utility-based routing, varying the parameter $U_{th}$ over $\{5, 10, 20, 30, 40, 50, 60, 100\}$, optimized for $N = 25$ and $X_{test} = 50$, which are the same settings for which the DeepRL agent used in testing was trained on. We use $U_{th} = 10$ in our simulations as this made a good trade-off between delay and number of transmissions.

**Seek-and-focus routing.** In Seek-and-focus routing [41], utility-based routing (focus phase) and random forwarding (seek phase) are combined. If the smallest timer (among all neighboring devices) to the destination is larger than the focus threshold $U_{f}$, the packet is in seek phase, forwarded to random neighbor with probability $prob$. Otherwise, the packet...
Learning to Route in Mobile Wireless Networks

is in the focus phase, and the carrier of the packet performs utility-based routing with utility threshold $U_{th}$. In addition to $U_f$, prob, and $U_{th}$, Seek-and-focus routing has three more parameters: the time-until-decoupling which controls the amount of time a device is not allowed to forward a packet back to a device it received the packet from, $T_{focus}$ which controls the maximum duration to stay in focus phase before going to re-seek phase (random forwarding of the packet to get out of a local minimum), and $T_{seek}$ which controls the maximum duration to stay in re-seek phase until going to seek phase (random forwarding of the packet until reaching a device with timer smaller than $U_f$).

To find the best parameters for Seek-and-focus, we ran a set of experiments exploring different combinations of parameter values. We vary $prob$ over $\{0.1, 0.2, 0.5, 0.9\}$, $U_{th}$ over $\{10, 100\}$, $U_f$ over $\{10, 20, 100\}$, $T_{focus}$ over $\{10, 20, 100\}$, $T_{seek}$ over $\{10, 50\}$, and time-until-decoupling over $\{10, 50\}$. We use $prob = 0.5$, $U_{th} = 100$, $U_f = 20$, $T_{focus} = 10$, $T_{seek} = 50$, time-until-decoupling = 10. We optimize these parameters for $N = 25$ and $X_{test} = 50$, which are the same settings on which the DeepRL agent used in testing was trained.

**DeepRL agent routing.** This strategy uses a DeepRL agent to make routing decisions. Fig. 5 plots DeepRL training performance and Table 3 lists the DeepRL agents we use in testing. Each training simulation is run for $T_{train}$ timesteps, where each timestep corresponds to one second. Training is done every $T_{round}$ timesteps using all data received up to that timestep but with a new neural network model. Our results in Fig. 5 show the DeepRL agent strategies converging; we discuss these results further in the next section.

To better understand the routing strategy being learned, Fig. 6 shows Q-values observed during training as a function of different state features. For example, Fig. 6(a) shows that Q-values are higher for low values of the distance and delay features, indicating that the DeepRL agent prefers forwarding a packet to next hops with low distance and delay.

5.2 Results

Figs. 5 to 6 show DeepRL agent training results while Figs. 7 to 8 show testing results. For all testing simulations in Figs. 7 to 8, no packets are dropped and all packets are delivered. All performance measures are computed over the delivered packets, such as the delay per packet delivered.

5.2.1 Impact of transmission reward, $r_{transmit}$. In our training results of Fig. 5, the different values of the transmission reward, $r_{transmit}$, make different trade-offs between delay and number of hops per packet. At each timestep the cumulative performance over all packets delivered up to the timestep is shown. When $r_{transmit} = -1$, there is no penalty
for transmission as a packet receives the same reward for transmitting as for staying at a device, since $r_{stay} = -1$. Correspondingly, we see that when $r_{transmit} = -1$, delay per packet delivered is the lowest among the different strategies, but the number of forwards per packet is the highest. Conversely, for $r_{transmit} = -10$, there is a large penalty for making a transmission, and so, while delay per packet delivered is the highest for this strategy, the number of forwards per packet is now the lowest. Though not shown, these results also hold true for $N_{train} = 64$ and $X_{train} = 30m, 50m$. In our testing results in Figs. 7 and 8, we use DeepRL agents trained with $r_{transmit} = -2$ to make decisions as it provides a good trade-off between delay and number of transmissions.

5.2.2 **Impact of network connectivity.** Fig. 7 compares the performance of two DeepRL agents trained on the network scenarios in Table 3 and then tested on other network scenarios varying the number of devices, $N$, and transmission range, $X_{test}$. Each simulation is run for $T_{test}$ timesteps, with traffic generated only for the first $T_{test} - T_{cooldown}$ timesteps; this ensures all packets are accounted for at the end of the simulation. We observe all packets delivered and no packets dropped in our testing results.

Each testing point is the average of 50 simulations; 95% confidence intervals are shown though are often too small to see. All performance metrics are averaged across all simulation runs for a given set of parameters, except for the maximum queue length which is the maximum seen in any of the simulation runs for a given set of parameters.

We see in Fig. 7 that except for the optimal strategy, the DeepRL agent has the lowest delay per packet delivered even for network scenarios on which it was not trained. We also see that as the network topology becomes more well connected (i.e., due to increasing $N$ or increasing transmission range $X_{test}$), the DeepRL agents start to have delay per packet delivered similar to the optimal strategy, albeit with more forwards per packet delivered. As the network topology becomes more sparse (i.e., due to decreasing $N$ or decreasing transmission range $X_{test}$), the DeepRL agents start to have delay that is significantly higher than the optimal strategy and approaching the delay of the other strategies.

The DeepRL agents have their highest delay per packet delivered for the $N_{train} = 25$ and $X_{train} = 30m$ scenario which is the most disconnected scenario considered in our simulations (see Fig. 4). Due to the few neighbors and relatively long intermeeting times between pairs of devices for this scenario, we expect including temporal neighborhood features would be beneficial, as well as taking into consideration predicted future neighbors when choosing next hop actions.

5.2.3 **Impact of context features.** While $r_{transmit}$ penalizes packet transmissions, the context history features themselves do not directly penalize transmissions. Instead, the context history features augment the state space to add context when actions are taken and ensure that actions that lead to unnecessary looping can be more easily identified.

In the training results of Fig. 5, the potential benefits of using context history are unclear. The testing results in Fig. 7, however, show the DeepRL agent with no context history has a consistently higher number of forwards per packet delivered despite the very similar delay per packet. This is an indication that the use of context history improves generalization of the learned routing strategy.

5.2.4 **Queue stability.** Figs. 7 (g) to (l) show the average and maximum queue lengths for the different strategies. The maximum queue length is the maximum queue length seen at any device over all simulations for a given setting of the network parameters. The average queue length is the average
Figure 7: Testing performance of the different routing strategies on different steady-state random waypoint scenarios.
of the queue lengths seen at all devices over all timesteps, and averaged over all simulations for a given parameter setting. We see in Figs. 7 (g) to (l) that the queue lengths are stable for all strategies. We also see that the maximum queue length can be significantly more than the average queue length, leading to possible bottlenecks in the network. The DeepRL agent and Seek-and-focus, however, have significantly lower maximum queue lengths than the Direct transmission or Utility strategies. The Optimal strategy is implemented as a multi-copy scheme with a recovery latency of zero, i.e., when the first packet copy is delivered to the destination with minimum number of hops, all other copies are removed from the network. Consequently the queue lengths for the Optimal strategy include all of the copies present until the first packet copy is delivered.

5.2.5 Transient behavior. Fig. 8 focuses on the initial routing performance when a simulation first begins, showing the first 30,000 of the \( N_{test} - N_{cooldown} = 90,000 \) simulation timesteps for the most extreme parameter settings. of Fig. 7.

In Fig. 8, we see that the different strategies can behave differently at the beginning of a simulation. For instance in Fig. 8 (f), the number of forwards for packets delivered in the first few rounds is quite high for the “No Context” DeepRL agent and then sharply drops even though the earlier number of forwards are still included. Even after the drop, though, the number of forwards is still higher than the other strategies, due the high connectivity of the network leading to many transmission opportunities but no context to avoid devices recently visited. In comparison, the “Context” DeepRL agent experiences a much smaller drop and also starts out at a much smaller number of forwards per packet delivered. We hypothesize that these drops are due to initial uncertainty of the DeepRL agent about the state of the network, with the “No Context” DeepRL agent more sensitive to uncertainty.

5.3 Discussion

All routing strategies must make some kind of trade-off between delay and number of transmissions. While the DeepRL agent uses its reward function to navigate this trade-off, Seek-and-focus uses a less straightforward approach. Among the six parameters of Seek-and-focus, we observe that varying the forwarding probability \( prob \) (while fixing the other five parameters) provides one way to manage this trade-off. But there is no clear way to set these parameters in coordination with \( prob \) to achieve a specific trade-off for a given network scenario. One solution would be to instead learn the optimal settings for the Seek-and-focus parameters.

Seek-and-focus also uses a timer (with transitivity) to evaluate the “utility” of a device for a destination. In contrast, the
DeepRL agent at each device collects and estimates mobility information from its neighbors. Among these features, the Euclidean feature directly estimates a device’s distance to a destination. Consequently, DeepRL’s assumption of device location is a potential advantage. The amount of control overhead due to sharing device locations by the DeepRL agent, however, is similar to the amount of overhead required to implement the timer transitivity used in Seek-and-focus.

6 CONCLUSIONS AND FUTURE WORK
In this work, we show that it is possible to use DeepRL to learn a scalable and generalizable single-copy routing strategy for mobile wireless networks. We leverage three key ideas: i) a reward function specifically crafted to trade-off competing network goals; ii) a novel set of relational features to characterize mobile networks; and iii) a flexible training approach. Our results show that our learned single-copy routing strategy outperforms all other strategies in terms of delay except the optimal strategy, even on scenarios on which the DeepRL agent was not trained.

There are a number of research directions to explore in future work. While we focus on learning a single-copy routing strategy in this work, we are also interested in learning multi-copy routing strategies. We would also like to refine the features we use. We expect that as the kinds of mobility the DeepRL agent sees become more diverse, more features will be needed to characterize the key differences in mobility and enable the DeepRL agent to generalize to a wide variety of mobile networks. Finally, we would like to evaluate our approach on an even broader set of mobility scenarios.

ACKNOWLEDGEMENTS
This material is based upon work supported by the National Science Foundation under Grant No. 2154190. Results presented in this paper were obtained using CloudBank, which is partially supported by the National Science Foundation under award #2154190. 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] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Russek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. Deep Reinforcement Learning meets Graph Neural Networks: exploring a routing optimization use case. https://arxiv.org/abs/1910.07421.
[2] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla, and Matthias Schwamborn. 2010. Bonnmotion: a mobility scenario generation and analysis tool. In Proceedings of the 3rd international ICST conference on simulation tools and techniques. 1–10.
[3] Fan Bai and Ahmed Helmy. 2004. A survey of mobility models. Wireless Adhoc Networks. University of Southern California, USA 206 (2004), 147.
[4] Andrew G Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. Discrete event dynamic systems 13, 1 (2003), 41–77.
[5] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. arXiv (2018). https://arxiv.org/pdf/1806.01261.pdf
[6] Oladay Bello and Sherali Zeadally. 2014. Intelligent device-to-device communication in the internet of things. IEEE Systems Journal 10, 3 (2014), 1172–1182.
[7] Yoshua Bengio. 2009. Learning deep architectures for AI. Now Publishers Inc.
[8] J. A. Boyan and M. L. Littman. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In Advances in neural information processing systems. 671–678.
[9] Han Cai and Do Young Eun. 2007. Crossing over the bounded domain: from exponential to power-law inter-meeting time in manet. In Proceedings of the 13th annual ACM international conference on Mobile computing and networking. 159–170.
[10] Tracy Camp, Jeff Boleng, and Vanessa Davies. 2002. A survey of mobility models for ad hoc network research. Wireless communications and mobile computing 2, 5 (2002), 483–502.
[11] Long Chen, Bin Hu, Zhi-Hong Guan, Lian Zhao, and Xuemin Shen. 2021. Multiagent Meta-Reinforcement Learning for Adaptive Multi-path Routing Optimization. IEEE Transactions on Neural Networks and Learning Systems (2021).
[12] S. P. Choi and D.-Y. Yeung. 1998. Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. Advances in Neural Information Processing Systems (1996), 945–951.
[13] Thomas Clausen, Philippe Jacquet, Cédric Adjih, Anis Laouti, Pascale Minet, Paul Mulhithaler, Amir Quyyum, and Laurent Viennot. 2003. Optimized link state routing protocol (OLSR). (2003).
[14] Valerio Di Valerio, Francesco Lo Presti, Chiara Petroilli, Luigi Picari, Daniele Spaccini, and Stefano Basagni. 2019. CARMA: Channel-aware reinforcement learning-based multi-path adaptive routing for underwater wireless sensor networks. IEEE Journal on Selected Areas in Communications 37, 11 (2019), 2634–2647.
[15] Thomas G Dietterich. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In ICML, Vol. 98. Citeseer, 118–126.
[16] Ahmed Elwhishi, Pin-Han Ho, Khirasagar Naik, and Basem Shihada. 2010. ARBR: Adaptive reinforcement-based routing for DTN. In 2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications. IEEE, 376–385.
[17] Fabien Geyer and Georg Carle. 2018. Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning. In Proc. of Big-DAMA.
[18] Robin Groenevelt, Philippe Nain, and Ger Koole. 2005. The message delay in mobile ad hoc networks. Performance Evaluation 62, 1-4 (2005), 210–228.
[19] Tianyi Hu and Yunsi Fei. 2010. An adaptive and energy-efficient routing protocol based on machine learning for underwater delay tolerant networks. In 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. IEEE, 381–384.
[20] David B. Johnson and David A. Maltz. 1996. Dynamic source routing in ad hoc wireless networks. In Mobile Computing. Kluwer Academic Publishers, 153–181.
