Learning an Adaptive Forwarding Strategy for Mobile Wireless Networks: Resource Usage vs. Latency

Victoria Manfredi[1*], Alicia P. Wolfe[1*], Xiaolan Zhang[2] and Bing Wang[3]

[1*]Department of Mathematics and Computer Science, Wesleyan University, 265 Church Street, Middletown, 06459, Connecticut, USA.

[2]Department of Computer and Information Sciences, Fordham University, 441 E. Fordham Road, Bronx, 10458, New York, USA.

[3]Computer Science & Engineering Department, University of Connecticut, 371 Fairfield Way, Storrs, 06269, Connecticut, USA.

*Corresponding author(s). E-mail(s): vumanfredi@wesleyan.edu; pwolfe@wesleyan.edu;
Contributing authors: xzhang@fordham.edu; bing@uconn.edu;

Abstract

Mobile wireless networks present several challenges for any learning system, due to uncertain and variable device movement, a decentralized network architecture, and constraints on network resources. In this work, we use deep reinforcement learning (DRL) to learn a scalable and generalizable forwarding strategy for such networks. We make the following contributions: i) we use hierarchical RL to design DRL packet agents rather than device agents to capture the packet forwarding decisions that are made over time and improve training efficiency; ii) we use relational features to ensure generalizability of the learned forwarding strategy to a wide range of network dynamics and enable offline training; and iii) we incorporate both forwarding goals and network resource considerations into packet decision-making by designing a weighted reward function. Our results show that the forwarding strategy used by our DRL packet agent often achieves a similar delay per packet delivered as the oracle forwarding strategy and almost always outperforms all other strategies (including state-of-the-art strategies) in terms of delay, even on scenarios on which the DRL agent was not trained.

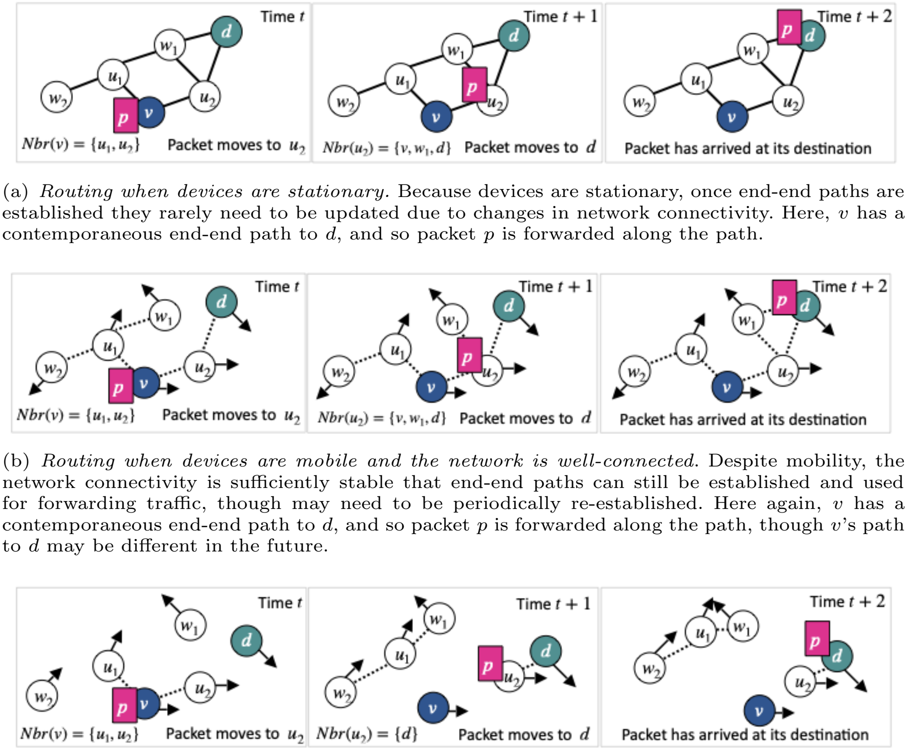
Keywords: Packet forwarding; Routing; Mobile wireless networks; Reinforcement learning; Neural networks

1 Introduction

Mobile wireless networks have been used for a wide range of real-world applications, from vehicular safety [1–3] to animal tracking [4, 5] to environment monitoring [6, 7] to search-and-rescue [8–10] to military deployments [11, 12] to the mobile Internet of Things [13]. These networks, however, present several challenges for learning systems. Devices are moving due to their association with, for instance, vehicles, robots, animals, or people, which causes changes in the network connectivity. As a consequence, devices are often only able to communicate with each other during limited windows of time when devices are within transmission range of each other. Furthermore, it may be difficult to predict when opportunities for communication may occur as these depend on the dynamics of device movement. Depending on the specific network problem to address, there may also be competing goals to trade-off, such as minimizing packet delivery delay vs. device resource usage. Finally, the network architecture is decentralized, complicating sharing of network state (and thus training of learning systems) as exchange of information is limited by the communication opportunities available in the mobile wireless network.

In this work, we focus on the problem of packet forwarding in a mobile wireless network. Traditionally, forwarding strategies are hand-crafted to target specific kinds of network connectivity. In static networks, see Fig. 1(a), once end-end paths are discovered, these paths are generally stable as the network connectivity does not change. In comparison, while end-end paths may occasionally exist in some mobile networks due to dense connectivity or slow device movement, see Fig. 1(b), these paths typically only exist for short periods of time and are periodically re-established using ad hoc routing algorithms [14–17]. In other mobile networks, devices meet only occasionally due to sparse connectivity or fast device movement, see Fig. 1(c), and so contemporaneous end-end paths rarely exist. Consequently, delay tolerant network (DTN) forwarding algorithms [18–20] are used to select the best next hop for a packet using criteria such as expected delay to meet the packet's destination.

In many real mobile wireless networks, however, a mix of connectivity is often found, see [21], motivating the need for an adaptive forwarding strategy. Forwarding strategies that do explicitly adapt to disparate network conditions often focus on switching between two different strategies, such as between ad hoc routing and flooding [22, 23], or between ad hoc routing and delay tolerant forwarding [24–27]. Strategy switching, however, risks instability and



(a) *Routing when devices are stationary.* Because devices are stationary, once end-end paths are established they rarely need to be updated due to changes in network connectivity. Here, v has a contemporaneous end-end path to d, and so packet p is forwarded along the path.

(b) *Routing when devices are mobile and the network is well-connected.* Despite mobility, the network connectivity is sufficiently stable that end-end paths can still be established and used for forwarding traffic, though may need to be periodically re-established. Here again, v has a contemporaneous end-end path to d, and so packet p is forwarded along the path, though v's path to d may be different in the future.

(c) *DTN forwarding when devices are mobile and the network is poorly connected.* Now contemporaneous end-end paths rarely exist. Only temporal paths exist from v to d, so devices independently choose packet p's next hop and p is forwarded hop-by-hop to d.

Fig. 1 Comparing packet forwarding in stationary vs. mobile wireless networks (see (a) vs. (b) vs. (c)). In all scenarios, packet p travels from device v to device u_2 to finally arrive at its destination device d, but whether routing or DTN forwarding is used depends on whether a contemporaneous path is present.

poor convergence if conditions change quickly or network state is only partially observable.

To address the above challenges, we use deep reinforcement learning (DRL) [28] to design an adaptive packet forwarding strategy, using deep neural networks (DNNs) [29] to approximate the RL policy. By choosing next hops locally at devices, both contemporaneous and temporal paths can be implicitly constructed by the DRL agent. Importantly, devices located in different parts of a mobile network can each independently run the same DRL agent as the agent will react appropriately to the local state at each device by using the parts of its forwarding policy relevant for that state. Consequently, a single forwarding policy can be applied to a state space that includes both well-connected and poorly connected parts of a mobile network.

Most works on DRL-based forwarding in wireless networks focus on stationary devices; those that do consider device mobility either target specific kinds of network connectivity or have other limitations (see Section 2). In this work, we specifically focus on designing a forwarding strategy for mobile wireless networks able to adapt to very different kinds of network connectivity. We make the following contributions.

i) *Packet-centric decision-making to simplify learning (Section 3.2).* We use *packet agents*, making packets, rather than devices, the decision-making agent, to accurately assign credit to those actions that affect packet success. Because packets can wait for several time steps before making a decision, we use hierarchical RL [30–32] with *fixed policy options* [31] to more efficiently back up from one decision point to another.

ii) *Relational features to ensure generalizability despite device mobility (Section 3.3).* We use *relational features* to represent the states and actions used by the DRL agent. Relational features model the relationship between network devices instead of describing a specific device, and so support generalizability to scenarios on which the DRL agent was not trained. This ability to generalize allows us to *train offline* thereby avoiding the need for decentralized communication. Relational features also allow us to structure the DNN representing the DRL forwarding policy to consider *one action at a time*, producing a single Q-value per state and action pair. The number of times the DNN is used to predict Q-values then corresponds to the number of actions available, allowing the trained DRL agent to handle varying numbers of neighbors (see Fig. 1) and so be network independent.

iii) *A weighted reward function to trade-off competing goals (Section 3.4).* To incorporate packet forwarding goals and network resource considerations into packet decision-making, we design a *weighted reward function* for the DRL agent. Using our reward function, more weight can be placed on higher priority forwarding considerations. For instance, in a resource-constrained network setting, minimizing the number of forwards may be of a higher priority than minimizing packet delivery delay, and so should be weighted more by the reward function.

iv) *Extensive evaluation (Section 4).* We evaluate our approach, which combines the above key design decisions, on two widely-used mobility models with varying numbers of devices, transmission ranges, and parameter settings, spanning the continuum from disconnected to well-connected networks. Our results show that our DRL agent trained on only one of these scenarios generalizes well to the other scenarios including those for a different mobility model. Specifically, our DRL agent often achieves delay similar to an oracle strategy and almost always outperforms all other strategies in terms of delay, including the state-of-the-art seek-and-focus strategy [18], even on scenarios on which the DRL agent was not trained. While the oracle strategy requires global knowledge of future device movement, our DRL agent uses only locally obtained feature information.

The rest of this paper is organized as follows. In Section 2, we overview related work on DRL-based forwarding. In Section 3, we describe how we formulate a DRL model able to learn an adaptive forwarding strategy. In Section 4, we present simulation results using our DRL model. Finally, in Section 5, we summarize our main results and provide directions for future research.

2 Related Work

In this section, we overview related work on routing and forwarding strategies for mobile wireless networks. We divide these strategies into two classes: *traditional* approaches not based on learning, and *RL-based* approaches which use RL to make decisions. We are primarily interested in single-copy forwarding strategies, as our focus is on strategies that are able to operate in both sparsely and densely connected networks. In contrast, multi-copy strategies, in which multiple copies of the same packet may be made to reduce packet delivery delay, typically consider only sparse network scenarios as the risk of introducing congestion from making packet copies is lower than in dense networks.

2.1 Traditional Approaches

Traditional approaches to routing and forwarding are generally designed for specific kinds of network connectivity. For instance, ad hoc routing strategies like DSR [14], AODV [15], OLSR [16] and DSDV [17] target relatively well-connected mobile networks (see Fig. 1(b)). In comparison, for delay tolerant networks [33], aka opportunistic networks, the network topology is typically so sparse that no contemporaneous path exists between a source and destination (see Fig. 1(c)), and so epidemic flooding [20] as well as more resource-efficient strategies like seek-and-focus and utility-based forwarding [18] have been developed. In our simulation results in Section 4, we compare our approach with the seek-and-focus and utility-based forwarding strategies of [18] as these are state-of-the-art single-copy forwarding strategies for delay tolerant networks. We also compare with an oracle strategy based on epidemic flooding that performs optimally regardless of network connectivity but is not practicably implemented.

In many real mobile wireless networks, however, a mix of connectivity is often found, see [21], motivating the need for an adaptive forwarding strategy. Forwarding strategies that do explicitly adapt to disparate network conditions have been less explored and often focus on switching between two different strategies, such as between ad hoc routing and flooding [22, 23], or between ad hoc routing and delay tolerant forwarding [24–27]. Strategy switching, however, can lead to instability and poor convergence if network conditions change quickly or network state is only partially observable.

A number of works additionally focus on designing multi-copy forwarding strategies, for instance [19, 34, 35], to improve forwarding when the network topology is sparse. Multi-copy strategies generate multiple copies of the same packet to increase the probability that at least one copy of the packet is delivered within a certain amount of time. While creating packet copies can reduce the delivery delay for a given packet, doing so also increases the traffic load on the network and hence congestion. Consequently, multi-copy strategies typically focus on sparse network scenarios in which devices only occasionally have neighbors and there is little traffic, reducing the risk of congestion from packet copies. Because our focus is on designing forwarding strategies that seamlessly adapt between sparse and dense network connectivity with possibly significant amounts of traffic, we focus in this work on designing a single-copy forwarding strategy (i.e., one that does not make packet copies).

2.2 DRL-Based Approaches

DRL-based approaches to routing and forwarding in mobile and wireless networks have the advantage of making it easy to take into consideration different network features and optimization goals that may be difficult for humans to reason about. DNNs specifically provide a natural way to unify different approaches to modeling mobility for forwarding decisions: 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 DRL agent learns the relationship between mobile network features and how best to forward traffic, with the learned policy represented using a DNN.

Early works [36–41] focus on distributed routing but use less scalable and less generalizable table look-up based approaches and typically learn online. When the network topology is changing, online learning can be problematic as devices may have few or no neighbors and there may be limited bandwidth to exchange any information necessary for training. Consequently, many of the recent works on DRL-based forwarding strategies focus on stationary networks where devices do not move, see [42–52].

Fewer works consider forwarding in mobile wireless networks, and those that do often optimize for specific kinds of network connectivity, such as focusing primarily on vehicular networks [53–56] or UAV networks [57–61]. Works that consider mobile networks more broadly have limitations: [62] extends early work on strategies that learn online [36, 38] to tactical network environments but uses RL to estimate the shortest path to the destination and focuses on multi-copy forwarding, i.e., making additional copies of packets to reduce delay, unlike the single-copy forwarding strategy we design in this work; [63] focuses on sparse network scenarios, specifically delay tolerant networks; [64] focuses on forwarding messages to network communities rather than individual devices in delay tolerant networks; [65, 66] focus on relatively limited network scenarios with a few fixed flows and up to 50 devices.

Differences from prior work. The goal of our work is to design an adaptive DRL-based forwarding strategy that can span the continuum from sparsely connected to well-connected mobile networks. In Section 4, we show that our learned forwarding strategy trained on a mobile wireless network with 25 devices can generalize during testing to mobile networks with 100 devices and varying transmission ranges. While our work builds off of ideas in [51], we consider the much harder network setting of mobile devices rather than the stationary devices considered in [51], and we design novel features to capture temporal and spatial network connectivity as well as propose a reward function to reflect competing network goals.

Our use of offline centralized training of the DRL agent (but online distributed operation) avoids online training's need for significant information exchange between devices that may be separated by many hops (e.g., information such as whether a packet was delivered or dropped). When deployed in a network, each device independently uses its own copy of the trained DRL agent to make distributed routing decisions.

An important takeaway of our work is the need to tailor existing machine learning techniques to handle the decentralized communication and resource constraints of mobile wireless networks. For instance, while we have some features that represent actual relationships between devices as in relational learning [67–69], we primarily use relational techniques to treat devices as interchangeable objects described by their attributes rather than their identities, to build a single model that works across all devices. While our aggregated neighborhood features are similar in spirit to graph neural networks [70–72], our features are simpler to compute and can more easily handle changing neighbors. While we use DRL to learn a policy, we handle actions differently than in a DQN [73] since the number of actions available at a device changes over time and varies across devices. Finally, while our weighted reward function could also be converted to use multi-objective reward techniques [74, 75], which find policies for a range of reward factor weightings, doing so would make training the DRL agent more computationally expensive.

3 Learning an Adaptive Forwarding Strategy

In this section, we overview our approach to using DRL to learn an adaptive forwarding strategy. We first give a brief overview of RL and DRL background material (Section 3.1). We then describe how we formulate the problem of forwarding packets as a learning problem (Section 3.2), the relational features we use to model a mobile wireless network (Section 3.3), and how we construct the DRL agent's states and actions from the relational features and derive a reward function for packet forwarding (Section 3.4). We finish by describing how our DRL agent makes forwarding decisions (Section 3.5), and how offline training of the DRL agent is performed (Section 3.6).

3.1 Background

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. The environment is modeled using a Markov decision process (MDP). An MDP comprises a set of states (S), a per state set of actions

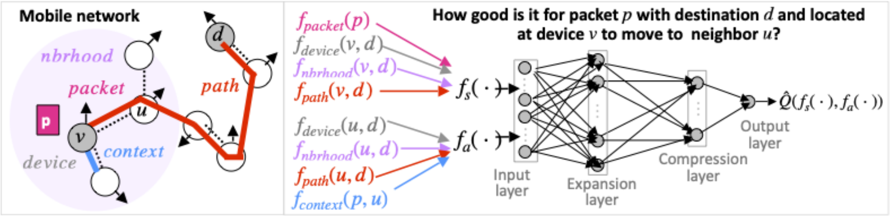


Fig. 2 Overview of how our DRL packet agent makes decisions. Packet p at device v makes a forwarding decision by activating the DNN at v once for each action a available in p's current state s. The DNN inputs are the features $f_s(\cdot)$ and $f_a(\cdot)$, which describe the state from p's point of view and the action under consideration. Packet p chooses the action with the best estimated Q-value.

($\mathcal{A}(s)$), a reward function, and 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 $a \in \mathcal{A}(s)$. 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 RL algorithm learns a Q-value for each (s, a) pair. A Q-value estimates the expected future reward for an RL 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.

When the MDP has a small number of states and actions, an RL agent can learn a Q-value function using Q-learning (see [76]). When the state space is too large for exact computation of the Q-values, function approximation can be used to find approximate Q-values. Here, we use DNNs for function approximation, see Fig. 2, as in Deep RL (DRL). Each state s and action a are translated into a set of features via the functions $f_s(\cdot)$ and $f_a(\cdot)$, respectively. These features are then used as input to the DNN, to produce as output an approximate Q-function $\hat{Q}(f_s(\cdot), f_a(\cdot))$.

3.2 Problem Formulation

In a mobile wireless network (and in networks generally), it is natural to think of devices as the DRL agents choosing next hops for packets, but doing so confuses the decision-making dependencies that are involved in forwarding a packet. That is, the steps from a packet's source device to its destination device (or drop) are constructed by the behavior of all devices through which the packet passes, rather than only by the device at which the packet is currently located. Thus, we use *packet agents*, making packets, rather than devices, the decision-making agent. Doing so allows us to more accurately assign credit to the actions that affect whether a packet is successfully delivered. With packet agents, the agent state can track what happens to the packet over time, as it moves from one device to the next. Thus, the sequence of states that led to the packet being delivered or dropped is known directly. Devices now only choose which packets should have the opportunity to make a decision, but do not choose the next hop for a packet. If we were to instead use device agents, the

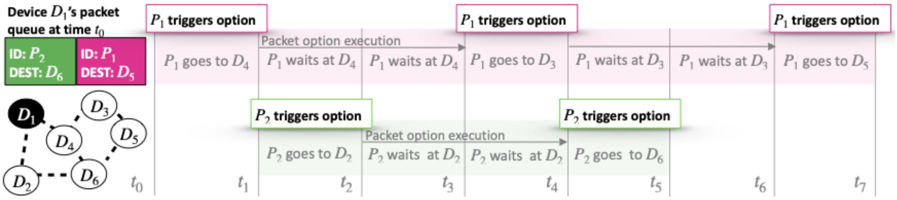


Fig. 3 Example of how our fixed policy options operate over time. Here, we show options operating concurrently for packets P_1 and P_2.

agent state would be a function of the packets seen at each device over time, making knowledge of what happens to packets after they leave the device not readily available.

Our use of packet agents does, however, give rise to a kind of multi-agent problem. Even though each packet agent greedily optimizes its own travel time, packets still indirectly interact with each other via device queues. However, we do not use a global cooperative reward function to coordinate packet agents. Instead, we have each device enforce fairness among the packets in its own queue, choosing which packet gets to choose a next hop. In this work, devices allow the first k packets to make a decision, where k is generally sufficiently large that all packets are forwarded. Alternative queuing disciplines include allowing only the packet at the front of the queue to make a decision or selecting a subset of packets based on the Q-value for each packet's best action combined with a fairness metric quantifying the number of decisions made by packets in the same flow (i.e., going from the same source to the same destination).

Importantly, much of a packet's life is spent waiting in a queue. In most networks, queuing delay is the largest contributor to total packet delivery delay. A packet must wait in a queue for its turn to make a forwarding decision at a device. Once a packet chooses a next hop, the packet is transmitted, but then goes back to waiting in the queue at that next hop. A packet has no actions it can take while it waits: there are no decisions for the packet to make but time still passes. This scenario is a natural case for *hierarchical RL* [30–32], in which an RL agent organizes its extended-time actions, or *options* [31] in a hierarchical time structure, using a semi-MDP rather than an MDP to model the environment.

We specifically use *fixed policy* options [31], where for the duration of an option, there is only a single action available to be selected on each time step, and it is always the same action. We make a slight modification (see Section 3.4), so that the first action of the option can vary (corresponding to the packet's next hop action selection), and the remaining actions are fixed with no decisions to be made (corresponding to the packet waiting in the next hop's queue). The initial next hop action plus the subsequent time interval that the packet may need to wait is then treated as a single option. How long (and whether) a packet must wait dictates how long it takes for the associated option to complete. If the next hop is the destination, for instance, the packet will be immediately delivered, with no waiting. Fig. 3 illustrates how such options operate over time. Packet P_1 triggers an option by choosing to move from device D_1 to device D_4. The option continues as P_1 waits at D_4 for two time steps, with no action selection possible. Once P_1 gets an opportunity to choose a next hop at D_4, the previous option terminates and a new option is triggered.

Compared to primitive actions, options generally require less data to be collected and allow smaller state spaces. Consequently, Q-learning proceeds more quickly. In our setting, if a packet spends τ time steps on average waiting at a device between decision points, doing backups at one-time step intervals during that waiting period would increase the size of our training data by a factor of τ, but without adding any useful information to the training data (as a packet has no decisions to make when waiting) and so would make learning slower. With options, reward signals are backed up over multiple time steps in a single update, which speeds up learning.

3.3 Mobile Network Features

Our goal is to be able to use the same learned forwarding strategy at different devices and in different mobile networks (e.g., dense or sparse networks with or without contemporaneous end-to-end paths) with unknown or unseen device mobility. To achieve this, we use *relational features* to represent the states and actions used by the DRL agent. Relational features, such as delay to destination, model the relationship between network devices instead of describing a specific device and so are not tied to a specific network topology.

We propose five classes of relational features specifically tailored to model mobile wireless networks. These classes give a general framework for organizing the features needed for forwarding packets in a mobile network. The features we propose for each class, however, are not exhaustive but rather only the specific features used by the DRL agents whose performance we evaluate in Section 4; we expect many other features could be proposed for each class. We next describe the features we use, from the point of view of a packet p when choosing its next hop. We assume that p's destination is device d and that p is currently located at device v which has neighbors $u \in Nbr(v)$.

1. *Packet features,* $f_{packet}(p)$, are a function of information about packet p. We propose the following packet features.
   (i) Packet p's *time-to-live (TTL).* TTL is a packet header field used in real networks to prevent the possibility of packets looping forever in the network. When a packet is generated at its source device, the packet's TTL field is set to some initial value. Then, the TTL field is decremented by one whenever a packet is forwarded to another device. A packet is dropped when its TTL reaches 0.
   (ii) Packet p's *time-at-device.* That is, how long p has been at its current device, v.

2. *Device features,* $f_{device}(v, d)$, are a function of device v's information and destination device d's ID. We propose the following device features computed at the current time step t.

   (i) Device v's *queue length.* 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 (or any packets in the same traffic flow as p) may be preferred to better distribute traffic and decrease delivery delay.

   (ii) Device v's *per-destination queue length* considering only packets for destination d. This captures the possibly differing amounts of congestion along paths to different destinations.

   (iii) Device v's *node degree.* This corresponds to the number of neighbors of device v, that is, the number of devices within wireless transmission range of v.

   (iv) Device v's *node density.* This is computed as the fraction of neighbors that v has out of the N devices in the network.

   Our DRL agent additionally keeps track of two device features, the x and y location coordinates of device v, which are not input into the DNN but are only used to compute the Euclidean distance, a path feature described later in this section. Even when a device knows its own location, locations for other devices can only be obtained when two devices meet and exchange features. Consequently, location (and thus distance) features may be out-of-date.

3. *Path features,* $f_{path}(v, d)$, describe the time-varying path from a device v to a destination device d. Unlike the other features discussed so far, path features can use not just current device information but also historical information. Consequently, these features may have some associated uncertainty. We propose the following path features.

   (i) *Last inter-meeting time.* This measures the amount of time that has elapsed since device v last met destination d.

   (ii) *Last meeting duration.* This measures the duration of the last meeting between device v and destination d.

   (iii) *Euclidean distance.* This measures the distance from device v to destination d. Euclidean distance is calculated using v's current x and y location coordinates, and device v's recorded (and possibly out-of-date) location coordinates of destination d (see description of device features).

   (iv) *Timer transitivity.* This is computed between device v and destination d, using the calculation proposed in [18] for utility-based and seek-and-focus forwarding and described next. Each device v maintains a timer for each other device d in the network, denoted as $\tau_v(d)$, which is the time elapsed since device v last met device d. We implemented the timer transitivity as defined in [18]: 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 $d_{u,v}$ (the distance between devices u and v), then $\tau_v(d)$ is set to $\tau_v(d) = \tau_u(d) + t(d_{u,v})$. When device locations are not known, the distance $d_{u,v}$ can be approximated using the transmission range: device v can only be so far away from device u for v to be within the transmission range of u. Timer transitivity captures the insight that, for many mobility models, a smaller timer value on average implies a smaller distance to a device, where the timer evaluates the "utility" of the device in delivering a packet to another device. In experiments we have done, we have observed correlation between timer values and distance. Consequently, timer transitivity can be used as a feature to approximate distance when location coordinates, and correspondingly the Euclidean distance feature, are not available.

4. *Neighborhood features,* $f_{nbrhood}(v, d)$, are computed over the current neighbors $Nbr(v)$ of a device v. We propose the following neighborhood features: for each device and path feature, $f_i \in f_{device}(v, d) \cup f_{path}(v, d)$, we compute the minimum, maximum, and average over device v's current neighborhood, $Nbr(v)$. This is similar in spirit to the aggregation function in a graph neural network [70–72]. These features compress the information obtained from a variable number of neighbors into a fixed size vector to input to the DNN in Fig. 2.

5. *Context features,* $f_{context}(p, u)$, provide context for other features. For a packet p at a device v considering a next hop $u \in Nbr(v) \cup v$, we propose context features that indicate whether p has recently visited u or not. Each packet p stores in its packet header the last $N_{history}$ device IDs that it visited, where $N_{history}$ is a predetermined constant. Let $\mathcal{H}(p, i)$ be the ID of the device that packet p visited i hops ago, for $0 \leq i < N_{history}$. When $i = 0$, then $\mathcal{H}(p, 0)$ is the device at which p is currently located. To make the context features relational, rather than use device IDs, we use a sequence of Boolean features, b_i, defined as:

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

The use of packet history reduces unnecessary packet transmissions. 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.

Basic features. We designate a subset of the above features as *basic features* (marked in Table 1), which we use in all but one of the trained DRL agents that we evaluate (see Table 5). Specifically, we designate packet TTL, queue length, per-destination queue length, node degree, and node density as basic features. We derive our basic features from information used by traditional routing and forwarding algorithms. The *queue length* feature measures the amount of congestion in the network. The *per-destination queue length* feature measures the amount of congestion on the path to a particular destination and comes from the backpressure forwarding algorithm (see [77]), which forwards a packet p to the next hop that has the largest difference in queue length relative to p's current device when considering only packets with the same destination as p. Backpressure is throughput optimal (i.e., maximizes the number of packets able to be delivered) when the network topology is relatively fixed and there is sufficient traffic such that queues have a chance to build up. The *packet TTL* feature is used in traditional routing and forwarding algorithms to avoid routing loops: if a packet has been forwarded more than a set number of times (typically related to the diameter of the network), it is assumed that the packet is undeliverable and the packet is dropped. The *node degree* and *node density* features help with generalizing the learned forwarding strategy to networks with different numbers of devices, varying transmission ranges, and heterogeneous mobility.

Additional features. We designate the remaining features of time-at-device, Euclidean distance, timer transitivity, last inter-meeting time, and last meeting duration as *additional features*, whose utility we evaluate through an ablation study in Section 4.4.1. We label Euclidean distance as an additional feature since the x and y location coordinates may not always be available. In such situations, the timer transitivity and other timing related features (i.e., packet time-at-device, last inter-meeting time, and last meeting duration) could be used instead.

Feature estimation. All features are estimated using local exchange of information between neighboring devices. A device discovers its neighbors 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 $u \in Nbr(v)$. Device v obtains the following information from each neighbor u: i) the features $f_{device}(u, d) \cup f_{path}(u, d)$, 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 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.

Feature normalization. Our goal when normalizing features is to re-scale them into the range of approximately 0 to 1. Mobility makes normalization challenging as the ranges of the raw feature values, such as for node degree, may be very different in different mobile networks. To address this, we make the

Table 1 For each feature f_i, normalization is done using $(f_i + 1)/(D + 1)$, where 1 is added to f_i to avoid zero values for features and the value D is given in the table. Neighborhood features are not normalized as they are a function of other normalized features; context features also are not normalized as they are Boolean valued. We also distinguish the *basic features* used by all but one of the DRL agents whose performance we evaluate in our simulations from the *additional features*, see Table 5.

Class	Feature	D	Type
Packet	Packet TTL	300	Basic
Packet	Packet time-at-device	200	Additional
Device	Queue length	20	Basic
Device	Per-destination queue length	20	Basic
Device	Node degree	10	Basic
Device	Node density	N	Basic
Device	x-coordinate location	500m	Additional
Device	y-coordinate location	500m	Additional
Path	Euclidean distance	2	Additional
Path	Timer transitivity	800	Additional
Path	Last inter-meeting time	800	Additional
Path	Last meeting duration	100	Additional

normalization a function of network properties when possible, such as the (approximate) number of devices in the network or the (approximate) size of the area in which devices are moving or the maximum expected inter-meeting time between pairs of devices. In this way, the normalization can better adapt to new network environments. Table 1 summarizes how we normalize features in our simulations.

During training vs. testing vs. real network deployment it can be useful to scale some features slightly differently. For instance, during training, a packet's time-to-live (TTL) should be sufficiently long to still allow packets to be delivered to the destination despite some random exploration, but also be short enough to allow packet drops due to expired TTLs. However, during testing, a large TTL should be used to eliminate packets drops due to expired TTLs, so as to prevent skewing of the results as the delays incurred by dropped packets cannot be usefully counted. Consequently, to obtain our testing results, we use a larger initial TTL than during training (see Table 3), but we re-scale a packet's raw TTL before normalizing to ensure that as long as the raw TTL is within the TTL range used in training then the TTL feature will have the same value as during training, and one otherwise. Conversely, in a real network deployment, a small TTL would be used to prevent undeliverable packets from looping too long in the network.

3.4 MDP Formulation

Let $Nbr(v)$ be the current neighbors of device v. Each individual packet agent p currently located at device v can choose between moving to one of v's neighbors or staying at device v. Packet p's actions therefore correspond to the set $Nbr(v) \cup \{v\}$. We next define the states, actions, and reward function for our packet-centric DRL agent from p's point of view.

- *States.* The state features that packet p uses to make a decision are a function of features derived from packet p, p's destination d, and p's current device v: $f_s(v, p, d) = f_{packet}(p) \cup f_{device}(v, d) \cup f_{path}(v, d) \cup f_{nbrhood}(v, d)$.
- *Actions.* The action features for each action u in packet p's action set $Nbr(v) \cup \{v\}$ are defined by $f_a(u, p, d) = f_{device}(u, d) \cup f_{path}(u, d) \cup f_{context}(p, u)$. This action description reuses many of the same features as in the state description, but is defined in terms of a device $u \in Nbr(v) \cup \{v\}$ rather than just packet p's current location at device v, and additionally includes the context features.

Reward function. Forwarding strategies for mobile wireless networks must trade-off competing goals for packet delivery, such as minimizing delivery delay while also minimizing resource usage like energy and link bandwidth. As a packet travels to its destination, it must also assign credit or blame to the devices it passes through, depending on the success or failure of the packet to reach its destination. To incorporate these forwarding goals and network resource considerations into packet decision-making, we design a *weighted reward function.* Using our reward function, more weight can be placed on higher priority forwarding considerations, such as to not waste resources by making unnecessary packet transmissions.

We define separate rewards for the action of a packet choosing to stay at its current device, r_{stay}, vs. moving to a neighboring device that is not the destination, $r_{transmit}$, as packet transmission requires expending energy. The ratio of r_{stay} to $r_{transmit}$ determines the trade-off that the forwarding strategy learns between minimizing packet delivery delay vs. number of transmissions. For instance, by setting $r_{stay} = r_{transmit}$, the DRL agent would minimize delay, but ignore the number of transmissions made. For mobile networks, where forwarding loops and unnecessarily long paths can easily arise, explicitly penalizing transmissions is important for learning a more efficient forwarding strategy. We define two other rewards, for actions that lead to a packet being delivered to its destination, $r_{delivery}$, or dropped, $r_{drop} = r_{transmit}/(1 - \gamma)$, where $\gamma \in [0, 1]$ is the RL discount factor. The drop reward is equivalent to receiving a reward of $r_{transmit}$ for infinite time steps. Our reward settings are given in Table 3.

Actions vs. options. 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 [31]):

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

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

Options for packets begin and end at transmission, i.e., when a packet leaves a device. Options can consist of a packet being i) *delivered*, ii) *dropped*, or iii) *transmitted and then staying* for some number of time steps. There are two types of options in this domain: terminal (packet delivery or drop) and non-terminal (transitions from one device to another and then staying at the new device). On packet delivery or drop, the option takes only a single time step and the next state s_{t_j} is the terminal state. The sample of return for delivery is $y = r_{delivery}$, and similarly for the drop option, $y = r_{drop}$.

Because transmit and then stay options have rewards that are constant for every time step over the life of the option after the first time step, all $r_k = r_{stay}$ in the option except at the first time step t_i, where $r_i = r_{transmit}$. Therefore, on transitions where the packet does a transmission action and then stays for several time steps, the sample of return is:

$$y = r_{transmit} + R_{stay}(t_j - (t_i + 1)) + \gamma^{(t_j - t_i)} \cdot \max_{a' \in \mathcal{A}(s_{t_j})} Q(s_{t_j}, a').$$

where $R_{stay}(t_j - (t_i + 1))$ is the return from the stay actions taken over the course of the option, from time $t_i + 1$ to time t_j, and is defined by

$$R_{stay}(t_j - (t_i + 1)) = r_{stay} \cdot \frac{1 - \gamma^{(t_j - (t_i + 1))}}{1 - \gamma}.$$

Because reward is constant for all but the first time step, only the beginning and end of an option need be stored during training. 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 training 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 Sections 3.3 and 3.4, we use "actions" rather than "options" to refer to extended-time actions.

3.5 Decision-Making

The DNN architecture we use to approximate the Q-value function for the DRL agent is shown in Fig. 2. Our DNN has four layers: input, expansion, compression, and output. Let $F = |f_s(\cdot)| + |f_a(\cdot)|$ be the number of input features and thus the size of the input layer. The expansion layer has $10F$ neurons and the compression layer has $F/2$ neurons. Our DNN settings are summarized in Table 2. Note that we are not currently using regularization or dropout during training, but plan to explore their use in future work. The DNN outputs a Q-value for each state, action pair, represented by the feature vectors $f_s(\cdot)$ and $f_a(\cdot)$. Each device has a copy of the same trained DNN that encodes the forwarding strategy, which allows decision-making to be done independently at each device.

Fig. 2 also overviews how our DRL packet agent uses the trained DNN to make a forwarding decision. To obtain the features $f_s(\cdot)$ and $f_a(\cdot)$ to input

Table 2 DNN settings used in our simulations.

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

into the DNN in Fig. 2, the DRL agent computes the following features for packet p with destination d currently at device v: $f_{packet}(p)$, $f_{device}(v, d)$, and $f_{path}(v, d)$ using local information at device v; $f_{nbrhood}(v, d)$, $f_{device}(u, d)$, and $f_{path}(u, d)$ using the features received by v from $u \in Nbr(v)$; and finally, $f_{context}(p, u)$ using only u's ID in addition to the information carried in packet p's header fields. The number of times the DNN is used to predict Q-values corresponds to the number of actions available to the packet. Using the DNN to separately make predictions for each action, rather than making predictions for all actions at once, allows the DRL agent to handle varying numbers of actions, and, correspondingly, varying numbers of neighbors (and therefore varying topologies). During training, ϵ-greedy action selection is used, with ϵ set as in Table 3.

3.6 Offline Training

In a mobile wireless networks, devices can only exchange information (such as whether a particular packet reached its destination or was dropped) using distributed communication. Consequently, it is typically not feasible to gather the information needed for training online due to limited network bandwidth. Instead, we use *offline training*, leveraging a network simulator to both generate the training data and train the DNN model.

During the training phase, we alternate between periods of collecting data, using the current DNN model to make routing decisions, with periods of training a new DNN model, using all of the data collected so far. This alternation allows the trained DNN to improve over time. Initially, the DNN parameters are random, and so the routing data collected from our network simulator is generated by a random forwarding strategy. But as the DNN parameters improve from training, so too does the forwarding strategy, and correspondingly, so does the training data. The periods of data collection are termed rounds and we set each round to be $T_{round} = 1000$ time steps long, see Table 3. In our training simulations in Section 4 we collect $T_{train} = 90,000$ time steps of data, i.e., 90 rounds of training data.

Table 3 Simulation parameters.

Symbol	*Meaning*	*Value*
N_{train}	# of devices during training	25
N_{test}	# of devices during testing	25, 64, 100
X_{train}	Transmission range for training	50m
X_{test}	Transmission range for testing	20m to 80m
-	RL max iterations	100
ϵ_{train}	RL exploration rate for training	0.1
ϵ_{test}	RL exploration rate for testing	0
γ	RL discount factor	0.99
TTL_{train}	TTL field initialization during training	300
TTL_{test}	TTL field initialization during testing	3000
B	Maximum queue size	200
$r_{delivery}$	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 time steps for training	90,000
T_{test}	# of time steps for testing	100,000
T_{model}	# of training time steps used by testing model	60,000
$T_{cooldown}$	# of time steps at simulation end with no traffic	10,000
T_{round}	# of time steps per round	1000

The training data collected using our network simulator includes the following information, recorded for every packet decision made: i) the state features, ii) the action features for each action considered for the state, iii) the action that was selected, and iv) the reward that was received. We append this information to the end of a single data file shared by all devices during training. From this data file, we construct the state, action, reward, next state tuples, (s, a, r, s'), that are needed for training the DNN approximating the DRL forwarding strategy. Specifically, we use the currently trained DNN and the (s, a, r, s') tuples to obtain the target Q-values associated with the tuples. Then a new DNN is trained with this data.

Because our relational features are device and network independent, we are able to train a single DRL agent via experience replay, taking trajectory samples from the data file. Samples are only taken for packets that have been dropped or delivered (and hence the full trajectory is contained in the data file) to allow the computation of each option return which is a function of the time it takes for a packet to reach a terminal state.

Once training has been completed, no more updating of the DNN model occurs. Consequently, the forwarding strategy that is used during testing does not change. In Section 4, for testing, we use the trained model produced after $T_{model} = 60,000$ time steps, see Table 3, as this has converged for both delivery delay and number of forwards. The trained model that has been chosen is then copied to each device and used *independently* at each device by packets to choose next hops.

4 Simulation Results

In this section, we overview our simulation setup and describe our simulation results. Our goal is to evaluate the performance and generalization capabilities of our formulation of DRL-based forwarding in mobile wireless networks. We first describe how we simulate a mobile wireless network (Section 4.1). Then, we describe the different forwarding strategies against whose performance we compare (Section 4.2). Next, we overview DRL training performance (Section 4.3). Finally, we overview DRL testing performance as well as the performance of the other forwarding strategies (Section 4.4).

4.1 Methodology

Our simulations are done using a custom discrete-time packet-level network simulator that we have implemented in Python3. The DRL agents are trained and all strategies are tested using this simulator. We use Keras v.2.5.0 [78] and Tensorflow v.2.11 [79] to implement the DNN. Table 3 gives our simulation parameters.

4.1.1 Device Mobility

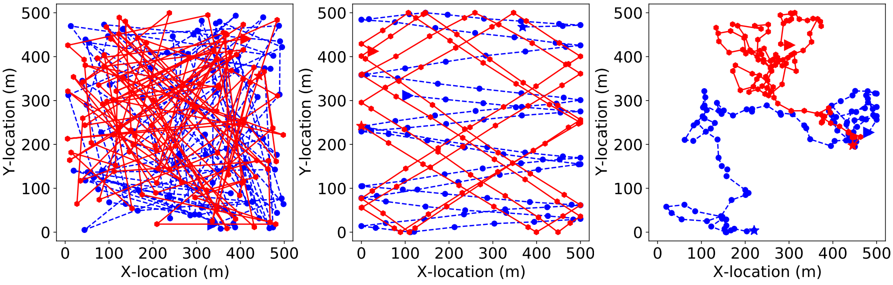
We use two widely-used mobility models in our simulations: the steady-state *random waypoint (RWP)* mobility model [80–82] and the original *Gauss-Markov (GM)* mobility model [83, 84].

RWP mobility. In the RWP mobility model, a device picks a random location and speed, and travels to the chosen location at the chosen speed. Subsequently, the device pauses for a random duration, and repeats the above process until the end of the simulation. In our simulations, our *RWP* scenario uses the following parameter settings. We set the average speed at which devices move to be 3 m/s with a speed delta of 2 m/s with no pause time. This means that each device moves to the chosen location at a speed chosen uniformly at random from the range $[3-2, 3+2]$, i.e., $[1, 5]$. Figure 4(a) gives example device movement for this RWP scenario.

GM mobility. In the GM mobility model, a device is assigned an initial velocity with which to move, which is then updated at fixed intervals, i.e., at time $t = \delta, 2\delta, ..., (n-1)\delta, n\delta, ...$, where δ is the update interval, indicating how long a device moves at a given velocity before updating. In the implementation of the GM mobility model that we use, the velocity of a device during the n-th interval $[(n-1)\delta, n\delta]$, denoted as (v_n^x, v_n^y), is calculated as follows [83, 85]:

$$\begin{aligned} v_n^x &= \alpha v_{n-1}^x + (1-\alpha)\mu\cos(\theta_{n-1}) + \sigma\sqrt{1-\alpha^2}w_{n-1}^x \\ v_n^y &= \alpha v_{n-1}^y + (1-\alpha)\mu\sin(\theta_{n-1}) + \sigma\sqrt{1-\alpha^2}w_{n-1}^y \end{aligned}$$

where $0 \leq \alpha \leq 1$ is a memory level parameter that controls how much a device's velocity in the n-th interval (v_n^x, v_n^y) depends on its velocity in the



(a) RWP mobility (b) GM mobility (c) GM-curly mobility

Fig. 4 Example device movements over time for the RWP, GM, and GM-curly mobility scenarios. For clarity, only the movements for two devices (device 2 in blue with dashed lines and device 18 in red with solid lines) are shown. We show the first 100 BonnMotion waypoints for these two devices, where dots represent the waypoints, the triangle symbol indicates the starting waypoint of a device, and the star symbol indicates the ending waypoint of a device.

previous interval (v^x_{n-1}, v^y_{n-1}), u is the asymptotic mean of the speed, σ is the asymptotic standard deviation of the velocity (where the asymptotic standard deviations of the x and y components of the velocity are set to be the same value, σ), θ_{n-1} is uniformly randomly distributed in $[0, 2\pi)$, and w^x_{n-1} and w^y_{n-1} are uncorrelated random variables from a Gaussian distribution with zero mean and unit standard deviation. Smaller values of α result in more randomness of device movement over time, with $\alpha = 0$ giving rise to the memory-less Random Walk model. Larger values of α result in stronger memory, with $\alpha = 1$ giving rise to mobility with a constant velocity. The GM mobility model gives rise to a slightly different spatial distribution of devices in the network, compared with the RWP mobility model. Specifically, in the RWP mobility model, devices are more concentrated near the center of the simulation region [80]; whereas in the GM mobility model, devices are evenly distributed in the simulation region.

In our simulations, we consider two scenarios for the GM mobility model, termed *GM* and *GM-curly* respectively. For the GM scenario, we set $\delta = 30$s, $\mu = 3.0$ m/s, $\sigma = 0.1$ m/s, and $\alpha = 0.6$, which gives the relatively straight-lined movement shown in Fig. 4(b). For GM-curly scenario, we set $\delta = 2$s, $\mu = 1.0$ m/s, $\sigma = 10$ m/s, and $\alpha = 0.3$, which gives the "curlier" movement shown in Fig. 4(c). Note that the GM-curly model is only used for testing: no DRL agents are trained on this model.

Mobility trace generation. In our simulations, we use BonnMotion [85] to generate mobility traces for devices moving under the above models. Specifically, for all models, BonnMotion generates a mobility trace as a sequence of waypoints for each device. For example, suppose that device v's mobility trace is (t^v_0, x^v_0, y^v_0), $(t^v_1, x^v_1, y^v_1), \ldots, (t^v_n, x^v_n, y^v_n)$, where $t^v_0, t^v_1, \ldots, t^v_n$ are the points in time when device v changes its velocity, and $(x^v_i,\ y^v_i)$ are device v's locations at time t^v_i. During the interval $[t^v_i, t^v_{i+1}]$, device v moves from location (x^v_i, y^v_i)

to location (x^v_{i+1}, y^v_{i+1}) at a constant speed. Since the time points at which the waypoint changes are decided by the mobility model, they do not necessarily coincide with the per-second time steps in our simulation. We therefore calculate the locations of each device v at each time step t by first finding the time interval $[t^v_k, t^v_{k+1}]$ in which t lies in, and then calculating the location at time t via a linear interpolation of v's locations at t^v_k and t^v_{k+1}.

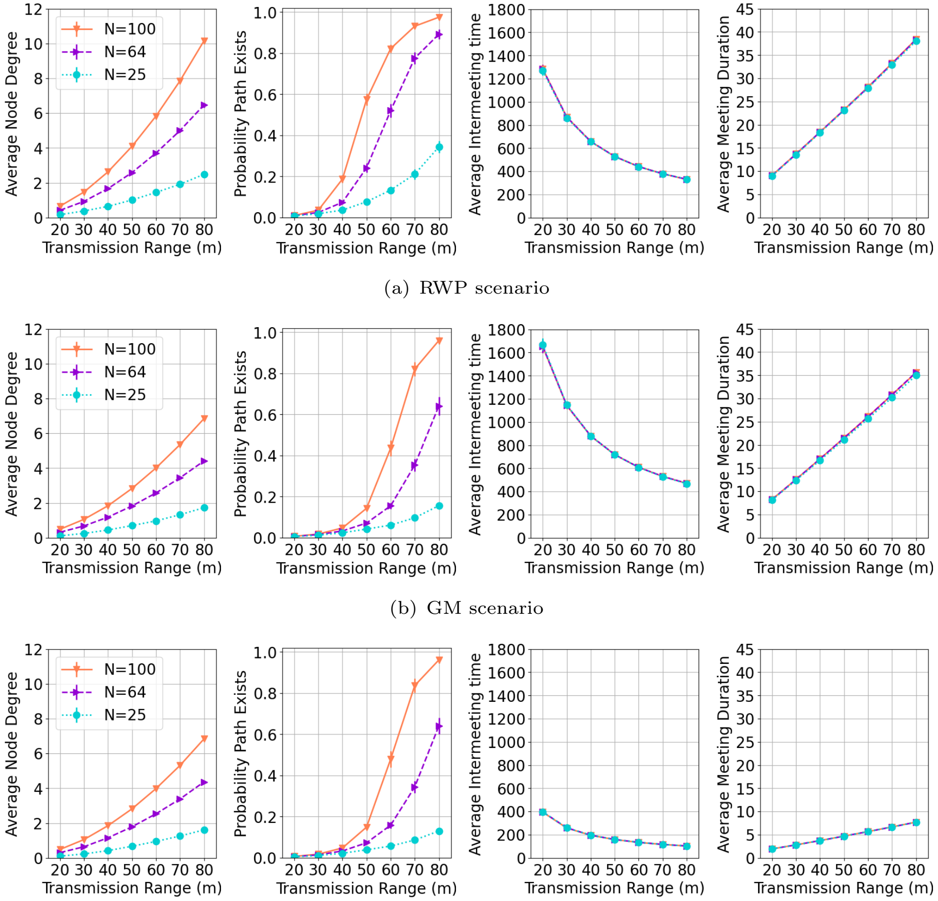
In our simulations, as listed in Table 3, all training scenarios use $N_{train} = 25$ devices moving in a 500m × 500m area and a transmission range of $X_{train} = 50$m. For testing, we consider $N = 25, 64$, and 100 devices moving in a 500m × 500m area and vary the transmission range X_{test} from 20m to 80m to obtain both poorly connected and well connected mobile scenarios. We generate separate mobility traces for the training and testing scenarios.

Connectivity of mobile scenarios. For the RWP scenario, Fig. 5(a) plots the average node degree, the probability that a path exists, the average intermeeting time, and the average meeting duration for various numbers of devices N and transmission ranges. The last two metrics are computed online between each possible pair of devices, and so are independent of the number of devices. As expected, the network connectivity increases as the number of devices, N, and transmission range increase, leading to higher average node degree, higher probability of having a path between a pair of devices, shorter intermeeting time, and longer meeting duration. The corresponding quantities for the GM and GM-curly scenarios are shown in Fig. 5(b) and (c). We observe lower average node degree and lower probability of a path in the GM and GM-curly scenarios compared to the RWP scenario. This is consistent with the observation that for the same number of devices and transmission range, devices are more concentrated near the center of the simulation region under the RWP model than under the GM model [80]. We also observe that the GM-curly scenario has more frequent meetings between devices (shorter intermeeting times) but also shorter meeting durations than does the GM scenario.

4.1.2 Network Traffic

We vary the amount of traffic over time by modeling flow arrivals, 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 (and thus the amount of congestion) as a function 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 run starts with $\lambda_F \lambda_D$ initial flows. Each device has a queue with a maximum size of $B = 200$ packets, beyond which additional packets are dropped.

Because flow arrivals are a function of the number of devices in the network, not just network connectivity but also traffic congestion are varied in the different network scenarios used for testing. Thus, when evaluating generalization of a DRL agent trained on one scenario to other scenarios, we are



(a) RWP scenario

(b) GM scenario

(c) GM-curly scenario

Fig. 5 The connectivity properties of the various network scenarios considered in our simulations.

evaluating not just generalization due to different connectivity and numbers of devices but also due to different amounts of traffic.

For the packet TTL field, we use different initialization values in training vs. testing. During training, a packet's TTL field is initialized to $TTL_{train} = 300$. We use this value since if TTL_{train} is too small, the DRL agent may not be able to deliver any packets due to the initial random forwarding, while if it is too large, the DRL agent may not be able to experience dropped packets caused by expired TTLs. During testing, a packet's TTL field is initialized to a much larger value, $TTL_{test} = 3000$. This is because the statistics we compute about delay and number of forwards are only for delivered packets: any dropped packets due to expired TTLs would skew these statistics. The different TTL_{train} and TTL_{test} values, however, do necessitate the re-scaling described in the last paragraph of Section 3.3. When deploying in a real network, however, it is likely desirable to use smaller TTL values than what we use, to ensure that undeliverable packets are dropped in a timely manner.

At each time step, every device in the network is given an opportunity to transmit up to $k = 200$ packets in its queue. Given our network settings, this k is sufficient for a device to transmit all of its packets, avoiding the need for device decision-making. But if the number of packets in a device's queue were larger than k, the best k packets could be forwarded where best is a function of the Q-value for each packet's best action combined with a fairness measure to ensure each packet regularly gets a transmission opportunity.

4.2 Forwarding Strategies

In our simulations, we compare the performance of five forwarding strategies, including a delay minimizing strategy (*oracle*) and a transmission minimizing strategy (*direct transmission*) to give bounds on the performance of the DRL agent. We also compare with the utility and seek-and-focus strategies from [18] as state-of-the-art strategies that trade-off delay with number of transmissions. Our goal is to understand which forwarding strategies have low packet delivery delay while also making a good trade-off in terms of resources used due to packet transmissions.

1. *Oracle forwarding* uses complete information about current and future network connectivity to calculate the minimal hop path that achieves the minimal delivery delay for each packet. To find these forwarding paths, we make use of epidemic routing [20]. Epidemic routing creates many copies for a packet and distributes them to the network. For those packet copies that reach the destination with the minimum latency, we further find the packet copy that reached the destination with the minimum number of hops. Although the oracle forwarding strategy minimizes delay while maintaining a good trade-off in terms of network resources, it is not practical to implement in real networks since it requires knowing the current and future network topology.
2. *Direct transmission forwarding* only forwards a packet one hop, directly from the source to the destination. This is optimal when the goal is to minimize the number of transmissions per packet.
3. *Utility-based forwarding* [18] maintains a timer at each device v for each device d in the network, denoted as $\tau_v(d)$, which is the time elapsed since device v last met device d. We implemented the timer transitivity as defined in [18]: 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 $d_{u,v}$ (the distance between devices u and v), then $\tau_v(d)$ is set to $\tau_v(d) = \tau_u(d) + t(d_{u,v})$. 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

Table 4 Parameter settings for utility-based and seek-and-focus forwarding strategies.

Symbol	*Meaning*	*Utility*	*Seek&Focus*
U_{th}	Utility threshold	10	100
U_f	Focus threshold	-	20
prob	Random forwarding probability	-	0.5
time_until_decoupling	Time before sending packet back to device	-	10
T_{focus}	Max duration to stay in focus phase	-	10
T_{seek}	Max duration to stay in re-seek phase	-	50

more than the utility threshold, U_{th}, the packet is forwarded to device u; otherwise, the packet is not forwarded. We optimize U_{th} for $N_{train} = 25$ and $X_{train} = 50$m, which are the same settings on which the DRL agent used in testing was trained; the value of U_{th} that we use is shown in Table 4.

4. *Seek-and-focus forwarding* [18] combines the utility-based strategy (*focus phase*) with random forwarding (*seek phase*). 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 is in the focus phase, and the carrier of the packet performs utility-based forwarding with utility threshold U_{th}. In addition to U_f, *prob*, and U_{th}, seek-and-focus 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). We optimize these parameters for $N_{train} = 25$ and $X_{train} = 50$m, which are the same settings on which the DRL agent used in testing was trained; the parameter values we use are shown in Table 4.
5. *DRL forwarding* uses a DRL agent to make forwarding decisions. We consider a number of different DRL agents trained on different scenarios, shown in Table 5. Specifically, we prefix these agents by the mobility scenario used in training, i.e., "RWP" or "GM", followed by the features used: e.g., basic features only (marked as "basic"), or basic and additional features (marked as "dist" for those that use Euclidean distance as an additional feature, or marked as "timer" for those that use the timer transitivity feature as an additional feature, or "timer$^+$" for those that use the timer transitivity feature and other timing features, or "timer$^-$" for those that use the timer transitivity feature but not all of the basic features). By default, we use $N_{history} = 5$ for context history; a scenario that does not use the context history feature is marked as "nohist". Finally, a scenario that does not use the default value of $r_{transmit} = -2$ appends a

Table 5 DRL training scenarios. All training scenarios use $N_{train} = 25$ devices and a transmission range of $X_{train} = 50$m. Scenarios are prefixed by the type of the mobility scenario used in training, either RWP or GM. All but the RWP-timer$^-$ strategy use all of the *basic* features specified in Table 1. The additional features used beyond the basic features are also listed for each scenario. We use $T_{train} = 60,000$ training time steps for the trained DRL model based on looking at the training plots in Fig. 6.

Scenario	$N_{history}$	$r_{transmit}$	Features
RWP-basic	5	-2	Basic
RWP-timer$^-$	5	-2	Timer, Queue Len., Per-Destination Queue Len, TTL
RWP-timer	5	-2	Basic, Timer
RWP-timer$^+$	5	-2	Basic, Timer, Intermeeting, Duration, Time-at-device
RWP-dist	5	-2	Basic, Euclidean distance
RWP-dist-1	5	-1	Basic, Euclidean distance
RWP-dist-10	5	-10	Basic, Euclidean distance
RWP-dist-nohist	0	-2	Basic, Euclidean distance
RWP-dist-1-nohist	0	-1	Basic, Euclidean distance
RWP-dist-10-nohist	0	-10	Basic, Euclidean distance
GM-dist	5	-2	Basic, Euclidean distance
GM-dist-1	5	-1	Basic, Euclidean distance
GM-dist-10	5	-10	Basic, Euclidean distance

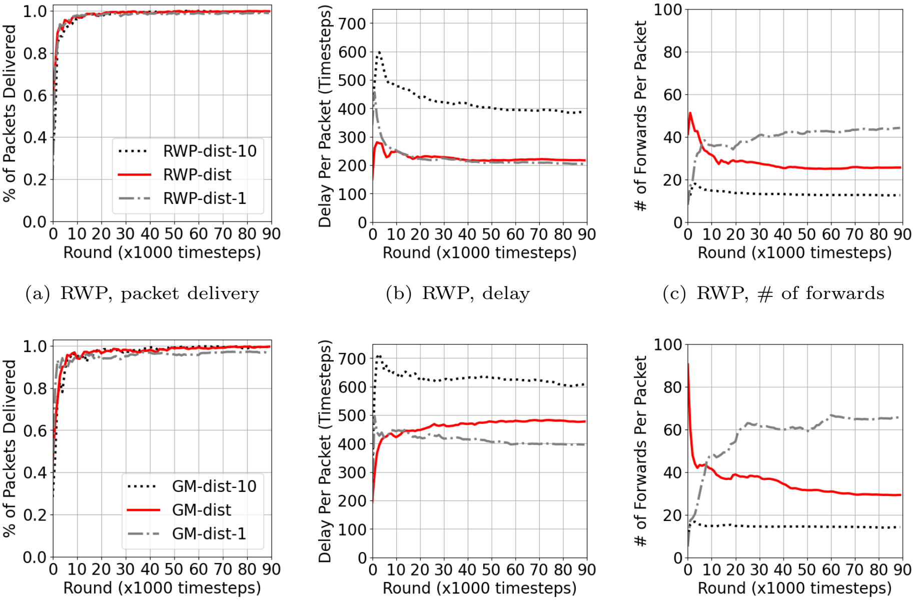
"-1" or "-10" to "dist", representing the other two $r_{transmit}$ values of -1 and -10 that we explore. During training, the DRL agent is essentially learning about different kinds of network connectivity over time, as well as different amounts of congestion over time (and the resulting variability in queue length). In Fig. 6, we show DRL training performance, discussed further in the next section.

In our current implementation of the timer transitivity calculation, which is used by the utility-based and seek-and-focus forwarding strategies as well as by the DRL agent when using the timer feature, we assume the distance $d_{u,v}$ between two neighboring devices u and v is known. In practice, we can use the transmission range as an over-estimate of $d_{u,v}$. This is because timer transitivity is calculated only when the two devices are within transmission range of each other, i.e., their distance from each other is no more than the transmission range. We expect that using the transmission range as an approximation of the distance, $d_{u,v}$, will have only a negligible impact on the timer value. The timer transitivity calculation itself is only a rough estimate since the true relationship between delay and distance in a given network depends on the actual movement of devices.

4.3 Training Performance

Fig. 6 overviews DRL training performance. Each training simulation is run for T_{train} time steps, where each time step corresponds to one second. At each time step, the cumulative performance over all packets delivered up to that time step is shown. The performance metrics we use are the packet delivery rate, delay per packet delivered, and number of forwards per packet delivered.

The top row of Fig. 6 shows DRL training performance on the RWP scenario, for three DRL agents, using the default amount of history (i.e., $N_{history} = 5$). In addition, one of the DRL agents uses the default transmission reward of $r_{transmit} = -2$, while the other two DRL agents use $r_{transmit} = -1$



(a) RWP, packet delivery (b) RWP, delay (c) RWP, # of forwards

(d) GM, packet delivery (e) GM, delay (f) GM, # of forwards

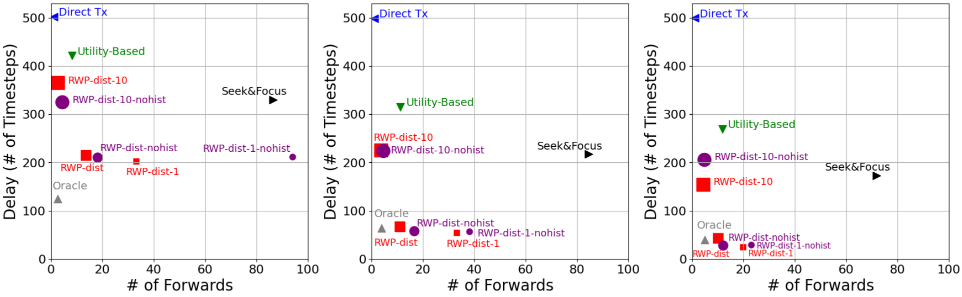
Fig. 6 DRL agent training performance. All training runs are done with $N_{train} = 25$ and X_{train}=50m.

and −10, as marked in the legend. The bottom row of Fig. 6 shows shows DRL training performance on the GM scenario for the same feature and reward settings. Fig. 6 shows convergence of the learned DRL strategies, with $r_{transmit}$ controlling the learned trade-offs between delay and number of forwards. For a given reward setting, the DRL agents trained on the GM scenario have a higher delay per packet delivered and a higher number of forwards per packet delivered than do the DRL agents trained on the RWP scenario, which is consistent with the observation that GM mobility leads to lower connectivity than does RWP mobility for the same setting (see Fig. 5). We explore this resource-delay trade-off in our testing results in the next section.

For clarity, Fig. 6 only presents the training performance for a subset of the DRL agents shown in Table 5. The training performance for the DRL agents not shown is similar to those in Fig. 6 and we observed convergence for all of the DRL agents that we use.

4.4 Testing Performance

The focus of our testing simulations is to evaluate the performance and generalization capabilities of our proposed DRL approach. All performance measures are computed over the delivered packets, such as the delay or number of forwards per packet delivered. All performance metrics are averaged across all simulation runs for a given set of parameters, except for the maximum queue



(a) $N_{test} = 25$, $X_{test} = 50\text{m}$ (b) $N_{test} = 64$, $X_{test} = 50\text{m}$ (c) $N_{test} = 100$, $X_{test} = 50\text{m}$

Fig. 7 Resource usage vs. latency on the RWP scenario. Each point is the average of 50 simulation runs. All DRL agents were trained with $N_{train} = 25$ devices and $X_{train} = 50\text{m}$ and all packets were delivered and no packets were dropped. Labels indicate the forwarding strategy.

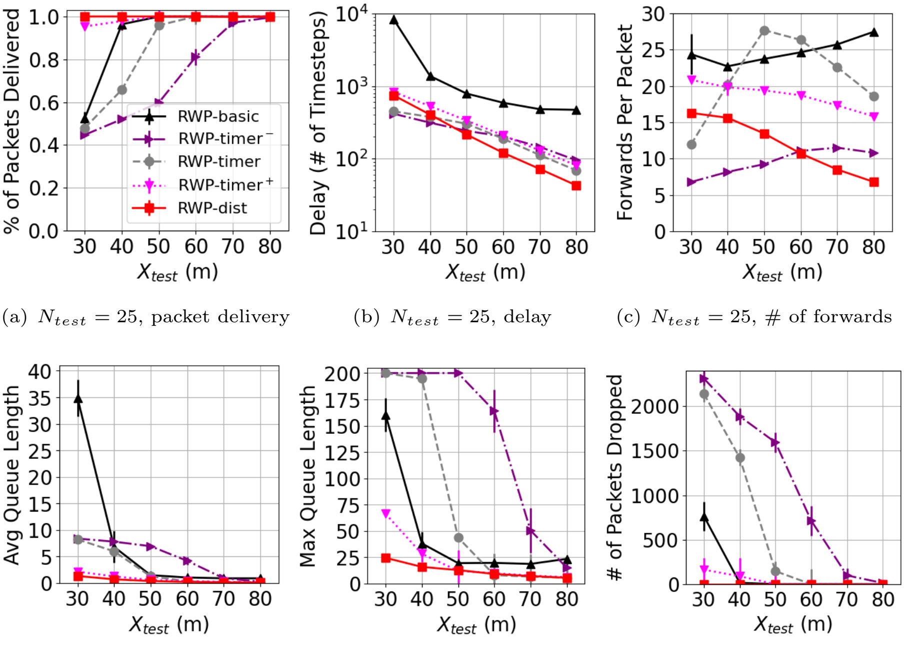
length which is the average of the maximums seen in each simulation run for a given set of parameters.

4.4.1 Impact of Parameter and Feature Choice

In this section, we investigate how the choice of training transmission reward, $r_{transmit}$, affects testing performance. We also perform feature ablation to understand how different features impact performance.

Impact of transmission reward. Fig. 7 plots the trade-off between the delay (i.e., latency) and the number of packet transmissions (i.e., resource usage) for various forwarding strategies. For the DRL strategies, we consider multiple strategies that differ in their choice of $r_{transmit}$ and whether the context history feature is being used. For comparison, we also show the trade-offs made by the transmission minimizing direct transmission strategy ("Direct Tx"), the delay minimizing oracle strategy, and the state-of-the-art utility-based and seek-and-focus strategies. In the following, we consider the impact of the choice of $r_{transmit}$ on the delay vs. resource usage trade-off; the impact of the history feature is deferred to later. The performance comparison of the DRL strategies with the other strategies is discussed in detail in Section 4.4.2.

Recall that the reward for staying at a device, r_{stay}, is fixed to -1. When $r_{transmit} = -1$, there is no penalty for transmission as a packet receives the same reward for transmitting as for staying at a node (i.e., $r_{transmit} = r_{stay}$). Correspondingly, we see that for the DRL strategies using this reward setting, RWP-dist-1 and RWP-dist-1-nohist, delay per packet delivered is the lowest among the various DRL strategies, but the number of forwards per packet is the highest. Conversely, for $r_{transmit} = -10$, corresponding to RWP-dist-10 and RWP-dist-10-nohist, there is a significant penalty for making a transmission, and so, while the delay per packet delivered is the highest for these DRL agents, the number of forwards per packet is now the lowest. Henceforth, the DRL agents used are trained with $r_{transmit} = -2$ as this setting provides a



(a) $N_{test} = 25$, packet delivery (b) $N_{test} = 25$, delay (c) $N_{test} = 25$, # of forwards

(d) $N_{test} = 25$ (e) $N_{test} = 25$ (f) $N_{test} = 25$, # of drops

Fig. 8 Testing performance for the RWP scenario, varying the features used by the DRL agents and the transmission range X_{test} from 30m to 80m. Each point is the average of 50 simulation runs; 95% confidence intervals are shown. All DRL agents were trained with $N_{train} = 25$ devices and $X_{train} = 50$m but with varying features, see Table 5. For the RWP-basic, RWP-timer$^-$, RWP-timer, and RWP-timer$^+$ strategies, not all packets were able to be delivered by the end of each simulation.

good trade-off between delay and number of transmissions. In future work, it would be interesting to evaluate how varying the other rewards, r_{stay}, r_{drop}, and $r_{delivery}$, impacts forwarding performance.

Impact of context history features. As shown in Fig. 7(a), context history can also serve as a deterrent to unnecessary packet transmissions: the DRL agents that do not use context history (i.e., RWP-dist-nohist, RWP-dist-1-nohist, and RWP-dist-10-nohist) have a higher number of forwards, compared to the the DRL agents with $N_{history} = 5$ (i.e., RWP-dist, RWP-dist-1, and RWP-dist-10), for the same setting of $r_{transmit}$. This gap is most pronounced when there is no transmission reward penalty (i.e., when $r_{transmit} = -1$) and the network is sparse (i.e., for $N_{test} = 25$). 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.

Impact of distance and timer features. Fig. 8 looks at the impact of different features on performance. The RWP-basic strategy uses the fewest features, i.e., only the basic features listed in Table 5. Fig. 8(b) shows that the RWP-basic strategy has larger average delay per packet delivered than any of the other strategies. The RWP-timer strategy uses the timer transitivity feature in addition to the features used by the RWP-basic strategy. Fig. 8 shows that adding the timer transitivity feature significantly reduces the delay compared to the RWP-basic strategy, but with increased forwards per packet delivered, and, more problematically, with significant numbers of packets dropped due to queues being full when the network topology is sparse (i.e., when $X_{test} < 60$m in Fig. 8(f)).

The RWP-timer^{+} strategy uses the time-at-device, last inter-meeting time, and last meeting duration features in addition to the timer transitivity and other features used by the RWP-timer strategy. Fig. 8 shows that these additional features give the RWP-timer^{+} strategy a delay similar to that of the RWP-timer strategy but with fewer forwards per packet and fewer packet drops. In results not shown, we observe that for the RWP-timer^{+} strategy, using the learned DRL model after 46,000 training time steps rather than 60,000 training results in no packet drops and stable queue lengths, though slightly larger delay but also significantly fewer forwards per packet delivered. Consequently, there may be some overfitting with the RWP-timer^{+} strategy due to the 60,000 time steps of training.

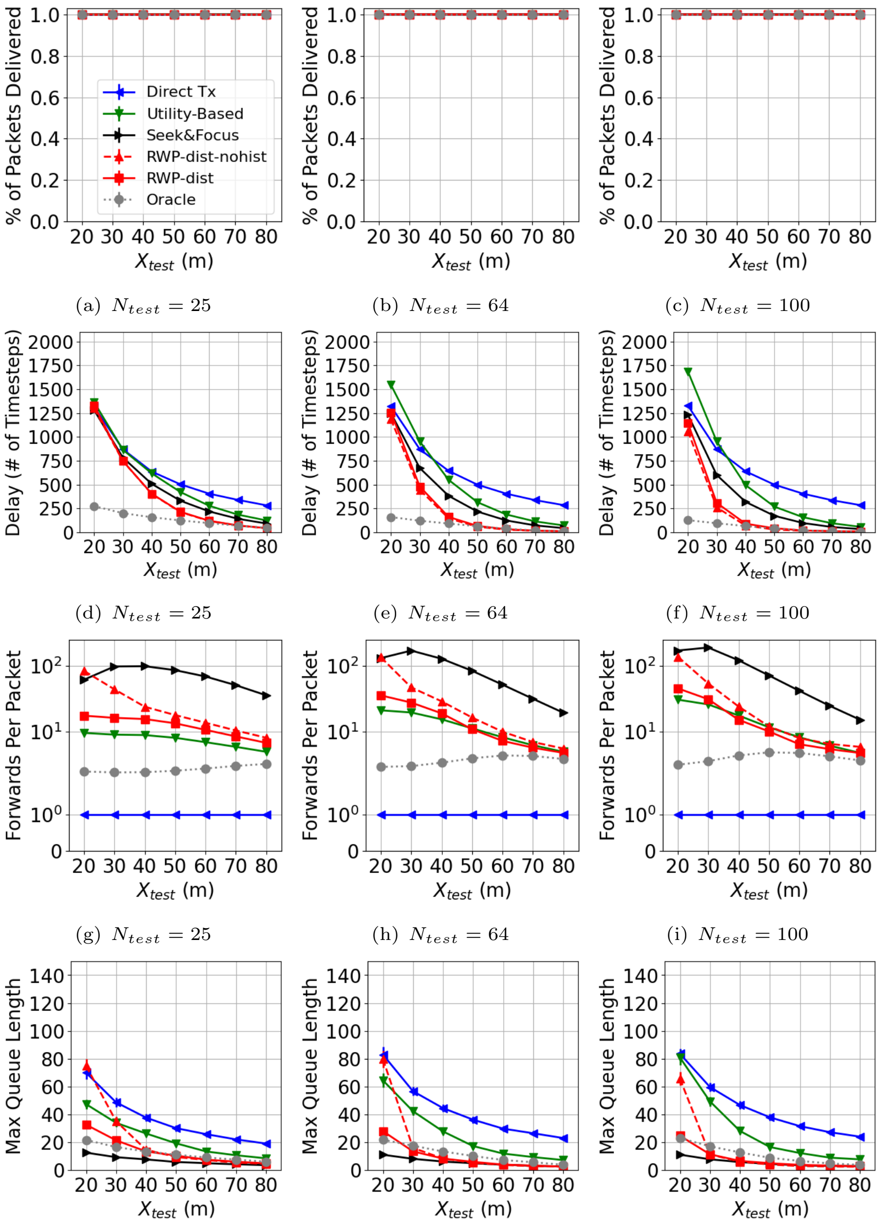
We also observe in Fig. 8 that the RWP-timer^{-} strategy, which only includes a subset of the basic features, and in particular excludes the node degree and node density features, performs significantly worse than the other strategies including the RWP-basic strategy in terms of packet drops and maximum queue length, indicating the utility of the node degree and node density features for generalization.

Fig. 8 shows that the RWP-dist strategy achieves the best performance for all of the performance metrics among all of the DRL strategies. Comparing the performance of the RWP-timer^{+} strategy with that of the RWP-dist strategy indicates that the timer transitivity feature can potentially be used as an approximation to the Euclidean distance feature. In the rest of our testing results in the next sections, we focus on the DRL agents that use the Euclidean distance feature in addition to the basic features.

4.4.2 Results for the RWP Scenario

In this section, we investigate how well the learned DRL forwarding strategies generalize during testing for the RWP scenario.

Overall results. In Fig. 9, we plot the the testing performance of two DRL agents, RWP-dist and RWP-dist-nohist, on the RWP scenario. The DRL agents were trained on the corresponding RWP scenarios described in Table 5, which use $N_{train} = 25$ devices and a transmission range of $X_{train} = 50$m. The



(a) $N_{test} = 25$ (b) $N_{test} = 64$ (c) $N_{test} = 100$

(d) $N_{test} = 25$ (e) $N_{test} = 64$ (f) $N_{test} = 100$

(g) $N_{test} = 25$ (h) $N_{test} = 64$ (i) $N_{test} = 100$

(j) $N_{test} = 25$ (k) $N_{test} = 64$ (l) $N_{test} = 100$

Fig. 9 Testing performance on the RWP scenario, varying the number of devices N_{test} from 25 to 100 and the transmission range X_{test} from 20m to 80m. Each point is the average of 50 simulation runs; 95% confidence intervals are shown. The legend shows DRL training conditions: both DRL agents were trained with $N_{train} = 25$ devices and $X_{train} = 50$m. All packets were delivered in these simulations and no packets were dropped.

testing scenarios shown in Fig. 9 include 25, 64 or 100 devices (the largest network size we investigated) with transmission ranges varying from 20m to 80m, leading to various connectivity levels (see Fig. 5). Most of the testing scenarios differ from the training scenario in terms of the number of devices and/or the transmission range, as well as the number of traffic flows. We see in Fig. 9 that our DRL agents are able to generalize well from their training scenarios to the various testing scenarios, including those on which they were not trained. We also observe that our DRL agents often achieve packet delivery delays similar to the oracle strategy, and outperform all other strategies in terms of delay.

All forwarding strategies must make some kind of trade-off between packet delivery delay and number of packet transmissions. While the DRL agents use a reward function to navigate this trade-off, seek-and-focus uses a less straightforward approach. Among the six parameters of seek-and-focus, we observed that varying the forwarding probability *prob* while fixing the other five parameters, see Table 4, 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.

Impact of network connectivity. Fig. 9 shows that as the network topology becomes more well connected (i.e., due to an increasing number of devices N_{test} or increasing transmission range X_{test}), the DRL agents start to have delay per packet delivered similar to that of the oracle strategy, with not too many more forwards per packet delivered. As the network topology becomes more sparse (i.e., due to decreasing N_{test} or decreasing transmission range X_{test}), Fig. 9 shows that the DRL agents start to have delay that is approaching the delay of the non-oracle strategies. The DRL agents have their highest delay per packet delivered for the $N = 25$ and $X_{test} = 20$m scenario which is the most disconnected testing scenario we consider (see Fig. 5). Due to the relatively few neighbors and long inter-meeting times between pairs of devices for this scenario, we hypothesize that using additional features, such as temporal neighborhood features, would improve DRL agent performance, as would considering predicted future neighbors when choosing next hop actions. We revisit these ideas again when we discuss the impact of network sparsity on forwarding performance more generally later in this section. Importantly, the DRL agents, trained on networks with $N_{train} = 25$ devices, are able to generalize their learned forwarding strategies to networks with different numbers of devices ($N_{test} = 64$ and $N_{test} = 100$), as well as to the varying numbers of actions available to packets in these networks.

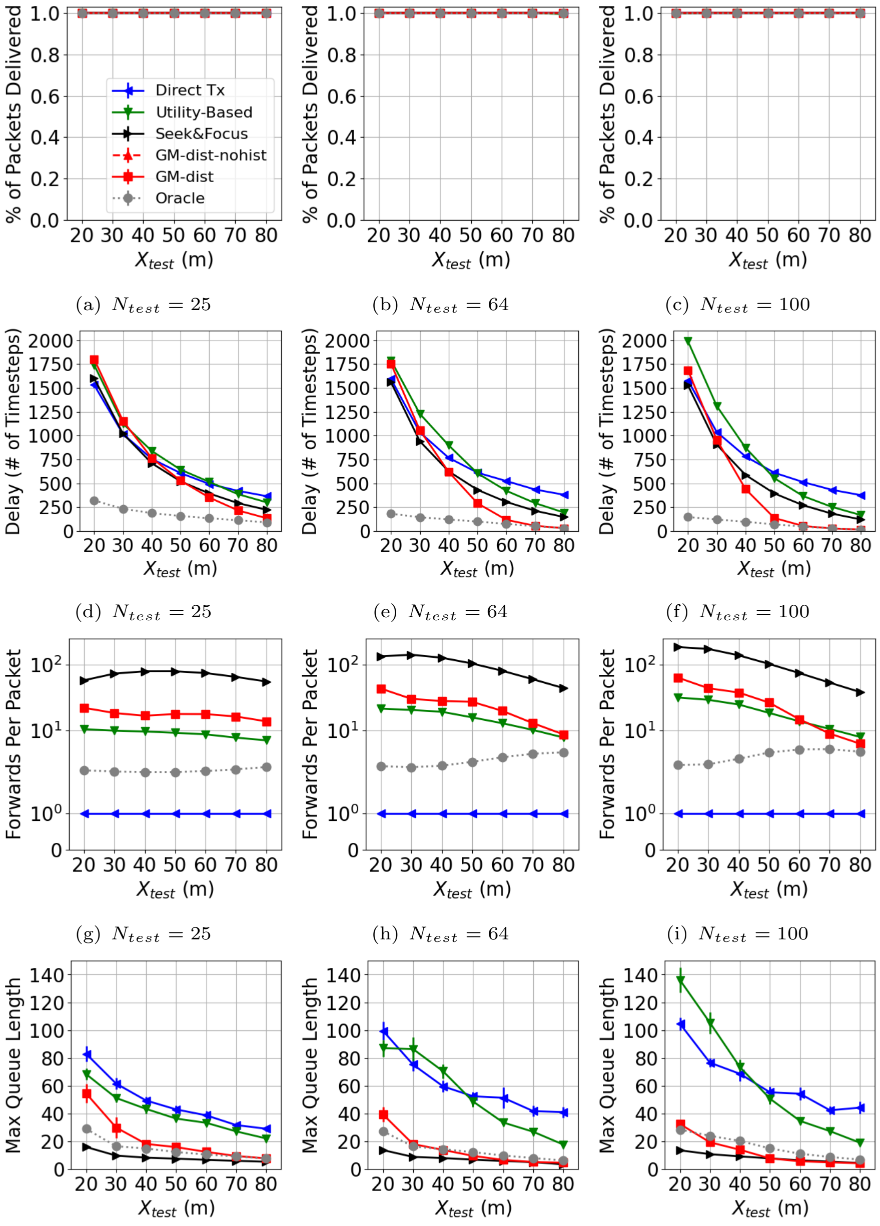
Impact of network traffic. The number of flows (and therefore amount of traffic) increases as a function of the number of devices N_{test} in the network (see Section 4.1.2). Network congestion, however, increases as the transmission range X_{test} decreases (due to correspondingly decreased network connectivity). Fig. 9 shows that the DRL strategies are able to generalize to network congestion levels different from those on which they were trained.

Impact of context history features. As we discussed earlier in Fig. 7, we found that the use of context history decreases the number of forwards per packet delivered. While the results we showed in Fig. 7 are for a fixed transmission range of $X_{test} = 50$m and varying numbers of devices N_{test}, Fig. 9 validates this for other transmission ranges (for X_{test} from 20m to 80m) as well as varying numbers of devices. Specifically, Fig. 9 shows that the DRL agent with no context history (i.e., RWP-dist-nohist) has a consistently higher number of forwards per packet delivered than the DRL agent with context history (i.e., RWP-dist) regardless of transmission range and despite having a similar delay per packet. This is an indication that the use of context history improves generalization of the learned forwarding strategy.

Handling network sparsity. As shown in Fig. 9, as the network becomes sparser (i.e., X_{test} decreases), the length of the oracle (i.e., optimal) path decreases. This is because in such scenarios, forwarding a packet to a neighbor is less likely to reduce packet delivery delay as that neighbor is unlikely to meet other devices, including the destination. In contrast, for the DRL strategy, as the network becomes sparser, path lengths increase, and delays deteriorate to be close to those of the non-oracle strategies. Compared to the DRL strategy, the more desirable behavior of the oracle strategy is a consequence of its ability to sample all possible paths to a destination in parallel via packet copies, enabling it to differentiate between dense and sparse networks and thereby make appropriate forwarding decisions. We hypothesize that adding additional features to the DRL agent that are able to detect sparsity more globally in the network would improve DRL forwarding behavior. Additionally, if the amount of network traffic is fixed, then as the network becomes sparser, traffic congestion and hence queue lengths increase. Therefore, training on network scenarios with more variable traffic to induce more queue length variation may also improve DRL forwarding behavior.

Queue stability. The stability of device queue lengths can be used to determine whether a network's capacity can support a given traffic load and whether the forwarding strategy is appropriately managing this traffic. Figs. 9 (j) to (l) show the maximum queue length seen in any of the simulation runs for a given setting. Though queue lengths increase as network connectivity decreases, we see that queue lengths are stable for all strategies, meaning that the forwarding strategy and network are able to support the traffic load. The direct transmission and utility-based strategies have noticeably larger queue lengths compared to the other strategies, with this gap increasing as the number of devices in the network increases.

While not shown, average queue lengths are small for all strategies, generally smaller than five packets. Note that the oracle 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 oracle strategy include all packet copies present but only until the first copy is



(a) $N_{test} = 25$

(b) $N_{test} = 64$

(c) $N_{test} = 100$

(d) $N_{test} = 25$

(e) $N_{test} = 64$

(f) $N_{test} = 100$

(g) $N_{test} = 25$

(h) $N_{test} = 64$

(i) $N_{test} = 100$

(j) $N_{test} = 25$

(k) $N_{test} = 64$

(l) $N_{test} = 100$

Fig. 10 Testing performance using GM mobility, varying the number of devices N_{test} from 25 to 100 and the transmission range X_{test} from 20m to 80m. Each point is the average of 50 simulation runs; 95% confidence intervals are shown. The legend shows DRL training conditions: the DRL agent was trained with $N_{train} = 25$ devices and $X_{train} = 50$m. All packets were delivered in these simulations and no packets were dropped.

delivered. Consequently, oracle queue lengths are not directly comparable to the queue lengths of other strategies.

4.4.3 Results for GM Mobility

In this section, we investigate how well the learned DRL forwarding strategies generalizes during testing for the GM scenario. As shown in Fig. 5, GM mobility gives rise to a significantly sparser network topology than does RWP mobility.

Overall results. In Fig. 10, we plot the the testing performance of one DRL agent, GM-dist, trained on the GM scenario. Like with our RWP results, most of the testing scenarios differ from the training scenario in terms of the number of devices and/or the transmission range, as well as the number of traffic flows. Fig. 10 shows that our DRL agent is able to generalize well from the training scenario to the various testing scenarios. The DRL agent often achieves packet delivery delays similar to the oracle strategy, and outperforms all other strategies in terms of delay except for the very sparsest network scenarios.

Impact of network connectivity. Like the RWP results in Fig. 9, the GM results in Fig. 10 show that as the network topology becomes more well connected, the DRL agent starts to have delay per packet delivered similar to that of the oracle strategy, with not too many more forwards per packet delivered. GM delay per packet delivered, however, is significantly higher than what is seen for the RWP results, due to the decreased connectivity of the GM mobility model.

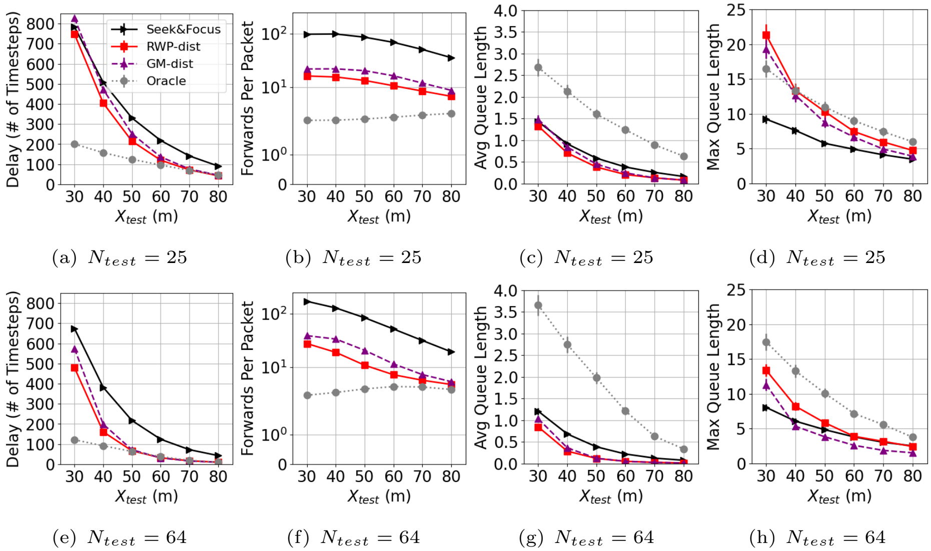
Impact of network traffic. Again, like the RWP results in Fig. 9, the GM results in Fig. 10 show that the DRL strategies are able to generalize to different traffic and congestion levels, despite the sparser connectivity of GM mobility.

Queue stability. While the queue lengths for the RWP results shown in Fig. 9 are relatively stable, the GM results in Figs. 10 (j) to (l) show higher queue lengths and, in particular, noisier queue lengths for the direct transmission strategy. Average queue lengths, again not shown, are still generally less than 5 packets.

4.4.4 Cross-mobility Model Generalization

In this section, we investigate how well the learned DRL forwarding strategies trained on one mobility scenario generalize to other mobility scenarios.

Training on GM and testing on RWP. In Fig. 11, we compare the testing performance of two DRL agents, GM-dist and RWP-dist, on the RWP scenario. While the RWP-dist strategy was trained on the RWP scenario, the GM-dist strategy was trained on the GM scenario. Our goal here is to confirm that our DRL-based approach to forwarding is able to generalize not just to different levels of connectivity and congestion for a given mobility model but also to different mobility models. Indeed, we observe in Fig. 11 that the GM-dist strategy



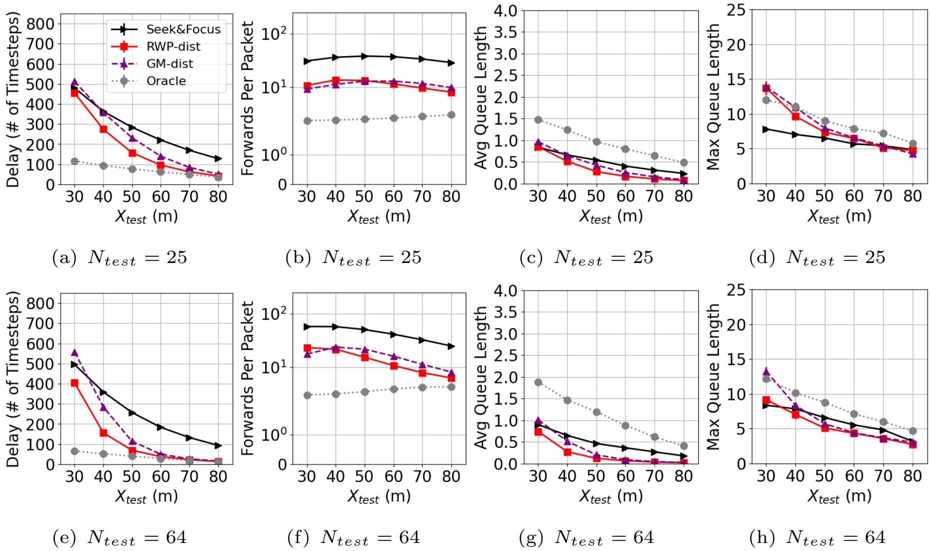
(a) $N_{test} = 25$ (b) $N_{test} = 25$ (c) $N_{test} = 25$ (d) $N_{test} = 25$

(e) $N_{test} = 64$ (f) $N_{test} = 64$ (g) $N_{test} = 64$ (h) $N_{test} = 64$

Fig. 11 Cross-mobility testing performance: we evaluate GM-dist performance, a DRL strategy trained on the GM scenario, on the RWP scenario. For comparison, the results when using the RWP-dist strategy, i.e., a DRL strategy trained on the RWP scenario (which matches the scenario that is used in testing) are also shown. Each point is the average of 50 simulation runs; 95% confidence intervals are shown. All packets were delivered in these simulations and no packets were dropped.

performs quite similarly to the RWP-dist strategy in terms of delay per packet delivered, with only slightly higher numbers of forwards. Interestingly, we also observe that the GM-dist strategy leads to slightly smaller maximum queue lengths compared to that of the RWP-dist strategy, potentially due to the GM-dist strategy training on the slightly sparser (and therefore more congested) GM mobility scenario.

Training on GM or RWP and testing on GM-curly. In Fig. 12, we compare the testing performance of two DRL agents, GM-dist and RWP-dist, on the GM-curly scenario. Fig. 12 shows that both DRL agents are able to generalize to this new scenario, despite not having been trained on it. Interestingly, the RWP-dist strategy performs better than the GM-dist strategy does on the GM-curly scenario: i.e., RWP-dist has lower delay and fewer forwards than does GM-dist. This may be a consequence of the higher randomness of movement found in the RWP training scenario compared to the GM training scenario (see Fig. 1), making it more similar to the GM-curly scenario despite being a different mobility model.



(a) $N_{test} = 25$ (b) $N_{test} = 25$ (c) $N_{test} = 25$ (d) $N_{test} = 25$

(e) $N_{test} = 64$ (f) $N_{test} = 64$ (g) $N_{test} = 64$ (h) $N_{test} = 64$

Fig. 12 Cross-mobility testing performance: we apply GM-dist, a DRL strategy trained on the GM scenario to testing on the GM-curly scenario Each point is the average of 50 simulation runs; 95% confidence intervals are shown. All packets were delivered in these simulations and no packets were dropped.

5 Conclusions and Future Work

In this work, we have shown that it is possible to use DRL to learn a scalable and generalizable forwarding strategy for mobile wireless networks. We leverage three key ideas: i) packet agents, ii) relational features, and iii) a weighted reward function. Our results show that our DRL agent generalizes well to scenarios on which it was not trained, often achieving delay similar to the oracle strategy and almost always outperforming all other strategies in terms of delay including the state-of-the-art seek-and-focus strategy [18]. The key ideas of our approach are generally applicable to other decision-making tasks in mobile wireless networks.

There are a number of research directions we would like to explore in future work. We expect that as the kinds of mobility that the DRL agent sees become more diverse, more features will also be needed to characterize the key differences in mobility and enable the DRL agent to generalize to a wide variety of mobile networks. Ultimately, we would like to evaluate how well our approach performs in real test-bed scenarios. Additionally, targeted sampling of the feature space during training would be helpful to ensure that a diversity of network connectivity and congestion is seen. In this work, we trade-off two conflicting goals by using different weights for the transmission and stay rewards. As future work, we will explore another approach to handle competing goals based on constrained RL (e.g., as in [86, 87]), where one goal is set as the objective function, and the other goal is set as a constraint.

We would also like to refine our reward function to incorporate additional network considerations such as fairness. Finally, we would like to explore device decision-making to complement our packet agents. For instances, devices could learn which subsets of packets should get to make a forwarding decision when a transmission opportunity arises.

Acknowledgements

The authors thank the reviewers for their helpful and insightful comments.

Declarations

- **Funding**. This material is based upon work supported by the National Science Foundation (NSF) under award #2154190 and award #2154191. Results presented in this paper were obtained in part using CloudBank, which is partially supported by the NSF under award #2154190. The authors also 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.
- **Conflicts of interest/Competing interests**. The authors have no competing interests to declare that are relevant to the content of this article.
- **Ethics approval**. Not applicable.
- **Consent to participate**. Not applicable.
- **Consent for publication**. Not applicable.
- **Availability of data and material**. Not currently available.
- **Code availability**. Not currently available. We are working on open-sourcing our code.
- **Authors' contributions**.
  - *Victoria Manfredi*: Conceptualization; Methodology; Formal analysis and investigation; Software; Writing - original draft preparation; Writing - review and editing; Funding acquisition.
  - *Alicia P. Wolfe*: Conceptualization; Methodology; Formal analysis and investigation; Software; Writing - review and editing; Funding acquisition.
  - *Xiaolan Zhang*: Conceptualization; Methodology; Formal analysis and investigation; Software; Writing - review and editing.
  - *Bing Wang*: Conceptualization; Methodology; Formal analysis and investigation; Writing - review and editing; Funding acquisition.

References

[1] Toh, C.K.: Ad Hoc Mobile Wireless Networks: Protocols and Systems. Prentice Hall, Saddle River, NJ (2001)

[2] Sommer, C., Dressler, F.: Vehicular Networking. Cambridge University Press, Cambridge, UK (2014)

[3] Gerla, M., Lee, E., Pau, G., Lee, U.: Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In: Proc. IEEE World Forum on Internet of Things (2014)

[4] Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L., Rubenstein, D.: Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In: Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (2002)

[5] Zhang, P., Sadler, C.M., Lyon, S.A., Martonosi, M.: Hardware design experiences in ZebraNet. In: Proc. of ACM SenSys (2004)

[6] Oliveira, L.F.P., Moreira, A.P., Silva, M.F.: Advances in forest robotics: A state-of-the-art survey. Robotics **10**(2) (2021)

[7] Albaladejo, C., Sánchez, P., Iborra, A., Soto, F., López, J.A., Torres, R.: Wireless sensor networks for oceanographic monitoring: A systematic review. Sensors **10**(7) (2010)

[8] Huang, J.-H., Amjad, S., Mishra, S.: CenWits: a sensor-based loosely coupled search and rescue system using witnesses. In: Sensys (2005)

[9] Jiang, L., Huang, J.-H., Kamthe, A., Liu, T., Freeman, I., Ledbetter, J., Mishra, S., Han, R., Cerpa, A.: SenSearch: GPS and witness assisted tracking for delay tolerant sensor networks. In: Proc. of International Conference on Ad-hoc and Wireless Networks (ADHOC-NOW) (2009)

[10] Robinson, W.H., Lauf, A.P.: Resilient and efficient MANET aerial communications for search and rescue applications. In: 2013 International Conference on Computing, Networking and Communications (ICNC), pp. 845–849 (2013)

[11] Ramanathan, R., Allan, R., Basu, P., Feinberg, J., Jakllari, G., Kawadia, V., Loos, S., Redi, J., Santivanez, C., Freebersyser, J.: Scalability of mobile ad hoc networks: Theory vs practice. In: MILCOM (2010)

[12] Poularakis, K., Qin, Q., Nahum, E.M., Rio, M., Tassiulas, L.: Flexible SDN control in tactical ad hoc networks. Ad Hoc Networks **85**, 71–80 (2019)

[13] Bello, O., Zeadally, S.: Intelligent device-to-device communication in the internet of things. IEEE Systems Journal **10**(3), 1172–1182 (2014)

[14] Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Mobile Computing, pp. 153–181. Kluwer Academic Publishers, Alphen aan den Rijn, Netherlands (1996)

[15] Perkins, C., Belding-Royer, E., Das, S.: RFC3561: Ad hoc on-demand distance vector (AODV) routing. RFC Editor (2003)

[16] Clausen, T., Jacquet, P., Adjih, C., Laouiti, A., Minet, P., Muhlethaler, P., Qayyum, A., Viennot, L.: Optimized link state routing protocol (OLSR) (2003)

[17] Perkins, C.E., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing for mobile computers. In: ACM SIGCOMM (1994)

[18] Spyropoulos, T., Psounis, K., Raghavendra, C.S.: Single-copy routing in intermittently connected mobile networks. In: 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004., pp. 235–244 (2004)

[19] Spyropoulos, T., Psounis, K., Raghavendra, C.S.: Efficient routing in intermittently connected mobile networks: The multiple-copy case. IEEE/ACM transactions on networking **16**(1), 77–90 (2008)

[20] Vahdat, A., Becker, D.: Epidemic routing for partially connected ad hoc networks. Duke Univ., Durham, NC, Tech. Report, CS-200006 (2000)

[21] Manfredi, V., Crovella, M., Kurose, J.: Understanding stateful vs stateless communication strategies for ad hoc networks. In: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, pp. 313–324 (2011)

[22] Danilov, C., Henderson, T.R., Goff, T., Brewer, O., Kim, J.H., Macker, J., Adamson, B.: Adaptive routing for tactical communications. In: MILCOM 2012-2012 IEEE Military Communications Conference, pp. 1–7 (2012). IEEE

[23] Seetharam, A., Heimlicher, S., Kurose, J., Wei, W.: Routing with adaptive flooding in heterogeneous mobile networks. In: 2015 7th International Conference on Communication Systems and Networks (COMSNETS), pp. 1–8 (2015). IEEE

[24] Lakkakorpi, J., Pitkänen, M., Ott, J.: Adaptive routing in mobile opportunistic networks. In: Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, pp. 101–109 (2010)

[25] Delosieres, L., Nadjm-Tehrani, S.: Batman store-and-forward: The best of the two worlds. In: 2012 IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 721–727 (2012). IEEE

[26] Raffelsberger, C., Hellwagner, H.: Combined mobile ad-hoc and delay/disruption-tolerant routing. In: International Conference on Ad-Hoc Networks and Wireless, pp. 1–14 (2014). Springer

[27] Asadpour, M., Hummel, K.A., Giustiniano, D., Draskovic, S.: Route or carry: Motion-driven packet forwarding in micro aerial vehicle networks. IEEE Transactions on Mobile Computing **16**(3), 843–856 (2016)

[28] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge, MA (2018)

[29] Bengio, Y.: Learning deep architectures for AI. Foundations and trends® in Machine Learning **2**(1), 1–127 (2009)

[30] Dietterich, T.G.: The MAXQ Method for Hierarchical Reinforcement Learning. In: ICML, vol. 98, pp. 118–126 (1998). Citeseer

[31] Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2), 181–211 (1999)

[32] Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete event dynamic systems **13**(1), 41–77 (2003)

[33] Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: Proc. of SIGCOMM (2004)

[34] Tie, X., Venkataramani, A., Balasubramanian, A.: R3: Robust replication routing in wireless networks with diverse connectivity characteristics. In: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, pp. 181–192 (2011)

[35] Yang, C., Stoleru, R.: Hybrid routing in wireless networks with diverse connectivity. In: Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 71–80 (2016)

[36] Boyan, J.A., Littman, M.L.: Packet routing in dynamically changing networks: A reinforcement learning approach. In: Advances in Neural Information Processing Systems, pp. 671–678 (1994)

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

[38] Kumar, S., Miikkulainen, R.: Confidence-based Q-routing: an on-line adaptive network routing algorithm. In: Proc. of Artificial Neural Networks in Engineering (1998)

[39] Hu, T., Fei, Y.: 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, pp. 381–384 (2010). IEEE

[40] Elwhishi, A., Ho, P.-H., Naik, K., Shihada, B.: ARBR: Adaptive reinforcement-based routing for DTN. In: IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications, pp. 376–385 (2010)

[41] Rolla, V.G., Curado, M.: A reinforcement learning-based routing for delay tolerant networks. Engineering Applications of Artificial Intelligence **26**(10), 2243–2250 (2013)

[42] Ye, D., Zhang, M., Yang, Y.: A multi-agent framework for packet routing in wireless sensor networks. Sensors **15**(5), 10026–10047 (2015)

[43] Valadarsky, A., Schapira, M., Shahaf, D., Tamar, A.: Learning to route with deep RL. In: NIPS Deep Reinforcement Learning Symposium (2017)

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

[45] Di Valerio, V., Presti, F.L., Petrioli, C., Picari, L., Spaccini, D., Basagni, S.: Carma: Channel-aware reinforcement learning-based multi-path adaptive routing for underwater wireless sensor networks. IEEE Journal on Selected Areas in Communications **37**(11), 2634–2647 (2019)

[46] Mukhutdinov, D., Filchenkov, A., Shalyto, A., Vyatkin, V.: Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system. Future Generation Computer Systems **94**, 587–600 (2019)

[47] Suarez-Varela, J., Mestres, A., Yu, J., Kuang, L., Feng, H., Barlet-Ros, P., Cabellos-Aparicio, A.: Feature engineering for deep reinforcement learning based routing. In: Proc. of IEEE International Conference on Communications (ICC), pp. 1–6 (2019)

[48] You, X., Li, X., Xu, Y., Feng, H., Zhao, J.: Toward packet routing with fully-distributed multi-agent deep reinforcement learning. In: 2019 International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT), pp. 1–8 (2019)

[49] You, X., Li, X., Xu, Y., Feng, H., Zhao, J., Yan, H.: Toward packet routing with fully distributed multiagent deep reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics: Systems (2020)

[50] Chen, L., Hu, B., Guan, Z.-H., Zhao, L., Shen, X.: Multiagent meta-reinforcement learning for adaptive multipath routing optimization. IEEE Transactions on Neural Networks and Learning Systems (2021)

[51] Manfredi, V., Wolfe, A., Wang, B., Zhang, X.: Relational deep reinforcement learning for routing in wireless networks. In: Proceedings of the 22nd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (2021)

[52] Almasan, P., Suárez-Varela, J., Rusek, K., Barlet-Ros, P., Cabellos-Aparicio, A.: Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case. Computer Communications **196**, 184–194 (2022)

[53] Li, F., Song, X., Chen, H., Li, X., Wang, Y.: Hierarchical routing for vehicular ad hoc networks via reinforcement learning. IEEE Transactions on Vehicular Technology **68**(2), 1852–1865 (2018)

[54] Schüler, C., Patchou, M., Sliwa, B., Wietfeld, C.: Robust machine learning-enabled routing for highly mobile vehicular networks with PARRoT in ns-3. In: Proceedings of the Workshop on ns-3, pp. 88–94 (2021)

[55] Lolai, A., Wang, X., Hawbani, A., Dharejo, F.A., Qureshi, T., Farooq, M.U., Mujahid, M., Babar, A.H.: Reinforcement learning based on routing with infrastructure nodes for data dissemination in vehicular networks (RRIN). Wireless Networks, 1–16 (2022)

[56] Luo, L., Sheng, L., Yu, H., Sun, G.: Intersection-based v2x routing via reinforcement learning in vehicular ad hoc networks. IEEE Transactions on Intelligent Transportation Systems (2021)

[57] Feng, M., Qian, L., Xu, H.: Multi-robot enhanced MANET intelligent routing at uncertain and vulnerable tactical edge. In: IEEE Military Communications Conference (MILCOM), pp. 1–9 (2018)

[58] Sliwa, B., Schüler, C., Patchou, M., Wietfeld, C.: Parrot: Predictive adhoc routing fueled by reinforcement learning and trajectory knowledge. In: 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), pp. 1–7 (2021)

[59] Rovira-Sugranes, A., Afghah, F., Qu, J., Razi, A.: Fully-echoed q-routing with simulated annealing inference for flying adhoc networks. IEEE Transactions on Network Science and Engineering **8**(3), 2223–2234 (2021)

[60] Schüler, C., Sliwa, B., Wietfeld, C.: Towards machine learning-enabled context adaption for reliable aerial mesh routing. In: 2021 IEEE 94th

Vehicular Technology Conference (VTC2021-Fall), pp. 1–7 (2021)

[61] Qiu, X., Xu, L., Wang, P., Yang, Y., Liao, Z.: A data-driven packet routing algorithm for an unmanned aerial vehicle swarm: A multi-agent reinforcement learning approach. IEEE Wireless Communications Letters **11**(10), 2160–2164 (2022)

[62] Johnston, M., Danilov, C., Larson, K.: A reinforcement learning approach to adaptive redundancy for routing in tactical networks. In: IEEE Military Communications Conference (MILCOM), pp. 267–272 (2018)

[63] Sharma, D.K., Rodrigues, J.J., Vashishth, V., Khanna, A., Chhabra, A.: RLProph: a dynamic programming based reinforcement learning approach for optimal routing in opportunistic IoT networks. Wireless Networks **26**(6), 4319–4338 (2020)

[64] Han, C., Yao, H., Mai, T., Zhang, N., Guizani, M.: QMIX aided routing in social-based delay-tolerant networks. IEEE Transactions on Vehicular Technology **71**(2), 1952–1963 (2021)

[65] Jianmin, L., Qi, W., Chentao, H., Yongjun, X.: Ardeep: Adaptive and reliable routing protocol for mobile robotic networks with deep reinforcement learning. In: IEEE 45th Conference on Local Computer Networks (LCN), pp. 465–468 (2020)

[66] Kaviani, S., Ryu, B., Ahmed, E., Larson, K., Le, A., Yahja, A., Kim, J.H.: Deepcq+: Robust and scalable routing with multi-agent deep reinforcement learning for highly dynamic networks. In: IEEE Military Communications Conference (MILCOM), pp. 31–36 (2021)

[67] Getoor, L., Taskar, B.: Introduction to statistical relational learning, vol. 1. MIT press Cambridge **10**, 1296231 (2007)

[68] Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., Pfeffer, A., Abbeel, P., Wong, M.-F., Meek, C., Neville, J., *et al.*: Introduction to Statistical Relational Learning. MIT press, Cambridge, MA (2007)

[69] Struyf, J., Blockeel, H.: Relational learning (2010)

[70] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)

[71] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the International Conference on Representation Learning (ICLR) (2017)

[72] Battaglia, P., Hamrick, J.B.C., Bapst, V., Sanchez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G.E., Vaswani, A., Allen, K., Nash, C., Langston, V.J., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. arXiv (2018)

[73] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S.L., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)

[74] Van Moffaert, K., Nowé, A.: Multi-objective reinforcement learning using sets of pareto dominating policies. The Journal of Machine Learning Research **15**(1), 3483–3512 (2014)

[75] Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., *et al.*: A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems **36**(1), 1–59 (2022)

[76] Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)

[77] Tassiulas, L., Ephremides, A.: Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In: IEEE Conference on Decision and Control (1990)

[78] Chollet, F., et al.: Keras. https://github.com/fchollet/keras

[79] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org (2015). http://tensorflow.org/

[80] Navidi, W., Camp, T.: Stationary distributions for the random waypoint mobility model. IEEE transactions on Mobile Computing **3**(1), 99–108 (2004)

[81] Navidi, W., Camp, T., Bauer, N.: Improving the accuracy of random

waypoint simulations through steady-state initialization. In: Proceedings of the 15th International Conference on Modeling and Simulation, pp. 319–326 (2004)

[82] Le Boudec, J.-Y., Vojnovic, M.: Perfect simulation and stationarity of a class of mobility models. In: Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies., vol. 4, pp. 2743–2754 (2005)

[83] Liang, B., Haas, Z.J.: Predictive distance-based mobility management for multi-dimensional pcs networks. In: IEEE/ACM Transactions on Networking, vol. 11 (2003)

[84] Bai, F., Helmy, A.: 1. A Survey of Mobility Modeling and Analysis in Wireless AdHoc Networks, (2006)

[85] Aschenbruck, N., Ernst, R., Gerhards-Padilla, E., Schwamborn, M.: Bonnmotion: a mobility scenario generation and analysis tool. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, pp. 1–10 (2010)

[86] Liu, Y., Ding, J., Zhang, Z.-L., Liu, X.: CLARA: A Constrained Reinforcement Learning Based Resource Allocation Framework for Network Slicing. In: IEEE International Conference on Big Data (Big Data) (2021)

[87] Xu, Y., Zhao, Z., Cheng, P., Chen, Z., Ding, M., Vucetic, B., Li, Y.: Constrained reinforcement learning for resource allocation in network slicing. IEEE Communications Letters **25**(5) (2021)