Robust and Scalable Routing with Multi-Agent Deep Reinforcement Learning for MANETs

Saeed Kaviani\(^1\), Bo Ryu\(^1\), Ejaz Ahmed\(^1\), Kevin Larson\(^2\), Anh Le\(^1\), Alex Yahja\(^1\), and Jae H. Kim\(^2\)

\(^1\)EpiSys Science, Inc.
\(^2\)Boeing Research and Technology

\{saeed, bo.ryu, ejaz, anhle, alex\}@episci.com
\{kevin.a.larson, jae.h.kim\}@boeing.com

We address the packet routing problem in highly dynamic mobile ad-hoc networks (MANETs). In the network routing problem each router chooses the next-hop(s) of each packet to deliver the packet to a destination with lower delay, higher reliability, and less overhead in the network. In this paper, we present a novel framework and routing policies, deepCQ+ routing, using multi-agent deep reinforcement learning (MADRL) which is designed to be robust and scalable for MANETs. Unlike other deep reinforcement learning (DRL)-based routing solutions in the literature, our approach has enabled us to train over a limited range of network parameters and conditions, but achieve realistic routing policies for a much wider range of conditions including a variable number of nodes, different data flows with varying data rates and source/destination pairs, diverse mobility levels, and other dynamic topology of networks.

We demonstrate the scalability, robustness, and performance enhancements obtained by deepCQ+ routing over a recently proposed model-free and non-neural robust and reliable routing technique (i.e., CQ+ routing). deepCQ+ routing outperforms non-DRL-based CQ+ routing in terms of overhead while maintains same goodput rate. Under a wide range of network sizes and mobility conditions, we have observed the reduction in normalized overhead of 10-15\%, indicating that the deepCQ+ routing policy delivers more packets end-to-end with less overhead used. To the best of our knowledge, this is the first successful application of MADRL for the MANET routing problem that simultaneously achieves scalability and robustness under dynamic conditions while outperforming its non-neural counterpart. More importantly, we provide a framework to design scalable and robust routing policy with any desired network performance metric of interest.

I. INTRODUCTION

Routing has been one of the most challenging problems in communication and computer networks, especially in distributed and autonomous wireless networks without coordination. In packet routing protocols, each router for a given data packet chooses another node as next-hop (i.e., unicasting) following its routing policy. For distributed routing algorithms, typically there is no coordination and no designated communication between nodes in the network to share information other than the data and acknowledgment (ACK) packets.

The routing problem is even more difficult when the network is highly dynamic, heterogeneous, and/or contains variable data traffics and network size. In particular, this is of great interest for mobile ad-hoc networks (MANET) in the tactical wireless networking domain where nodes are mobile, mobility and traffic patterns are unpredictable, and the network topology and size changes constantly due to the battlefield environmental conditions such as terrain and jammers. Many traditional MANET routing protocols are often unreliable in these environments and require constant re-computation of the end-to-end routes upon network changes. Hence, they are not robust and suffer from a periodic loss in throughputs due to routing table computations. For reliability of packet delivery and rapid exploration in highly dynamic networks, broadcasting (i.e., transmission of a packet to all neighbors) has been added to possible routing decisions \[1, 2, 3, 4\].

To bring robustness to the routing protocols in highly dynamic MANET, many solutions have considered adapting routing protocols to variations in the network conditions, e.g., fish-eye state routing protocol (FSR) \[5\] use adaptive link-state update rates, and the adaptive distance vector (ADV) routing protocol \[6\] use a threshold-based adjustment of the routing update rates based on the network dynamics. While these protocols outperform traditional routing schemes with lower overhead and topology information sharing, they are not responsive enough in highly dynamic MANETs.

The seminal work in \[7\] proposed Q-routing, which uses a reinforcement learning (RL) module (i.e., Q-Learning \[8\]) to route packets and minimize delivery time. Each node decides next-hop based on locally acquired and maintained statistics, i.e., Q-values. Each Q-value represents the quality of each next-hop (or route). For a particular target of minimizing the end-to-end delay in the routing algorithm, the Q-value represents the estimation of the delay for each path. The Q-table is maintained and updated for each pair of destination and next hop consistently through ACK messages. Nodes that transmit a data packet to a neighbor may receive an ACK message. The ACK messages contain a value to update Q-table. Q-routing protocol sends the data packet to the next hop (unicast) with the best Q-value. Q-routing is efficient in low dynamic and static networks. In dynamic networks, Q-values can get outdated and the algorithm requires some adjustments.

The Q-routing framework was improved for dynamic MANET in \[9\] with the addition of confidence level table (i.e., C-values) as so-called CQ-routing protocol. C-values are incremented when Q-value is updated and decremented...
when gets outdated. CQ-routing becomes inefficient in highly dynamic networks as it is based on only uni-casting to a single node. Uni-casting makes the network exploration (Q-value updates) slow when the network changes rapidly. It will be exhaustive to uni-cast to single nodes when neighboring nodes have changed or little information is available. Since we consider the routing in the networks without any designated communication between nodes, CQ-routing protocol provides great foundation to disseminate information through ACK only and monitor the state of the routes systematically. Therefore, CQ-routing protocol is the state-message monitoring protocol that we use as the foundation of this work. Specifically, a recent robust routing protocol proposed for highly dynamic and tactical MANETs developed in [10] which is based on the CQ-routing but with addition of broadcast decision.

Adaptive routing (AR) [11] and smart robust routing (SRR) algorithms [10] are efforts to dynamically switching between unicast and broadcast to improve robustness in tactical networks while reducing high overhead caused by the flooding. The routing technique proposed in [10] uses the CQ-routing protocol (i.e. C and Q-values) but extends it by adding broadcast procedure for high reliability, robustness, and rapid network exploration as needed in the tactical and highly dynamic MANETs. Therefore, it is justified to refer to it as CQ+ routing protocol. Although CQ+ routing uses a simple but efficient switching policy to choose between unicast and broadcast, it has several areas that can be improved. For example, the CQ+ routing policy is threshold-based and its decisions depend on a single network parameter (best path confidence level). It does not have the perspective of the entire network and can settle only on a locally optimal solution. CQ+ routing also does not account for the change rate of network parameters and possible congestion in the forward paths. A routing policy that predicts the network dynamics and changes during packet traveling or queuing over a path especially in large networks can improve routing performance significantly. Hence, we can explore more advanced decision making policies that make these considerations and improve performance.

Adaptive decision making of routing protocols can be interpreted in the reinforcement learning (RL) framework. This has been initiated by the Q-routing protocol where it was based on the concept of Q-learning in RL [7]. Following the Q-routing approach, many other techniques and algorithms from the RL community have been applied to the packet routing and scheduling problem in the communication networks [12], [13], [14], [15], [16], [17], [18], [19]. [12] uses MADRL to design independent deep routing policies for each agent based on an off-policy deep Q-learning RL algorithm. Deep Q-learning approaches are based on so-called value estimation, estimation of the return (expected reward) of the actions at certain state. The problem with the deep Q-learning and value estimation policies is that they are not scalable easily as the expected reward is dependent on many network parameter and conditions that are not known prior to decisions. Moreover, in MADRL-based approaches in the literature like [12], training independent policies for each agent raises the scalability issue. It is unclear how to use the policies trained for a specific number of agents (network size) when the network is extended or shrunk to a different size. A similar deep Q-learning RL-based approach is used in [13], where it considers clusters in the network and accounts for inter-cluster routing performance. They also assume a feedback link is available from the source node to the cluster lead agents. Although it is claimed that their approach is expected to be scalable to larger networks and dynamics but it is not clear how this would scale and perform if only trained on smaller networks. The deep neural network (DNN)-based design of a routing policy is challenging for MANETs with high dynamics, as it is a multi-agent environment. Optimization of the policy for one agent is dependent on the policy and actions of other agents and therefore suffers from the non-stationarity issue. This is particularly difficult in dynamic networks as many network parameters and topology are rapidly changing.

To the best of our knowledge, there have not been works on a scalable and robust routing policy design framework using MADRL in MANET. To provide robustness and reliability, we use a similar approach to CQ+ routing, where CQ-routing is combined with adaptive flooding. In this paper, we use MADRL and advanced RL algorithms and techniques to train a robust and reliable routing policy that can be applied to any network size, traffic, and dynamics. As it is closely related to the CQ+ routing protocol, we refer to it as deep robust routing for dynamic networks (deepCQ+ routing).

II. ROBUST ROUTING FRAMEWORK

We consider a robust routing protocol which monitors the quality and confidence level of the routes (next-hops) via ACK messages, i.e. CQ-routing protocol [9]. We use this protocol as it does not add any designated communication link for coordination between nodes in the network, and it is robust for dynamic networks with addition of confidence level, i.e. C. We specifically use the extended version of CQ-routing by addition of adaptive broadcasting to bring more reliability (higher delivery rates) and rapid network exploration (for highly dynamic networks), as introduced in [10] and we refer to it as CQ+ routing. We summarize the CQ+ routing in Algorithm 1, which shows how it chooses between broadcast and unicast (and next-hop) adaptively.

A. CQ+ routing Protocol

The smart robust routing (SRR) algorithm proposed in the CQ+ routing work [10] uses the network parameters C and H for the routing decisions primarily introduced in seminal CQ-routing [9]. Each node i has a H-factor, h(i, j, d) (i.e. i ⇝ j ⇝ d), which represents an estimate of the least number of hops between node i and destination d which passes through potential next-hop j. To monitor the dynamics of the network, each node i also have a confidence level or C-value, c(i, j, d), that represents the confidence in likelihood the packet will reach its destination h(i, j, d). This C-value is increased and corrected with any packet transmission success (receiving the
ACK). Every packet transmission, the C-value is degraded by a decay factor. C- and H-factors are updated through the $c_{\text{ack}}$ and $h_{\text{ack}}$ which are propagated by the acknowledgement (ACK) packets from the receiving node (e.g., next-hop) to the transmitting node. These ACK values are computed at the next-hop node $j$ as

$$h_{\text{ack}} = 1 + h(j, \hat{k}, d) \quad (1)$$

$$c_{\text{ack}} = c(j, \hat{k}, d) \quad (2)$$

where $\hat{k}$ is the best path (next-hop) estimate of node $j$ to destination $d$ and it is found by

$$\hat{k} = \arg \min_k h(j, d, k) (1 - c(j, k, d)) \quad (3)$$

In other words, C- and H-values are exponential moving average of the $c_{\text{ack}}$ and $h_{\text{ack}}$, respectively. If a transmission fails or there is no ACK to update C- and H-levels, then H-level cannot be updated. However, we degrade the C-level as in $c_{\text{ack}} = 0$ to reflect the path failure. The updates of the C- and H-levels given by

$$h_{t+1}(i, j, d) = (1 - \alpha)h_t(i, j, d) + \alpha h_{\text{ack}} \quad (4)$$

$$c_{t+1}(i, j, d) = \begin{cases} (1 - \lambda)c_t(i, j, d) & \text{failure} \\ (1 - \lambda)c_t(i, j, d) + \lambda c_{\text{ack}} & \text{otherwise} \end{cases} \quad (5)$$

where $0 \leq \alpha \leq 1$ is a discount factor for the new observation with an adaptable value of

$$\alpha = \max (c_{\text{ack}}, 1 - c_t(i, j, d)) \quad (6)$$

$\lambda$ is the decay factor for the new observation of $c_{\text{ack}}$ (or $1 - \lambda$ is the decay factor for the old observation). If packet is received at the destination $d$, then $c_{\text{ack}}$ and $h_{\text{ack}}$ are set to 1, indicating that we have full confidence that we are 1 hop away from destination.

The SRR algorithm for the CQ+ routing is summarized in Algorithm 1 where more details are available in [10]. SRR algorithm includes (i) reception of ACK and consequently updating C- and H-levels, (ii) reception of non-ACK packets by checking for duplication, loops, and pushing into the queue, (iii) transmission of data packets from the queue by routing it according to the policy in use. We simply refer to this decision policy as CQ+ routing policy.

1) CQ+ routing Policy

The CQ+ routing decision policy chooses the next-hop based on the minimization of the information uncertainty and the expected number of hops. This is given by

$$j^* = \arg \min_j h(i, j, d) (1 - c(i, j, d)) \quad (7)$$

The next-hop $j^*$ is considered by the CQ+ routing policy if it unicast the packet to one single node. However, the CQ+ routing policy enables broadcasting to minimize the next-hop information uncertainty. The uncertainty of the next-hop is measured by $1 - c_i(d, j^*)$. Therefore, CQ+ routing decides to broadcast if the uncertainty about information obtained is high to explore the network more and make a reliable transmission by flooding the neighbors. The CQ+ routing policy is probabilistic and it assigns the probability of broadcast as

$$P_{\text{BC}} = \epsilon + (1 - c(i, j^*, d))(1 - \epsilon) = 1 - c(i, j^*, d)\epsilon \quad (8)$$

where a small value $\epsilon$ is used for the minimum probability of broadcast (defined for exploration purposes) and correspondingly $\hat{\epsilon} = 1 - \epsilon$ is the maximum probability of unicast.

In this paper, we use the CQ+ routing protocol, meaning that the reception procedure of our algorithm is the same as described in [1] but our improvements are applied to the transmission procedure and the CQ+ routing policy. In this work, we preserve the protocol and concentrate on routing policy optimization.

### III. DEEP REINFORCEMENT LEARNING FOR THE CQ+ ROUTING (DEEP-CQ+)

#### A. Deep Reinforcement Learning Background

Our robust routing problem fits as a decentralized partially-observable Markov decision process (Dec-POMDP), that is used for decision-making problems in a team of cooperative agents [20]. Dec-POMDP tries to model and optimize the behavior of the agents while considering the environment’s and other agents’ uncertainties. POMDP is defined by a set of states $S$ describing the possible configurations of the agent(s), a set of actions $A$, and a set of observations $O$. The actions are selected using a stochastic policy $\pi_0 : O \times A \rightarrow [0,1]$ (parameterized by $\theta$), which results to the next state defined by the environment’s state transition function $T : S \times A \rightarrow S$. As a result of this transition, a reward is obtained by an agent(s) and it is described as a function of the state and action, i.e. $r : S \times A \rightarrow \mathbb{R}$. Each agent$^2$ receives an observation related

---

Algorithm 1: CQ+ routing algorithm [10]

Receive incoming packet at node $i$:

if Packet is ACK then
  Update $c$ and $h$ from (4) and (5);
else
  if packet traversed a loop;
     then Drop packet, do not return ACK;
  end
  if packet is already in queue then
     Drop packet
     Find best next-hop $j^*$ from (7)
     Compute $c_{\text{ack}}$ and $h_{\text{ack}}$ from (2) and (1) using $j^*$
     Return ACK
  end
  if packet is not duplicate then
     Add packet to the queue
  end
end

if Queue is not empty then
  Pick up packet from queue
  ROUTING DECISION POLICY
  $P_{\text{BC}} \leftarrow \epsilon + (1 - \epsilon)c(i, j^*, d)$
  Choose
     Broadcast with probability $P_{\text{BC}}$
     Unicast to $j^*$ with probability $1 - P_{\text{BC}}$
  if Broadcast then
     Forward packet to all
  end
  if Unicast then
     Forward packet to $j^*$
  end
end

---

$^2$ In the Dec-POMDP, these sets are defined for each agent separately
to the state as \( o : S \rightarrow O \). Below, we give an overview and brief introduction of the DRL algorithms and techniques used in this work to design the deepCQ+ routing policies.

1) Value Optimization and Deep Q-learning

To find the optimal policy, Q-Learning\(^4\) is a widely used model-free algorithm that estimates the action-value function for policy \( Q^\pi(s,a) \). The action-value function (or Q-function) is the expected maximum sum of (discounted) rewards, perceived at state \( s \) when taking action \( a \) and it is given by

\[
Q^\pi(s,a) := \mathbb{E}\left[ \sum_{t=0}^{T} \gamma^t r_t \mid s_t = s, a_t = a \right]
\]

where \( \gamma \) is the discount factor and \( T \) is the time horizon. The action-value function can be recursively written (and calculated) as \( Q^\pi(s,a) = \mathbb{E}_{a'}[r(s,a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^\pi(s',a')]] \).

The recent deep learning paradigm enables the RL algorithms to approximate the Q-function using a deep neural network, i.e. \( Q(s,a) \approx Q(s,a;\theta) \), where \( \theta \) is the set of neural network parameters.\(^3\) A popular method to do this is known as deep Q-network (DQN)\(^1\). DQN learns the optimal action-value function \( Q^* \) by minimizing the loss:

\[
L(\theta) = \mathbb{E}_{s,a,r,s'} \left[ (Q^*(s,a;\theta) - (r + \gamma \max_{a'} Q^*(s',a')))^2 \right]
\]

where \( \bar{Q} \) is a target Q-function with parameters periodically (or gradually) updated with the most recent parameters \( \theta \) to further stabilize the learning. DQN also benefits from a large experience replay buffer \( \mathcal{B} \) of experience tuples \( (s,a,r,s') \).

Finally, the optimal (deterministic) policy is described as

\[
\pi^*(s) = \arg \max_{a \in A} Q(s,a;\theta^*)
\]

when the optimal parameters \( \theta^* \) are found.

2) Policy Optimizations

Policy gradient methods are another popular choice that is based on the optimization of the policy directly. Let \( \pi \) denote a stochastic policy which assigns a probability \( \pi(a_t|s_t) \) for an action \( a_t \) given a state \( s_t \). In the policy optimization methods, the main idea is to optimize the parameters of the policy, \( \theta \), to maximize the expected discounted return objective function,

\[
J(\theta) = \mathbb{E}_{s_0,a_0,s_1,...,s_T} \left[ \sum_{t=0}^{T} \gamma^t r_t \right]_{R_\tau}
\]

where the experience sequence (or path) \( \tau \) is denoted as \( \{s_0,a_0,s_1,...\} \) with \( s_0 \sim p_0(s_0) \) and \( a_{t+1} \sim \pi(a_t|s_t), s_{t+1} \sim T(s_{t+1}|s_t,a_t) \). The discounted return for the sequence \( \tau \) is expressed by \( R_\tau \). \( p_0 \) is the distribution of the initial state \( s_0 \).

The action-value function \( Q^\pi \), and the value function \( V^\pi \), and the advantage function \( A^\pi \) are defined as

\[
Q^\pi(s_t,a_t) = \mathbb{E}_{s_{t+1},a_{t+1},...} \left[ \sum_{l=0}^{T} \gamma^l r_{t+l} \right], \quad (13)
\]

\[
V^\pi(s_t) = \mathbb{E}_{a_{t+1},s_{t+1},...} \left[ \sum_{l=0}^{T} \gamma^l r_{t+l} \right], \quad (14)
\]

\[
A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t), \quad (15)
\]

where \( a_t \sim \pi(a_t|s_t) \) and \( s_{t+1} \sim T(s_{t+1}|s_t,a_t) \).

A class of popular policy optimization methods, policy gradient (PG), optimizes the policy parameters \( \theta \) by descending toward the gradient direction \( \nabla_\theta J(\theta) \). Therefore, we review how this gradient direction can be expressed and simplified and empirically calculated. Using the fact that \( \nabla_x \log y = \frac{\nabla y}{y} \), we can rewrite the gradient \( \nabla_\theta \mathbb{E}_r \) as

\[
\nabla_\theta J(\theta) = \sum \nabla_\theta P(\tau;\pi_\theta) R_\tau = \sum \nabla_\theta \log P(\tau;\pi_\theta) R_\tau = \mathbb{E}_r \left[ \nabla_\theta \log P(\tau;\pi_\theta) R_\tau \right]
\]

In other words, the gradient direction \( \nabla_\theta J(\theta) \) to maximize the expected return can be interpreted as expectation of the gradient of the log-likelihood of the paths, i.e. \( \nabla_\theta \log P(\tau;\theta) \) weighted by the path return value \( R_\tau \). This means that the descending in the gradient direction will increase the probability of paths with positive returns, and decrease the probability of path with negative returns. One approach to empirically compute this gradient direction is to average this term over all observed paths through our search as given by

\[
\nabla_\theta J(\theta) \approx \frac{1}{M} \sum_{m=0}^{M} \nabla_\theta \log P(\tau^{(m)};\theta) R_{\tau^{(m)}}.
\]

Nevertheless, this requires finding \( \nabla_\theta \log P(\tau^{(m)};\theta) \) empirically. Now, the probability of the experience path \( \tau \) can be computed by decomposing the path into states and actions. Hence, we have

\[
P(\tau;\theta) = \prod_{t=0}^{\infty} T(s_{t+1}|s_t,a_t) \cdot \pi(a_t|s_t;\theta)
\]

\[
\log P(\tau;\theta) = \sum_{t=0}^{\infty} \log T(s_{t+1}|s_t,a_t) + \sum_{t=0}^{\infty} \pi(a_t|s_t;\theta)
\]

Using \( (19) \) and the fact that the environment’s state transition \( T(s_{t+1}|s_t,a_t) \) is independent of \( \theta \), the gradient of the log-likelihood of path \( \tau \), i.e. \( \nabla_\theta \log P(\tau;\theta) \), can decompose into log-likelihood sum of the stochastic policy \( \pi(a_t|s_t) \) as given in

\[
\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t|s_t;\theta)
\]
Hence, the gradient can be written as

$$\nabla \theta J(\theta) = E_t \left[ \sum_t \nabla \theta \log \pi(\theta|s_t; \theta) R_t \right]$$  \hspace{1cm} (22)

Now, from the gradient estimator of the Vanilla policy gradient algorithm [23], we know that the gradient expression can be simplified as

$$\nabla \theta J(\theta) = E_t [\nabla \theta \log \pi_\theta(\theta|s_t) A_t]$$  \hspace{1cm} (23)

Let $\eta_{ti}$ denote the probability ratio

$$\eta_{ti} = \frac{\pi_\theta(a|s_t)}{\pi_{\theta_{old}}(a|s_t)}$$  \hspace{1cm} (24)

and therefore $\eta_{told} = 1$. A popular state-of-the-art policy gradient algorithm is called proximal policy optimization (PPO) [24], [25], which uses the clipped objective function

$$L_{\text{clip}}(\theta) = E_t \left[ \min \left( \eta_{ti} A_t, \text{clip}(\eta_{ti}, 1 - \epsilon, 1 + \epsilon) A_t \right) \right]$$  \hspace{1cm} (25)

as an optimization problem of interest. PPO is indeed a family of policy optimization methods that uses multiple epochs of stochastic gradient ascent to perform each policy update. PPO inherits the stability and reliability of the trust region methods [25] but implemented in a much simpler way. We have used PPO algorithm for the optimization of the routing policy in the following sections.

3) Centralized Training, Decentralized Execution

It is important to understand that in multi-agent reinforcement learning, each agent can have its policy while sharing the environment with other agents. In a communication network setup, partial observability and/or communication constraints necessitate the learning of decentralized policies, which condition only on the local action-observation history of each agent. Decentralized policies also naturally avoid the exponentially growing joint action-space with the number of agents, while the traditional single-agent RL methods are impractical.

Fortunately, decentralized policies can be learned in a centralized fashion specially in a simulated or laboratory setting. The centralized training has access to hidden state information of other agents and removes inter-agent communication constraints. The paradigm of centralized training with decentralized execution has attracted attention in the RL community [26], [27], [20]. However, many challenges surrounding how to best exploit centralized training remain open.

4) Parameter Sharing

In the context of centralized training but decentralized execution, a common strategy is to share the policy parameters between agents that are homogeneous [28], [29], [30] (parameter sharing). Note that in the heterogeneous communication networks, we can always categorize nodes as homogeneous with some individual parameters and those can be also given as input to the policy with shared parameters.

The multi-agent environment for the routing problem and our proposed framework are also summarized in Figure 1 where the centralized training, decentralized execution, and parameter sharing are illustrated.

B. Our proposed DRL Framework and Approach

We consider large wireless and dynamic communication networks with a relatively wide range of sizes (e.g. $10 \leq N \leq 50$). Each node $i$ holds a queue with data packets intended to be delivered to various destinations. The nodes can have any arbitrary traffic data flows. The nodes are consistently moving at various random velocities. The dynamic level of each node is different and random. The mobility of the nodes typically follows a reliable model in MANET, Gauss-Markov model, which is also discussed in Section IV-A2. At each time $t$, each node $i$, picks a packet from its data queue (according to a pre-defined packet scheduling algorithm). We use a simplified first-in-first-out (FIFO) packet scheduling but the routing decisions and policy design are independent of this scheduling method. Each node holds a table of network parameters which indeed contains $C$- and $H$-levels discussed in Section II-A. These network parameters are indeed 2-dimensional $(N - 1) \times (N - 1)$ matrix for each node (i.e. $c(i,j, d)$, $h(i, j, d)$), and it is defined for each destination node id, and for each next-hop $(N - 1$ other nodes). These matrices are initialized when an episode is started. Hence, the network parameter matrices are often sparse as each node may have a limited number of neighbors and therefore many network parameters will stay at their initial reset values.

During the execution, each node $i$ has already loaded a pre-trained policy $\pi_\theta$, where it is also the same as other nodes. Therefore, there is no selection or classification of the nodes to identify which policy to be used for each.

Each episode contains a specific number of packet duration time-slots $L$, with various data rates. Higher data rates may introduce high queue backlogs. We end the episode when the length of the episode exceed a maximum traffic length $L_{\text{max}}$ and all the nodes have empty data queues. We stop any new upcoming data traffic when the episode length is larger than $L_{\text{max}}$. Therefore, episode length is slightly larger than the maximum traffic length. The goodput will be computed as the number of packets delivered (non-duplicate) to the total
number of packets injected as input traffic.

1) Scalability
The main distinction of our work from similar DRL-based routing algorithms is that our solution is scalable in terms of (i) communication network size (ii) network dynamics and mobility levels and topology (iii) different data flows, data rates, source, and destination. In other words, the proposed DNN Policy is designed and trained such that it is not over-fitted to specific configurations and system parameters. It works for a wide range of communication network sizes, various dynamic levels, different sources and destinations, and different data flows.

2) DNN Policy
To satisfy the scalability requirement, the proposed DNN policy needs to have a network architecture to accommodate variable number of agents. The routing decisions are generally dependent on the network parameters (i.e. C- and H-values) of the neighbors. For example, if a subset of agents have zero C-levels then there is no need to play them in our routing decisions. Hence, we perform a pre-processing of the network parameters available (state characteristic) to have fixed number of inputs fed into our DNN policy. Therefore, we select the best K neighbors out of total N agents in the network for each node i and use their network parameters as the current state of that specific node. For the sake of simplicity, we drop the current node i, and destination d from C- and H-levels and use the next-hop as index (i.e. \( c_t(i, j, d) \rightarrow c_t(i, j) \)). We also order the next-hop indices according to ascending order of \( h_j \) (1 - \( c_{ij} \)). Now, in our problem formulation the observation of an agent i at time t obtained from its state is given by

\[
o_t(i) = [c_t(i), h_t(i)],
\]

where

\[
c_t(i) = [c_t(i, i_1), c_t(i, i_2), \ldots, c_t(i, i_K)]
\]

\[
h_t(i) = [h_t(i, i_1), h_t(i, i_2), \ldots, h_t(i, i_K)]
\]

(27)

such that the neighboring nodes \( i_1, \ldots, i_K \) of i are ordered as \( h_t(i, i_1)(1 - c_t(i, i_1)) \leq \cdots \leq h_t(i, i_K)(1 - c_t(i, i_K)) \).

Training individual policy for each node is impractical due to the scalability constraints of the design. For example, if we have trained a policy for one node within a specific network size, it is not clear how the trained policies are assigned to larger network sizes and new nodes added to the network. We consider to train one policy for all, \( \pi(a|s_t; \theta) \). Training a universal policy has a scalability advantage meaning that any node added to the network or any network dynamics, topology, or data flow will have the same policy to be used. This is indeed consistent with the original CQ+ routing policy as it offers the same policy for all, although it is not DNN and it is probabilistic decisions only (see (8)).

We use fully-connected DNN but the established framework can extend it to other DNN architectures. We add the change between current observations and previous observations, and also the previous effective action taken by the agent to the observations at node i that is fed into the DNN policy as input features. Including the previous observations for the input features helps to capture temporal change rate of network parameters. Alternatively, we can benefit from recurrent neural network architectures but we have found that training such networks are typically takes longer. The input features to the DNN policy at node i are given by

\[
o_t(i) = [c_t(i), h_t(i), \Delta c_t(i), \Delta h_t(i), a_{t-1}(i)],
\]

where \( \Delta c_t(i) = c_t(i) - c_{t-1}(i) \) and \( \Delta h_t(i) = h_t(i) - h_{t-1}(i) \).

C. CQ+ routing as a DRL-based Policy
In this section, we reformulate the RL problem so that its optimal policy results in the same policy as the CQ+ routing policy. This particularly helps in defining a proper rewarding system for our CQ+ routing policy improvements and extensions.

First, we consider a stochastic routing policy \( \pi(a|s_t) \) where the action’s probabilities are only dictated by the policy. This is indeed similar to the CQ+ routing policy where it computes a probability of broadcast. We give the following rewards to the unicast action (\( a = 0 \)) and the broadcast action (\( a = 1 \)) as

\[
r_t(s_t, a_t) = \begin{cases} 
1 - c_t(i, i_1)\hat{\epsilon} & a_t = 1 (\text{broadcast}) \\
0 & a_t = 0 (\text{unicast}) 
\end{cases} 
\]

(Reward 1)

(30)

where \( \hat{\epsilon} = 1 - \epsilon \). Note that \( c_t(i, i_1) \) is closely tracks the success/freshness probability of the path via the next-hop. Then, in order to maximize the total expected (discounted) reward as

\[
R = \mathbb{E}_{s_t \sim \tau, a_t \sim \pi} \left[ \sum_{t=0}^{T} \gamma^t r_t \right] = \mathbb{E}_{s_t \sim \tau, a_t} \left[ \sum_{t=0}^{T} \gamma^t \left( \pi(a_t = 0|s_t)c_t(i, i_1)\hat{\epsilon} + \pi(a_t = 1|s_t)(1 - c_t(i, i_1)\hat{\epsilon}) \right) \right]
\]

(31)

Now, with zero horizon (i.e. \( T = 0 \)), maximizing the expected reward given by (31) leads to a policy with probability of broadcast is

\[
\pi(s_t, a_t = 1) = P_{BC} = \begin{cases} 
1 & c_t(i, i_1)\hat{\epsilon} < 1/2 \\
0 & c_t(i, i_1)\hat{\epsilon} > 1/2 .
\end{cases}
\]

(32)

This is indeed the deterministic version of the CQ+ routing routing policy previously discussed as in (8).

D. deepCQ+ routing
An immediate improvement of the CQ+ routing policy using DRL can be pursued as maximization of the expected return,

| Routing Protocol | C/Q-values | Broadcast | MADRL |
|------------------|------------|-----------|-------|
| CQ-routing [1]   | ✓          | ×         | ×     |
| CQ+ routing [10] | ✓ ✓ ✓      | ✓         | ✓     |
| deepCQ+ routing (this work) | ✓ ✓ ✓ | ✓ | ✓ |

TABLE I
COMPARISON OF ROBUST ROUTING PROTOCOLS IN DYNAMIC NETWORKS
As in [31] with $\gamma > 0$ and some large horizon $T$, where the immediate reward $r_i(s_i, a_i)$ is given by [30]. We can use efficient RL algorithms such as PPO to find a DNN policy. We refer to this DNN-based version of the CQ+ routing as deepCQ+ routing with reward type 1. Note that different reward designs will give different performance objectives.

The CQ+ routing policy aims to choose the next-hop that minimizes uncertainties (probability of failure) in dynamic networks. This is done by accounting for the probability of failure for the path with the lowest link uncertainties and number of hops. Although the CQ+ routing policy indirectly optimizes the network throughput, it is not accounted for directly in its optimization. We define the overhead in the routing of the dynamic networks as the ratio of the total number of transmissions in the communication network, denoted by $N_{TX}$, to the total number of packets delivered, denoted by $N_D$. For a specific packet rate and a window of time. The normalized overhead by the network size, $N$, is given by

$$OH = \frac{1}{N} \cdot \frac{N_{TX}}{N_D}.$$  

(33)

To show the effectiveness of the DRL approach on the CQ+ routing policy optimization, the deepCQ+ routing objective is to minimize overhead while keeps the goodput rate $\rho$ at the same level (or higher) than the goodput rate provided by the CQ+ routing, i.e. $\rho_0$ (for the same input data flows and the same horizon). This can be formulated as

$$\min_{\theta} \frac{N_{TX}}{N_D} \text{ such that } \rho \geq \rho_0.$$  

(34)

Since the target is the efficiency of our routing policy, we can simply assume $\rho = \rho_0$, and consequently, $N_D = \rho_0 T$ for some time horizon $T$ (for some specific input data rate). With this approach, we maintain the goodput rate while we achieve lower overhead and it is fair to compare with previous robust routing policies. Therefore, the optimization problem in (34) can be simplified as

$$\min_{\theta} \frac{N_{TX}}{N_D} \text{ such that } N_D = \rho_0 T.$$  

(35)

From our experiments, using the rewarding system in [30] give us the goodput rate as $\rho_0$ while keep the overhead lower than CQ+ routing. Hence, we can consider the overhead minimization by reformulating our rewarding system to accommodate (35) as

$$r_i(t) = w_1 I_D - w_2 I_Z - w_3 \frac{N_{ack}}{N} + \begin{cases} 1 - c_i(i, i_1) & \epsilon a_i = 1 \\ c_i(i, i_1) \epsilon a_i = 0. & \end{cases}$$  

(36)

where $I_D$ is the reward for packet delivery. If a packet is delivered successfully to the destination and node $i$ has contributed to that delivery then it will be rewarded. Note that we do not reward for duplicate delivery of a packet. Also, $I_Z$ is a reward (penalty) indicator which is enabled when we have not received any ACKs from our transmissions. This term is not directly related to the optimization problem (35) but it prevents the system to learn uncasting all the packets initially, which may happen if the initial reward is negative and the agent wants to end the episode to prevent increased penalties.

The next term in the reward, i.e. $N_{ack}/N$, is the normalized number of ACKs received as a result of the action taken and therefore closely represents the number of copies of a packet at other nodes. A naive approach can be to penalize for every transmission directly. However, the cause of transmissions at each node is not the actions taken at that specific node, but it is the transmissions of the packets from other nodes to that specific node. Hence, $N_{ack}$ received at each node is used to estimate the added number of transmissions dictated to the network as a result of each node’s action. The last reward component remains intact from the reward type 1 as it was already outperforming CQ+ routing in terms of overhead while achieving the same goodput (see Figures 4 and 5). The weights of the reward components, (i.e. $w_1, w_2, w_3$), have been tuned according to the overhead minimization problem [35]. The proposed deepCQ+ routing technique with modified (extended) reward definition is referred to as deepCQ+ routing with reward type 2. The deepCQ+ routing algorithms are summarized in Algorithm 2.

**Algorithm 2: The proposed deepCQ+ routing (with reward type 1 and 2)**

- Receive incoming packet at node $i$:
  - if *Packet is ACK* then
    - Update $c$ and $h$ using (4) and (5)
  - else
    - if *packet traversed a loop* then
      - Drop packet, do not return ACK
    - else
      - if *packet is already in queue* then
        - Drop packet
      - else
        - if *packet is not duplicate* then
          - Add packet to the queue
      - end
    - end
  - end
- if *Queue is not empty* then
  - Pick up packet from queue of node $i$ (at current time $t$)
  - Pre-process best ($K = 4$) neighbors using (28)
  - Form the input to the DNN-based routing policy
    - $\alpha_i(i) = \{c_i(i), h_i(i), c_{i-1}(i), h_{i-1}(i), \alpha_{i-1}(i)\}$
  - ROUTING DECISION POLICY
    - deepCQ+ routing (reward 1 or 2):
      - Create $\theta_{deepCQ+ reward-1 or -2}$
      - Choose
        - Broadcast with probability $\pi_\theta(a = 1|\alpha_i(i); \theta)$
        - Unicast with probability $\pi_\theta(a = 0|\alpha_i(i); \theta)$
  - if *Decision is Broadcast* then
    - Forward packet to all
  - else
    - if *Decision is Unicast* then
      - Forward packet to $i_1$
  - end
- end

IV. EXPERIMENTS AND NUMERICAL RESULTS

A. Environment Modelling and Training Platforms

To model the CQ+ routing environment and for rapid development of the design, algorithms, and testing, we have
developed a CQ+ routing network simulator and constructed an RL CQ+ routing environment to train and test our approach and the CQ+ routing baselines. Our environment platform is built in Python and it has all CQ+ routing protocol features including generation of random dynamic networks with data flows, different number of nodes, and configurable network setup parameters (dynamic levels, range, node locations, data rates, source and destinations, data queues and backlogs, link quality computation, etc.). The CQ+ routing protocol contains the tracking, distributing C- and H-values, duplicate packet checking, etc. It also extracts performance evaluation information. The environment is interfaced with the Ray, which is a powerful distributed computing platform for the machine learning and RLlib library which provides scalable software primitives for the RL algorithms.

A snapshot of the environment is given in Figure 2, where the source and destination are in colored as gray. The color of the nodes shows the queue length or the congestion level of the nodes. The color of the links is associated with their quality with green links represents high quality and low packet error rates. The thickness of the links shows the traffic level of the link. The heatmap images of the C- and H-levels are monitored to show the state of the network visually. The heatmap images of $C_i(d,j)$ are 2-D vectors as they consider single destination node (i.e. node $d = 25$). Each row is associated with the current node $i$ and each column corresponds to the other nodes $j$ of each current node $i$. For example, the C-levels for the other nodes that are closer to the destination ($j \geq 20$) are relatively high as there is high confidence of the path going to the destinations through them.

1) Benchmark Topology

A benchmark topology was considered based on the adaptive routing work, and the CQ+ routing. In this topology scenario, there are $N$ ($N \geq 30$) nodes considered in the network which are randomly located in an area. The nodes between the source and destination nodes are moving in random directions with variable speeds. The closer they are to the source or destination, the slower their speed is, reflecting the heterogeneous tactical network environments. An example that is considered in the training and testing in the area of 800 (m) by 300 (m) with the wireless range of the nodes is 150 (m). The packet error rates drop rapidly when the distance between nodes exceeds that range. We assume no interference between transmissions to focus on the routing layer. Source and destination nodes exhibit a slow mobility pattern, similar to the nodes in the outermost regions. The data flow between the source and destination nodes can have various data rates (e.g. 20 packets per second at up to 1000 bytes each; each flow generates up to 160Kbps payload traffic). In different experiments, we use different routing policies starting from the generic CQ+ routing policy. The same network topology will be used to compare other routing algorithms such as deepCQ+ routing with reward types 1 and 2. For both testing and training (to monitor the policy training progress), we monitor the resulting performance metrics such as broadcast rate (the percentage of the broadcast actions in the network), the goodput rate. Goodput is the rate of the delivery of intended data packets that are successfully received at the destination. We exclude any duplicate packets from counting as delivered packets, and in the RL context, we do not count them towards any sort of reward. More importantly, we consider the total overhead as a metric that quantifies the efficiency of our decisions and it is the ratio between the total number of data packets delivered to the total transmissions that have been made. Other network metrics are also recorded, such as the average number of hops between source and destination. Factors such as overhead and the number of hops are normalized by the size of the network so they apply to networks of different scales.

2) Mobility Model

In the mobile ad-hoc networks (MANET), there have been various mobility models proposed and discussed recently and particularly in their impact on the network routing protocols. The MANET mobility models represent the moving behavior of each mobile node in the MANET. MANET mobility models are particularly important from our training and testing perspective. These models need to be realistic for the dynamic tactical wireless networks, while still need to have enough randomization to cover corner cases and extreme scenarios in our training and properly reflect the performance impact of the network dynamics, scheduling, and resource allocation protocols. The random waypoint mobility model is often used in the simulation study of MANET, despite some unrealistic movement behaviors, such as exhibiting sudden stops and sharp turns. The Gauss-Markov mobility model has been shown to solve both of these problems. Therefore, we have used this model in our training and testing process, and implemented in our network simulator platform. To be complete, we have also tested the trained policies using the Gauss-Markov model on the networks that follow random way-point model and observed similar performance and behavior. Compared to the random way-point model, the Gauss-Markov mobility model has improved modeling performance at higher dynamics (e.g. as fast as fast automobiles) while it holds the same performance as a random way-point at human running speeds. For the training of the MANET, the important factor is the sensitivity of the throughput (or goodput) and the end-to-end delay to the different levels of the randomness settings, and the Gauss-Markov model shows no effect on the accuracy of these metrics.

In the Gauss-Markov model, the velocity of mobile node is assumed to be correlated over time and modeled as a Gauss-Markov stochastic process. In a two-dimensional simulation and emulation field (as in this study), the value of speed and direction at the $n$th time instance is calculated on the basis of the value of speed and direction at the $n-1$th time instance and a random variable using the following equations:

$$v^{(n)} = \mu v^{(n-1)} + (1 - \mu) \bar{v} + \sqrt{(1 - \alpha^2)} \tilde{v}^{(n-1)}$$

$$\phi^{(n)} = \mu \phi^{(n-1)} + (1 - \mu) \bar{\phi} + \sqrt{(1 - \mu^2)} \tilde{\phi}^{(n-1)}$$

where $v(n)$ and $\phi(n)$ are the new speed and direction of the node at time interval $n$; $0 \leq \mu \leq 1$ is the tuning parameter for the randomness (and correlation to previous time instance);
Fig. 2. An example of our developed CQ+ routing environment (simulator) for a 25-node dynamic network.

\( \bar{v} \) and \( \bar{\phi} \) are constants representing the mean value of speed (i.e., dynamic level) and direction as \( n \to \infty \); \( \tilde{v}^{(n-1)} \) and \( \tilde{\phi}^{(n-1)} \) are random values from a Gaussian distribution to add randomness. In a two-dimensional simulation and emulation field (as our study), the Gauss-Markov model gives the next location based on the current location at time instance \( n \) as

\[
x^{(n)} = x^{(n-1)} + s^{(n)} \cos(\phi^{(n)}) \tag{39}
\]

\[
y^{(n)} = y^{(n-1)} + s^{(n)} \sin(\phi^{(n)}) \tag{40}
\]

where \( (x^{(n)}, y^{(n)}) \) and \( (x^{(n-1)}, y^{(n-1)}) \) are the \( x \) and \( y \) coordinates of the MANET node’s location at the \( n \)th and \( (n - 1) \)th time intervals, respectively. The mean angle will be adjusted when nodes reach the region edges to limit the movement within the region.

To more accurately reflect the benchmark topology used in the CQ+ routing papers (discussed in the previous section), we have divided the environment into 5 groups symmetrically between the source and destination nodes. The source and destination regions are at the left and right corners of the area. The closer the region is to the center, the faster the nodes move. Also, the regions are overlapping by %10 to prevent too many network partitions. The central region has double the speed variance of that of the mid-left and mid-right regions. The source and destination regions have half the speed variance compared to the mid-right/left regions. The mobility of the MANET nodes is simulated and shown in Figure 3.

**B. Hyperparameter Tuning and Configuration Parameters**

In this section, we discuss the hyperparameter tuning process and list the parameters that obtained through tuning. We also itemize the configuration parameters used in the training process to generate the CQ+ routing environment.

| Parameters         | Value                  |
|--------------------|------------------------|
| number of nodes    | 10-30                  |
| learning rate \( \gamma \) | 0.00005               |
| episode length     | 3000 packet length     |
| discount factor \( \gamma \) | 0.99               |
| network size       | FCC(256, 128, 64, 32)  |

**C. Numerical Results**

The results in Figure 7 confirms that in general, the deepCQ+ routing with reward type 2 outperforms the generic (non-DRL based) CQ+ routing technique. Note that the test is performed over network sizes of 10 to 30 while the training is only performed on the 12-node networks. This is evidence that the CQ+ routing is scalable for different network sizes and dynamics. The main performance metric of interest is the normalized overhead as it shows the efficiency of the network routing in terms of delivery of intended data packets. The
Fig. 4. Goodput rate (delivery rate) while training vs. number of steps.

Fig. 5. Normalized overhead metrics during training vs. number of steps.

normalized overhead is indeed the total overhead (transmitted) packet rate in the network divided by the goodput rate. The normalized overhead is further divided by the network size to be fair in comparison across different network sizes. The normalized overhead is at least 15% lower in deepCQ+ routing with reward type 2 compared with the generic non-DRL-based CQ+ routing while the goodput rate is about the same. Lower normalized overhead values are indicative of more efficient policies. Note that this is only an example of our DRL framework routing policy design to show achieving the target objective (e.g., minimize normalized overhead while maintaining goodput rate) and still being scalable (can be trained for certain network sizes and dynamics but performs satisfactorily for the rest of network parameters).

Our MADRL framework for the network routing policy design also enables us to train over variable network sizes and dynamics if it is required. Figure 8 shows the performance of DeepCQ+ with reward type 2 trained on 12-node networks and the same policy trained over variable network sizes 10 to 30. The results show that the training over 12-node networks scales perfectly for the entire network size domain and although it is possible to train over 10 to 30-node networks, there is not much gain in doing so if any.

V. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we have shown a MADRL approach to design a robust, reliable, and scalable policy for dynamic wireless communication networks. Our DRL framework is specially designed for scalability and enables us to train and test the routing policies for variable network sizes, data rates, and mobility dynamics and performs well in other scenarios. Our DNN-based robust routing for dynamic networks policy, i.e., deepCQ+ routing, is based on the CQ-routing but also monitors network statistics to improve broadcast/unicast decisions. It is shown that deepCQ+ routing is much more efficient than traditional CQ+ routing techniques, significantly decreasing normalized overhead (number of transmissions per number of successfully delivered packets). Moreover, the policy is scalable and uses parameter sharing for all different nodes, makes it possible to use the same trained policy for networks and agents with various mobility dynamics, data rates, and network sizes.

In our future works, we will expand the action space of the deepCQ+ routing to include next-hop selection. We can also extend deepCQ+ routing to accommodate heterogeneous networks with dynamic node types. Another possible direction is to update and redesign the network parameter distribution via ACK, and include more efficient and useful information. The DRL-based policies can be designed and trained to accommodate different performance metrics such as end-to-end delay minimization, overhead minimization, goodput rate maximization, etc.

VI. ACKNOWLEDGEMENT

Research reported in this publication was supported in part by Office of the Naval Research under the contract N00014-19-C-1037. The content is solely the responsibility of the authors and does not necessarily represent the official views of the Office of Naval Research. The authors would like to thank
Fig. 7. Comparison of the results of the deepCQ+ routing with reward type 1 and reward type 2 trained for a 12-node network only versus the CQ+ routing; The results are tested across various network sizes from 10 to 30. Although the deepCQ+ routing PPO policy is trained on 12-node networks, it scales perfectly for various network sizes. The deepCQ+ routing with reward type 2 achieves significantly lower normalized overhead (overhead rate divided by the goodput rate divided by the network size).

Dr. Santanu Das (ONR Program Manager) for his support and encouragement.

REFERENCES

[1] C. E. Perkins and P. Bhagwat, “Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers,” ACM SIGCOMM computer communication review, vol. 24, no. 4, pp. 234–244, 1994.

[2] C. E. Perkins and E. M. Royer, “Ad-hoc on-demand distance vector routing,” in Proceedings WMCSA’99. Second IEEE Workshop on Mobile Computing Systems and Applications. IEEE, 1999, pp. 90–100.

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

[4] J. Moy et al., “OSPF version 2,” 1998.

[5] G. Pei, M. Gerla, and T.-W. Chen, “Fish eye state routing: A routing scheme for ad hoc wireless networks,” in 2000 IEEE International Conference on Communications. ICC 2000. Global Convergence Through Communications: Conference Record, vol. 1. IEEE, 2000, pp. 70–74.

[6] R. V. Boppana and S. P. Konduru, “An adaptive distance vector routing algorithm for mobile, ad hoc networks,” in Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213), vol. 3. IEEE, 2001, pp. 1753–1762.

[7] J. A. Boyan and M. L. Littman, “Packet routing in dynamically changing networks: A reinforcement learning approach,” Advances in Neural Information Processing Systems, 1994.

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

[9] S. Kumar and R. Miikkulainen, “Confidence-based Q-routing: An online adaptive network routing algorithm,” in Proceedings of Artificial Neural Networks in Engineering, 1998.

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

[11] C. Danilov, T. R. Henderson, T. Goff, O. Brewer, J. H. Kim, J. Macker, and B. Adamson, “Adaptive routing for tactical communications,” in MILCOM 2012 - 2012 IEEE Military Communications Conference, 2012, pp. 1–7.

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

[13] R. E. Ali, B. Erman, E. Baştığ, and B. Çiilli, “Hierarchical deep double Q-routing,” in ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020, pp. 1–7.

[14] Z. Mammeri, “Reinforcement learning based routing in networks: Review and classification of approaches,” IEEE Access, vol. 7, pp. 55 916–55 950, 2019.

[15] C. Yu, J. Lan, Z. Guo, and Y. Hu, “DROM: Optimizing the routing in software-defined networks with deep reinforcement learning,” IEEE Access, vol. 6, pp. 64 533–64 539, 2018.

[16] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, “A deep-reinforcement learning approach for software-defined networking routing optimization,” arXiv preprint arXiv:1709.07080, 2017.

[17] H. Ye, G. Y. Li, and B.-H. F. Juang, “Deep reinforcement learning based resource allocation for V2V communications,” IEEE Transactions on Vehicular Technology, vol. 68, no. 4, pp. 3163–3173, 2019.

[18] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, “QoS-aware adaptive
Fig. 8. Scalability of the deepCQ+ routing with reward type 2 trained on 12-node networks is shown when compared with the policy trained over all 10- to 30-node networks (variable network sizes). The policy trained on 12-node shows as good as or even better in terms of normalized overhead on network sizes of 10 to 30.

Routing in multi-layer hierarchical software defined networks: A reinforcement learning approach,” in 2016 IEEE International Conference on Services Computing (SCC). IEEE, 2016, pp. 25–33.

[19] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, “Learning to route,” ser. HotNets-XVI. New York, NY, USA: Association for Computing Machinery, 2017, pp. 185–191. [Online]. Available: https://doi.org/10.1145/3152434.3152441

[20] F. A. Oliehoek, C. Amato et al., A concise introduction to decentralized POMDPs. Springer, 2016, vol. 1.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., “Human-level control through deep reinforcement learning,” nature, vol. 518, no. 7540, pp. 529–533, 2015.

[22] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in Advances in neural information processing systems, 2000, pp. 1057–1063.

[23] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Machine learning, vol. 8, no. 3-4, pp. 229–256, 1992.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.

[25] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in International Conference on Machine Learning, 2015, pp. 1889–1897.

[26] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in Advances in neural information processing systems, 2017, pp. 6379–6390.

[27] L. Kraemer and B. Banerjee, “Multi-agent reinforcement learning as a rehearsal for decentralized planning,” Neurocomputing, vol. 190, pp. 82–94, 2016.

[28] J. K. Gupta, M. Egorov, and M. Kochenderfer, “Cooperative multi-agent control using deep reinforcement learning,” in International Conference on Autonomous Agents and Multiagent Systems. Springer, 2017, pp. 66–83.

[29] J. K. Terry, N. Grammel, A. Hari, L. Santos, B. Black, and D. Manocha, “Parameter sharing is surprisingly useful for multi-agent deep reinforcement learning,” arXiv preprint arXiv:2005.13625, 2020.

[30] X. Chu and H. Ye, “Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning,” arXiv preprint arXiv:1710.00336, 2017.

[31] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan et al., “Ray: A distributed framework for emerging AI applications,” in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 561–577.

[32] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, “Rllib: Abstractions for distributed reinforcement learning,” in International Conference on Machine Learning, 2018, pp. 3053–3062.

[33] A. K. Gupta, H. Sadawarti, and A. K. Verma, “Performance analysis of MANET routing protocols in different mobility models,” International Journal of Information Technology and Computer Science (IJITCS), vol. 5, no. 6, pp. 73–82, 2013.

[34] V. Timcenko, M. Stojanovic, and S. B. Rakas, “MANET routing protocols vs. mobility models: performance analysis and comparison,” in Proceedings of the 9th WSEAS international conference on Applied informatics and communications. World Scientific and Engineering Academy and Society (WSEAS), 2009, pp. 271–276.

[35] J. Ariyakhajorn, P. Wannawilai, and C. Sathitwiriyawong, “A comparative study of random waypoint and gauss-markov mobility models in the
performance evaluation of MANET,” in *2006 International Symposium on Communications and Information Technologies*, 2006, pp. 894–899.

[36] B. Divecha, A. Abraham, C. Grosan, and S. Sanyal, “Impact of node mobility on MANET routing protocols models,” *J. Digit. Inf. Manag.*, vol. 5, no. 1, pp. 19–23, 2007.

[37] F. Bai and A. Helmy, “A survey of mobility models in wireless ad-hoc networks,” 2004.

[38] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa, “Stochastic properties of the random waypoint mobility model,” *Wireless Networks*, vol. 10, no. 5, pp. 555–567, 2004.

[39] B. Liang and Z. J. Haas, “Predictive distance-based mobility management for PCS networks,” in *IEEE INFOCOM’99, Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 3. IEEE, 1999, pp. 1377–1384.