# Identifying Malicious Nodes in Multihop IoT Networks Using Reinforcement Learning

Avery Peiffer and Mai Abdelhakim
Department of Electrical and Computer Engineering, University of Pittsburgh
{aep65, maia}@pitt.edu

***Abstract*—Internet of Things (IoT) technologies have been widely adopted in many systems and critical infrastructures, yet they are vulnerable to various security threats. IoT networks heavily rely on mesh topologies due to their flexibility. A main challenge in multihop mesh networks is the difficulty of locating faults or attacks due to the absence of direct links with a centralized entity. This work focuses on data manipulation attacks in multihop mesh networks, which present a challenging security threat where intermediate devices corrupt information packets as they are being forwarded to the destination. In this paper, we propose a reinforcement learning (RL) approach to identify compromised components in a mesh network. The network is first modeled as a graph, after which a Q-learning algorithm is applied. Without any prior knowledge of the network's compromised devices, the Q-learning algorithm learns to take paths that do not contain compromised devices. The results of the algorithm on sample networks show that the approach learns the safest, most efficient paths from the source to the destination. In addition, by processing the learned Q-values, we can identify compromised or faulty nodes/links in the mesh network with high accuracy. The effectiveness of the proposed approach is demonstrated through simulation results.**

***Index Terms*—Internet of Things, Reinforcement Learning, Routing Attacks, Mesh Networks.**

## I. Introduction

Internet of Things (IoT) technologies are transforming today's systems and critical infrastructure, such as energy systems, cyber-manufacturing, and smart homes. Network connectivity is a defining feature that enables remote monitoring, control and automation. Several IoT network protocols, such as ZigBee and Thread [1], [2], heavily rely on a mesh network topology. For example, in ZigBee, end-devices communicate over a multihop mesh network to reach the coordinator/controller. A key advantage of a mesh network is its flexibility and self-healing capabilities. Specifically, in a mesh topology, devices communicate with each other directly as long as they are within their predefined communication range. Exchanging information beyond the direct communication range is accomplished by routing information over multiple hops, creating a flexible multihop network. If a link fails, alternate links can be used to route the information as connection is not fixed (unlike star or tree topology). Hence, mesh topology is regarded as self-healing.

The main challenge in multihop networks is that it is difficult to identify intermediate faults (or suspicious behavior) due to the absence of direct links with a centralized entity. If there is a path with $N$ hops, i.e., $N-1$ interconnecting/routing devices, and the information is not received correctly by the final destination, either dropped or manipulated, then it is challenging to identify which of the intermediate nodes or links is compromised or is failing. Artificial intelligence/machine learning is expected to provide autonomous tools that would help in revealing problems hidden within meshed network interconnections and in identifying faulty/compromised nodes in a mesh network.

Routing attacks are among the serious security threats in multihop mesh networks [3]. In this work, we focus on the data manipulation attack, where intermediate devices corrupt information packets as they are being forwarded to the destination. This type of attack can be detected at the destination with the use of a keyed hash function or message authentication codes [4]. However, the task of identifying which intermediate device manipulated information is still an open research problem. Such compromised nodes consume network resources by processing and forwarding corrupted information, increase the latency in decision making, and could potentially result in taking wrong actions in an autonomous system. Thus, it is crucial to identify these nodes in the network to replace and/or recover them (e.g. through software patches, firmware upgrades, etc.).

There are several existing techniques aiming at identifying compromised nodes in a network. A popular approach relies on having nodes overhear neighboring transmissions, such as the watchdog techniques [5]. A main limitation with these approaches is in the energy consumed at nodes for continuously overhearing and processing information. In many applications, nodes in a mesh network are resource constrained with limited energy resources. The watchdog approaches also suffer from possible falsification of the monitored results by compromised nodes. Other techniques rely on deploying several nodes in the network for the purpose of identifying compromised relays. For example, in [6] Sentinel nodes are deployed in the network to monitor traffic exchanged by interconnecting devices (relays) utilizing the broadcast wireless channel nature. Directly overhearing relays' transmissions means that a large number of additional nodes would be required to capture all communications. Network diversity has been a key in robust transmission under routing attacks. For example, in [7], [8] we proposed to utilize multiple joint paths simultaneously, and have each relay append specific information to packets as they are forwarded to the destination to help in identifying compromised relays. A main limitation with that approach is

the large overhead in transmitted packets and the large packet sizes that would need to be processed. In [9], we used K-means clustering to divide nodes into benign and malicious groups, and relied on network diversity for accurate clustering.

In this paper, we utilize reinforcement learning to identify compromised nodes launching data manipulation attacks in a mesh network. Reinforcement learning (RL) is a branch of machine learning that has shown tremendous success in video games, such as Atari and AlphaGo, and enables a machine to outperform human performance [10]. In this paper, Q-learning, which is a popular RL algorithm, is applied. In Q-learning, the agent learns "Q-values" for each action at each network state, where in this paper an action specifies a route and a state specifies a node. These Q-values represent how good an action is at a given state. We show that without any prior knowledge of the network's compromised devices, the Q-learning algorithm learns to take paths that do not contain compromised devices. The results of the algorithm on sample networks show that the approach learns the reliable paths from the source to the destination. In addition, by processing the learned Q-values in the algorithm, we can identify compromised or faulty nodes in the mesh network with high accuracy. Furthermore, the proposed approach provides a way to assess the quality of individual links, which is useful for network monitoring and can indicate a link failure or an inconsistent node performance across different links.

This paper is organized as follows. We describe the network and threat model in Section II, and the proposed RL approach for malicious node identification in Section III. The performance evaluation is illustrated in Section IV. The conclusion of this work is presented in Section V.

## II. System and Threat Model

We consider a network segment that is a multihop mesh network of $N$ nodes (IoT devices), including a source node $S$ and a sink node $D$, as shown in Figure 1. The network is modeled as a graph composed of (i) nodes, either end-devices (source or destination) or interconnecting devices (relays), and (ii) edges that represent the links between nodes. Let the $N$ devices be denoted as $\{n_o, n_1, n_2, ...n_{N-1}\}$, where $n_o$ is the source $S$ and $n_{N-1}$ is the destination $D$.

The source $S$ and the sink $D$ are more powerful devices placed in the network to send probe packets periodically over multiple paths between them to examine nodes' behavior. In general, there could be more than two devices to serve this purpose and assess the trustworthiness of nodes in different network segments [11]. Here, we focus on a single network segment. We assume that $S$ and $D$ know or can obtain the network topology, and the source $S$ uses source routing to specify the routing path for each probe packet. That is, when a destination receives a packet, we assume it knows which nodes were along the routing path of that probe packet.

When a probe packet reaches the sink, its integrity is checked using a keyed hash function. Thus, the sink will know if the transmission is corrupted; however, the node(s) that caused the corruption will still be unknown at this point. The source and sink are computationally more capable devices than other nodes and can apply artificial intelligence algorithms (RL). In this work, the sink will act as the RL agent to identify compromised nodes. Once identified, the sink can communicate this information to the source and to a centralized controller to take action (through route management, node replacement or reset, etc.). That centralized controller can be a gateway or a software defined network controller [12]. The source, sink and controller can communicate via different secure channels that do not need to go through the mesh network (e.g. cellular).

The source and sink are assumed to be trusted, while other relays could be compromised. In our system model, the percentage of compromised nodes is $\alpha\%$. Compromised nodes launch data manipulation attacks, such that they will corrupt a transmission sent through them with probability $p$ (with a probability $(1-p)$ a node will act benignly and will forward the packet without manipulation). In the following section, we will describe the proposed algorithm for identifying malicious nodes.

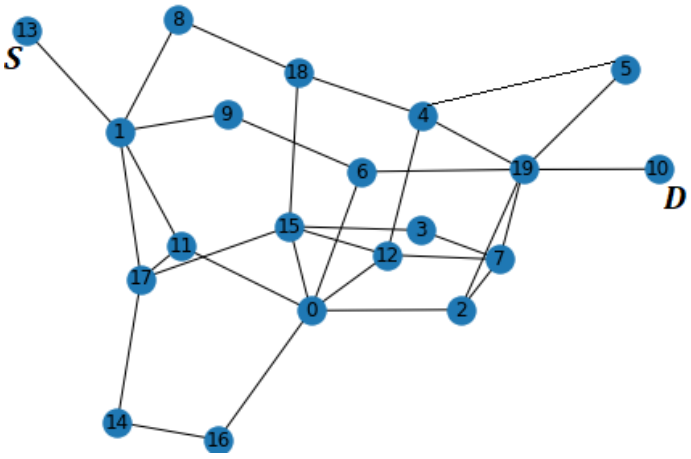


Fig. 1. An example network of IoT devices, $N = 20$.

## III. RL-Based Algorithm for Malicious Node Identification

### A. Reinforcement Learning Overview

Reinforcement learning (RL) aims to learn the best policy to interact with the environment in which it exists. Three key components need to be defined for any RL algorithm, namely states, actions, and rewards. The central idea of RL is that an agent observes the state of the environment, then takes action based on the state, and receives reward (or penalty) depending on the action taken. By interacting with the environment, an RL agent learns which action will maximize its expected future reward at any given environment state.

In other words, let the environment be represented by states, where the state at time $t$ is denoted by $S_t$. Let $A_t$ denote the action that the agent takes at time $t$. As the agent takes $A_t$, the environment states will transition (to $S_{t+1}$) and a reward will be observed by the agent. The reward at time $t$ is denoted

as $R_t$. A general diagram showing these interactions is shown in Figure 2. The objective of RL is to learn a policy that describes how to interact with an environment to maximize the expected future reward, given by $E[\sum_{t=0} \gamma^t R_{t+1}]$, where $\gamma$ is a discount factor for future rewards. In the context of this work, the reward is proportional to the amount of packets arriving correctly to their destination as will be described.

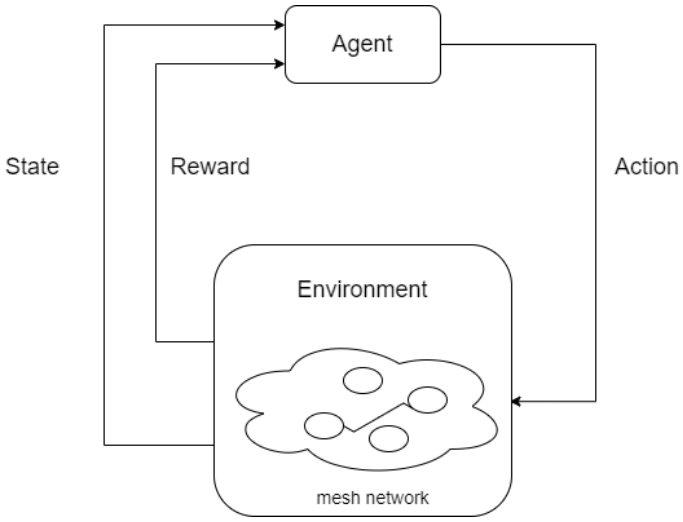


Fig. 2. Generic flow of reinforcement learning.

The RL algorithms that are used by agents to learn the optimal policy can be classified into model-based approaches and model-free approaches [13]. In the former, the environment's state transitions are modeled/learned first, then actions are taken accordingly; by contrast, the latter attempts to find optimal actions based on states without explicit modeling of the environment. In this paper we incorporate one of the well-known model-free RL approaches, called Q-learning, to identify unreliable routes and nodes. In Q-learning, the agent learns a table (called Q-table) with entries corresponding to each state-action pair. The entries are initialized (e.g. with zeros, or -$\infty$), then updated every time step such that the entry $Q(s, a)$ eventually reflects the value of the action $a$ at state $s$ (i.e., how good action $A_t = a$ is at that specific state $S_t = s$).

### *B. Q-Learning for Identifying Malicious Nodes*

In this section, we will describe how Q-learning is utilized to identify compromised nodes in a mesh network described in Section II. In this paper, the state space is all the nodes in the network segment, and the action space represents the possible next hop at a given network state (i.e., at a given node). Recall that the source $S$ generates and sends probe packets to the sink $D$ to examine the network along multiple paths and assess the nodes' behavior as stated earlier. The Q-learning algorithm uses a single agent at the sink to maintain a state and action space for the network. The agent knows the path through the graph, starting at $S$ and moving to a neighboring node at each time step until it reaches $D$. At any time step, the currently-occupied node (i.e., the node that has the probe packet and should relay it) is the state, $s$, where $s \in S = \{n_0, n_1, ..., n_{N-1}\}$. The neighboring nodes form the action space $A$, which is the set of possible actions that the agent can take.

The Q-table is of size $N \times N$, for the total of $N$ states and actions. Yet, some actions cannot be taken if there is no direct link between the corresponding nodes (in this case we set the corresponding Q-value to $-\infty$). Let $S_t = n_i$, and let node $n_i$ have neighbors in the set $NB_i = \{n_1^i, n_2^i ...\}$. Then, we have the feasible set of $A_t \in NB_i$. Also, the actions will be chosen based on which neighboring nodes have not yet been visited along the current path; this prevents cycles from occurring in the path. For each time step $t$ along the route from $S$ to $D$, the state is updated as follows:

$$S_{t+1} \leftarrow \arg\max_a Q(S_t, A_t = a), \tag{1}$$

where $Q(S_t, a)$ represents the value if the agent took action $a$ at state $S_t$. The agent chooses the action that maximizes the Q-value and sets that to $A_t$. When the action is taken, the state will transition to $S_{t+1}$ (which is the next hop). We use the $\epsilon$-greedy algorithm in training the algorithm, where at each step there is a probability $\epsilon$ that the agent will instead visit a neighboring node randomly (and with probability $(1-\epsilon)$ it will use the best known action). This allows the agent to further explore the action space and prevents overfitting in the training process. The random actions are still selected from the neighboring nodes that have not yet been visited on the path.

This process continues until a path has been found from $S$ to $D$. Let $P = \{(S_1 = S, A_1), (S_2 = A_1, A_2), ..., (S_n = A_n - 1, A_n = D)\}$ be the set of the $n + 1$ state-action pairs along the path, where $n$ is the number of hops in the path. Note that the first state $S$ is the source node and the final action $D$ is the sink node. The paths/Q-values are evaluated by the agent and communicated to the source S to know the route it should send over. Alternatively, the path finding, which is obtaining the available paths from $S$ to $D$ and having the Q-table reflect that, can be determined by a controller and communicated to $S$. However, $D$ will update the Q-table based on path reputation and the information will be communicated with S and the controller. There would be alternate secure communication between $S$, $D$, and the centralized controller to exchange control information (for example, cellular or long range communications).

*a) Reward and Q-table update:* For each packet received at the sink, the sink uses the keyed hash function (e.g., HMAC with SHA 256) to detect if the packet is corrupted and updates the path reputation accordingly. Specifically, the reputation of a path $P$ is denoted as $R(P)$ and it represents the total accumulated reward for that path. $R(P)$ can be expressed as:

$$R(P) = \begin{cases} -\rho_{cp} - n & \text{P is corrupted,} \\ \rho_{cp} - n & \text{P is not corrupted,} \end{cases} \tag{2}$$

where $\rho_{cp}$ is a predefined penalty for a corrupted path. Subtracting the number of hops $n$ provides the added benefit of

rewarding shorter paths from $S$ to $D$. For the first transmission, the Q-values of the state-action pairs along a path P will be the path reputation $R(P)$. As more probe packets are transmitted over various paths, the Q-table will be updated using (3) for the state-action pairs involved based on the reputation of multiple paths. The Q-values will eventually reflect the reputation of each state-action. To keep the Q-table normalized, it contains the average penalty incurred when following a path using a state-action pair $(s, a)$. We omit subscript here for simplicity. Let $V(s, a)$ be the number of times that the algorithm has visited $(s, a)$ pair. Each transmission from $S$ to $D$, the Q-values are updated for the corresponding state-action pairs involved in path $P$ as follows:

$$Q(s,a) \leftarrow \frac{Q(s,a) * V(s,a) + R(P)}{V(s,a) + 1}. \tag{3}$$

Note the path $P$ changes for different probe packets. The full algorithm is shown below in Algorithm 1.

**Algorithm 1** Path finding Algorithm from $S$ to $D$

```
Input graph G, source S, sink D, training iterations I
i ← 1;
for m, n in G do
  if m and n are not directly connected in G then
    Q(m, n) ← −∞
  else
    Q(m, n) ← 0
  end if
end for
while i ≤ I do
  s ← S;
  P ← ∅;
  while s ≠ D do
    P ← P ∪ s;
    if s is at a dead end then
      Penalize last (s, a) combination of P in Q-table;
      break
    else
      Select an action a;
      s ← a;
    end if
  end while
  Check if P is corrupted;
  Modify Q-values accordingly;
  i ← i + 1;
end while
```

*Remark on dead-end devices:* In the Q-learning algorithm, it is important to not select actions that would create a cycle in the graph. Therefore, it is possible that a path will reach a point in which there is no valid action to select. We refer to these nodes as dead-end nodes. Let state $s_m$, action $a_n$ specify the connection to a dead-end node. We set the Q-value $Q(s_m, a_m) \leftarrow -\rho_{de}$, where $\rho_{de}$ is a pre-defined penalty assigned to dead end nodes. This ensures that the agent will most likely not take this exact path again in future iterations of the algorithm.

There are two types of dead-end nodes. A node $n_i$ will always be a dead-end node if it only has one link. These nodes do not participate in the relaying process and are ignored from the classification. Alternatively, a node $n_i$ may appear to be a dead-end for a specific path $P$ if its neighbors have already been visited along that path (hence will create a loop if any neighbor is selected).

*b) Identifying malicious nodes:* After training and obtaining a relatively stable Q-table (i.e., values do not change significantly), the Q-values are used to identify compromised nodes. We will show that the Q-values converge relatively fast. In our proposed approach, nodes are classified based on the maximum of their corresponding column in the Q-table. In particular, let the trust value of node $i$ be $T(n_i)$, then $T(n_i) = max_j Q(s = n_j, a = n_i)$. That implies that a node is trusted if it is selected to be the best next hop after being tested on several paths, and the node is not trusted if it operates poorly across the different paths it is part of. Let us define $y(n_i) = 1$ if $n_i$ is malicious, and $y(n_i) = 0$ otherwise. Then

$$\hat{y}(n_i) = \begin{cases} 1 & T(n_i) < -\rho_{th}, \\ 0 & T(n_i) > -\rho_{th}, \end{cases} \tag{4}$$

where $\rho_{th}$ is a pre-defined threshold that can be tuned.

Note that the Q-table can also reveal potential issues in specific links, since $Q(s = n_i, a = n_j)$ represents the value/health of the link from $n_i$ to $n_j$. Hence, in general, the approach improves the network monitoring capability and provides autonomous means for identifying failure(s).

## IV. Simulation and Results

### A. Simulation Setup

Using the Python package `networkx`, random graphs were created for simulation. We constructed 20 graphs for three different network sizes $N$. Specifically, we tested the approach for $N$ equals to 20 nodes, 50 nodes, and 100 nodes. For the purposes of this study, edges directly connecting $S$ and $D$ were not allowed. We set the percentage of malicious nodes to be $\alpha = 30\%$. If a node is malicious, it attacks with a $P_i$ selected from a uniform distribution between [0.2, 0.8]. Note that this is a relatively challenging attack to detect, especially if $P_i$ is low. We run 100,000 path-finding iterations on each graph. For training parameters, we set $\epsilon = 0.1$, and $\gamma = 1$.

A Q-table is initialized for the network, where the rows represent the network's states and the columns represent the actions. All invalid combinations of states and actions (i.e., nodes that do not connect to each other), are set to $-\infty$ to ensure that the RL agent will never select them. Since a path through the network is penalized based on its number of hops, $\rho_{de}$ and $\rho_{cp}$ were defined based on the size of the network. We set $\rho_{th} = \rho_{de} = 2N$ and $\rho_{cp} = 10N$ for simplicity, where $N$ is the number of nodes in the network.

The Q-table's columns can be used to immediately reflect nodes that are dead-ends. If all of the finite Q-values in a column are equal to $\rho_{de}$, the node corresponding to that column is a dead end. This is because the pathfinding process does not create cycles in the graph. Nodes are then classified based on the maximum of their corresponding column in the Q-table, ignoring any state-action combinations in which the node is a dead end. We use (4) and set $\rho_{th} = \rho_{de}$. To assess the performance of the approach we evaluate the precision and recall values for the different network sizes.

### *B. Results*

Table I shows the average precision and recall over the 20 generated networks for each network size. The high values of recall indicate that the algorithm is correctly classifying most of the malicious nodes as such. By contrast, the more middling precision values indicate that the algorithm mis-classifies some benign nodes as malicious. This may be attributed to the harsh penalties assigned to all nodes involved in a corrupted path, encouraging the algorithm to not visit nodes in future iterations, even if those nodes were not malicious. However, the high recall value is encouraging and serves as evidence of the algorithm's potential to identify the malicious nodes of a network.

The increase in average precision and recall as larger networks are simulated should be expected, as the RL agent has more paths from which it can reward and penalize nodes. In less populated networks, some nodes will always be in the same path. With insufficient training iterations or a lack of random exploration by the RL agent, these nodes will be coupled together and equally rewarded or penalized, regardless of their individual statuses.

TABLE I
AVERAGE PRECISION AND RECALL FOR NETWORKS OF DIFFERENT SIZES.

| Nodes in Network | Avg. Precision | Avg. Recall |
|---|---|---|
| 20 | .54 | .73 |
| 50 | .55 | .91 |
| 100 | .62 | .93 |

Figures 3 and 4 show the spread of individual precision and recall values for simulation runs of varying network sizes. Each point corresponds to one simulation run of the algorithm. The variance in both metrics very clearly decreases as larger networks are simulated; however, the difference in variance between 20 nodes and 50 nodes is much larger than between 50 nodes and 100 nodes. These results show that the algorithm will achieve superior accuracy with a narrower confidence interval as network diversity increases. Note that the larger number of nodes here means more routing paths from the sources to the destination.

Figure 5 shows the convergence of Q-table values for a sample network of $N = 50$ nodes over a simulation run. For each batch, which is defined as 1000 training iterations, the total change to the Q-table is accumulated and recorded. The

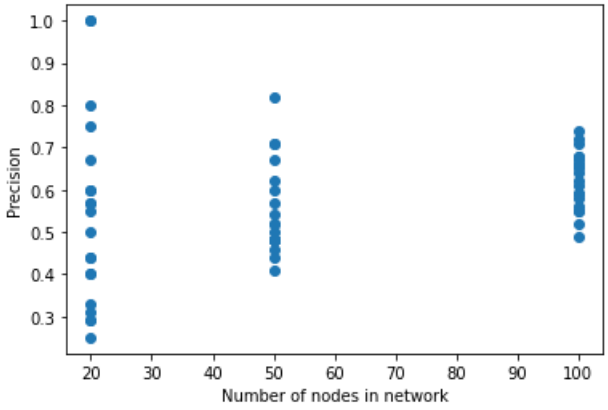


Fig. 3. Variance of precision over simulation runs at different network sizes

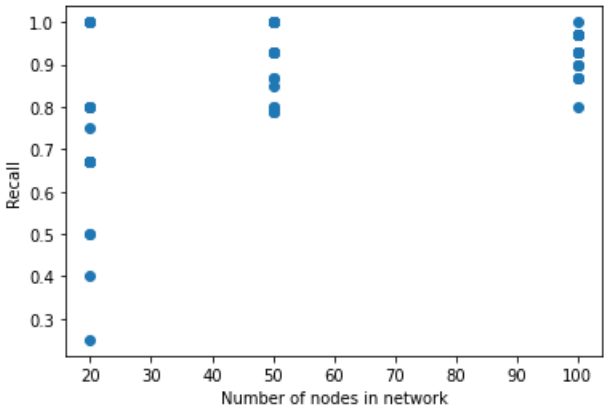


Fig. 4. Variance of recall over simulation runs at different network sizes

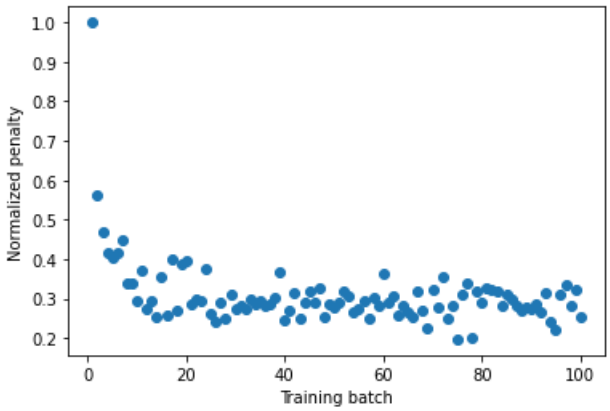


Fig. 5. Convergence plot for a sample network of 50 nodes.

penalties are then normalized for ease of presentation. For this network, most of the changes to the Q-table happened within the first 10 batches, after which the batch penalty stays relatively consistent.

It is noted that, in contrast to our previous work in [9], the proposed Q-learning approach is simpler and provides more information for network monitoring. Specifically, the Q-table

can point out problems in specific links (e.g., $Q(s = n_1, a = n_2)$ represents the value/health of the link from $n_1$ to $n_2$). This helps in identifying failure(s) in specific links or inconsistent node performance across different links. However, in this work the node behaviour is not link dependent. Investigating different attack models is an area of future work.

## V. Conclusion

In this paper, a reinforcement learning approach was utilized to identify malicious IoT devices in multihop networks. A Q-learning algorithm was used to navigate the network and assign rewards or penalties to nodes depending on the end result of the transmission. Simulation results showed that the proposed approach was effective in identifying compromised nodes. It is evident that better performance can be achieved with bigger, more diverse networks, represented by the larger number of paths. We also demonstrated that the Q-table converges relatively fast. Another key advantage of the proposed approach is that the learned Q-table can also provide information about links' operation, which can identify inconsistent node performance across different links. Overall, we showed that reinforcement learning has a great potential in identifying nodes launching routing attacks in a multihop mesh network. It is worth mentioning that the algorithm could be further improved, for example by optimizing the thresholds used for different network sizes, changing the exploration-exploitation strategy in the learning algorithm and/or by applying different RL techniques. This is an area for future work.

## References


[1] "ZigBee Alliance," http://www.zigbee.org/.

[2] "Thread Group," https://www.threadgroup.org/.

[3] C. Wang, T. Feng, J. Kim, G. Wang, and W. Zhang, "Catching packet droppers and modifiers in wireless sensor networks," in *2009 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2009, pp. 1–9.

[4] "HMAC: Keyed-Hashing for Message Authentication," https://datatracker.ietf.org/doc/html/rfc2104.

[5] K. Ioannis, T. Dimitriou, and F. C. Freiling, "Towards intrusion detection in wireless sensor networks," in *Proc. of the 13th European Wireless Conference*. Citeseer, 2007.

[6] A. Tandon, T. J. Lim, and U. Tefek, "Sentinel based malicious relay detection in wireless iot networks," *Journal of Communications and Networks*, vol. 21, no. 5, pp. 458–468, 2019.

[7] M. Abdelhakim, L. Lightfoot, J. Ren, and T. Li, "Reliable communications over multihop networks under routing attacks," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[8] M. Abdelhakim, X. Liu, and P. Krishnamurthy, "Diversity for detecting routing attacks in multihop networks," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, 2018, pp. 712–717.

[9] X. Liu, M. Abdelhakim, P. Krishnamurthy, and D. Tipper, "Identifying malicious nodes in multihop IoT networks using diversity and unsupervised learning," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[10] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, no. 518, p. 529–533, 2015.

[11] X. Liu, M. Abdelhakim, P. Krishnamurthy, and D. Tipper, "Identifying malicious nodes in multihop IoT networks using dual link technologies and unsupervised learning," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 109–125, 2018.

[12] S. Nivetha, N. Senthilkumaran, P. Suguna, and T. Vanaja, "Analysing the performance of software defined network controllers with various topologies," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 2021, pp. 1–5.

[13] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Advances in Neural Information Processing Systems (NeurIPS)*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/5faf461eff3099671ad63c6f3f094f7f-Paper.pdf