Triarchy-Based for DDoS-Resilient IoT Networks

Amir Javadpour*‖, Forough Ja'fari§**, Tarik Taleb¶††, Chafika Benzaïd‡‡‡

*ICTFICIAL Oy, Espoo, Finland
‡Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland
§Department of Computer Engineering, Sharif University of Technology, Iran
¶Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany
‖a.javadpour87@gmail.com (Corresponding Author) **azadeh.mth@gmail.com
††tarik.taleb@rub.de ‡‡chafika.benzaid@oulu.fi

***Abstract*—Identifying cost-effective attack scenarios in large-scale IoT networks from an adversarial perspective is vital for proactive defense planning, since protecting even a subset of nodes can significantly improve overall resilience. However, no dedicated algorithm currently pinpoints the most efficient attack paths under realistic cost constraints. In this paper, we introduce a new graph structure, the *triarchy graph*, to formalize cost-efficient attack-path identification and establish the problem's NP-completeness. Building on this formulation, we propose CRL-MTD, a cost-effective reinforcement learning (RL)-based moving target defense strategy that learns to identify high-impact attack paths and strategically perturbs them to delay adversaries while minimizing operational burden. We evaluate CRL-MTD on real-world SNDlib-based IoT topologies under simulated Mirai botnet attacks. Experimental results show that CRL-MTD improves solution efficiency by 15%, increases attack execution delay by 111%, and reduces system overhead by 90% compared with representative baseline approaches, demonstrating strong practicality for DDoS-resilient IoT deployments.**

I. Introduction

Communication is central to modern technological progress, particularly in automating interactions among non-human entities. The Internet of Things (IoT) has emerged and rapidly expanded, enabling ubiquitous access to interconnected devices [1]. IoT networks often include both Internet-connected devices and isolated ones to enhance privacy and security [2], with isolated devices relying on connected ones as gateways in emergencies [3]. Some nodes, such as patient-monitoring sensors, are critical assets whose compromise can severely disrupt system functionality [4]. Direct Internet exposure increases the risk of compromise, making defense strategies essential.

One approach is delaying adversaries during the botnet formation phase, allowing defenders to analyze tactics, identify compromised nodes, and act. Moving Target Defense (MTD) has been proposed to counter such threats by dynamically changing the network attack surface, rendering previously gathered intelligence ineffective [5], [6], [7].

Identifying the adversary's lowest-cost attack path is key to designing cost-effective MTD. By targeting the most efficient attack scenario, defenders can focus on delaying this path to maximize adversarial cost. Prior work has explored attack path discovery [8], [9], but often overlooks the role of previously compromised hosts in enabling further compromise at no extra cost.

To address this, we propose a novel graph structure called the *triarchy graph*, which effectively models IoT attack propagation. We prove that finding the optimal attack path in such a graph is NP-complete. To approximate this, we introduce a reinforcement learning (RL)-based MTD method, *CRL-MTD* (Cost-effective RL-based MTD), which dynamically learns and perturbs optimal paths [10]. CRL-MTD is evaluated on realistic IoT topologies and simulated Mirai botnet attacks, demonstrating its ability to delay attacks and reduce overhead effectively.

The primary contributions of this paper are summarized as follows:

- We propose the *triarchy graph* to model IoT topology and attack propagation.
- We prove that optimal attack-path discovery on triarchy graphs is NP-complete.
- We develop an actor–critic RL model to learn cost-effective attack paths.
- We design CRL-MTD, an RL-driven moving target defense to disrupt learned paths.
- We validate CRL-MTD on real-world IoT topologies under a realistic botnet setting.

The paper is structured as follows: Section II reviews related work; Section III defines triarchy graphs; Section IV presents problem formulation and complexity; Section V details the proposed RL-based CRL-MTD; Section VI shows evaluation results; and Section VII concludes the paper.

TABLE I: Summary of Related Works on Attack Cost Minimization

Study	Summary and Approach
Agmon et al. [11]	Heuristic placement of IoT devices via tree evaluation to reduce exploitation risk
Gao et al. [12]	Defense path selection using PSO and Q-learning to raise attack cost
Puzis et al. [13]	Introduced fake vulnerabilities; used AI search (DFS, A*) for optimal attack paths
Son and Kim [14]	Applied A* for UAV security in large-scale scenarios with efficient performance
Yoon et al. [8]	Developed BAP heuristic; starts from exposed hosts and builds attack paths based on node criticality
Javadpour et al. [9]	Proposed SCEMA using link-count to protect nodes near critical assets with MTD

II. Related Work

This section reviews existing research on minimizing attack costs within a network.

Agmon et al. [11] proposed a heuristic algorithm that organizes IoT device placements to minimize exploitation risk, using a tree structure to evaluate possible deployments. To raise attack cost, Gao et al. [12] developed a strategy selector combining particle swarm optimization and Q-learning to identify critical paths and apply defense mechanisms.

Puzis et al. [13] introduced fake vulnerabilities to increase attack cost after finding optimal paths using AI-based search (DFS branch-and-bound, A*). Similarly, Son and Kim [14] applied A* to analyze UAV communication security, demonstrating high performance in large-scale networks.

Yoon et al. [8] presented a greedy heuristic, BAP (Backward Attack Path), which starts from externally accessible hosts and builds a path based on vulnerability and criticality, protected using MTD. Javadpour et al. [9] proposed a link-count-based metric to identify hosts most connected to critical assets, and applied an MTD method (SCEMA) focused on those nodes to reduce cost.

However, none of these works consider how already-compromised hosts influence the total attack cost. Addressing this gap is the main focus of our paper.

III. Triarchy Graphs

This section defines triarchy graphs and related concepts. Table II summarizes the notations.

TABLE II: Symbols and descriptions used in triarchy graphs

Symbol	Description
G	Triarchy graph instance
G_z, G_y, G_x	Gateway, connector, and terminal vertex sets
$G_w(i)$	Weight of vertex i (positive integer)
G_e	Edge set of G
G_v	All vertices in G
G^V	Subgraph of G over vertex set V

A triarchy graph is an undirected, vertex-weighted graph $G = (G_z, G_y, G_x, G_w, G_e)$ with disjoint vertex sets (gateways, connectors, terminals). Vertices are non-isolated, connectors lie on paths from gateways to terminals, and only connectors may connect to same-type vertices.

Definition 1. *$G = (G_z, G_y, G_x, G_w, G_e)$ is a triarchy graph if:*

- *G_z, G_y, G_x are disjoint*
- *$G_w(i) \in \mathbb{N}$ for $i \in G_z \cup G_y \cup G_x$*
- *G_e is symmetric; edges only exist between different types or among connectors*
- *Each $i \in G_y$ lies on at least one path: $G_z \to G_y^* \to G_x$*

We define $G_{zy} = G_z \cup G_y$ and $G_{zyx} = G_z \cup G_y \cup G_x$. The shortest path length from a node to a gateway is its *gateway proximity*.

Definition 2. *A **partial path** is a simple path $(p_1, \ldots, p_k)$ where $p_1 \in G_z$, $p_i \in G_y$ for $i > 1$, and $(p_i, p_{i+1}) \in G_e$.*

Definition 3. *A **complete path** is a partial path ending in a terminal: $p_1 \in G_z$, $p_k \in G_x$, intermediates in G_y.*

Definition 4. *Two complete paths P and Q are **distinct** if $p_k \neq q_k$ or $p_{k-1} \neq q_{k-1}$.*

Definition 5. *A **subgraph** G^V of G over $V \subseteq G_{zyx}$ is defined as:*

$$G^V = (G_v^V, G_e^V), \quad \textit{where}$$
$$G_v^V = V \cup G_x, \quad G_e^V = \{(i,j) \in G_e \mid i,j \in G_v^V\}$$

We state a key lemma below.

Lemma 1. *In G, the number of distinct complete paths to a terminal $i \in G_x$ equals $|Q_i|$, where Q_i is the set of its neighbors that lie on some partial path.*

Proof. Let p_i be the number of such paths. Clearly, $p_i \leq |Q_i|$. If $p_i < |Q_i|$, then at least $|Q_i|$ distinct complete paths exist ending at i (each through a different $q \in Q_i$), contradicting the assumption. Hence, $p_i = |Q_i|$. □

IV. Problem Definition

We define the network and threat model and analyze the complexity of the optimization problem under study.

A. Network Model

The IoT network consists of normal and critical devices. Critical devices provide vital services and are protected against compromise but remain vulnerable to DDoS attacks. The adversary initiates infection via Internet-connected devices (G_z), expands the botnet by compromising normal devices (G_y), and finally launches DDoS attacks against critical devices (G_x) once enough bots are gathered. Each normal device has a compromise cost, while each critical device i has a DDoS threshold $G_w(i)$ the number of distinct paths that must be compromised to launch an attack on i.

This behavior is modeled as a triarchy graph $G = (G_z, G_y, G_x, G_w, G_e)$, satisfying:

- Vertex weights are integers (scaled if needed).
- No edges exist between gateways (G_z) or between terminals (G_x).
- No isolated vertices exist.
- Each connector lies on at least one complete path (gateway $\to$ connector(s) $\to$ terminal).

B. Min Flood Problem and Complexity

We define the following optimization problem:

Problem 1 (Min Flood)**.** *Given a triarchy graph G, find a minimum-weight subset of $G_{zy} = G_z \cup G_y$ such that, for every $i \in G_x$, there exist at least $G_w(i)$ distinct complete paths ending at i.*

We show that this problem is NP-complete (Theorem 1). It belongs to NP because a solution can be verified in polynomial time:

- Membership check and total weight verification are polynomial.

- Verifying the existence of required distinct paths (via BFS/DFS on modified graphs) has complexity $O(|G_{zyx}|^3)$.

To show NP-hardness, we reduce from the known NP-complete problem Min Ones 4-SAT [15]:

Problem 2 (Min Ones 4-SAT)**.** *Given a Boolean formula with clauses of 4 literals, find a truth assignment with the fewest variables set to true such that the formula is satisfied.*

We define the decision versions of both problems and construct a polynomial-time reduction:

- We label vertices in G using Boolean expressions derived from their paths (using sum-of-products logic).
- From a solution λ of Min Flood Decision, we build a Boolean expression $F(G)$ such that a corresponding variable set λ' satisfies Min Ones 4-SAT Decision.
- Conversely, from any YES instance of Min Ones 4-SAT, we construct a triarchy graph where selecting gateways based on true variables satisfies the Min Flood Decision.

Theorem 1. *The Min Flood problem is NP-complete.*

Proof. The proof consists of:

1) Showing Min Flood is in NP by verifying paths and weight constraints in polynomial time.
2) Reducing Min Flood Decision to Min Ones 4-SAT Decision by constructing Boolean expressions from graph labels, preserving the solution's validity.
3) Reducing Min Ones 4-SAT Decision to Min Flood Decision by constructing a triarchy graph based on variable-clause structure.

Each step is polynomial in time. Hence, the problem is NP-complete. □

V. Proposed Security Method

Now that we have proved that Min Flood problem is NP-complete, we know that *"finding the set of normal devices, in our network model, leading to the adversary's goal, in our threat model, with the minimum cost"* is NP-complete. Hence, we try to approximately solve it using RL. An actor-critic RL model is proposed to find the optimal attack paths, and then our proposed Cost-effective RL-based MTD approach (CRL-MTD) uses it to mutate the devices in an IoT network effectively. This mutation increases the attack delay, and therefore, the defender has more time to detect and mitigate it.

A. The RL model

The agent in an RL model explores the environment, which indicates the problem that is attempted to be solved, and performs some defined actions. Each action in each environment state results in a specific reward that leads the agent toward the optimal solution. One can assume this process as a game and the agent as its player. The agent tries to obtain the most possible score in each step of this game. In an actor-critic RL model, the actor is the one who decides which action to be taken, and the critic is the one responsible for analyzing the efficiency of that action. Due to the higher scalability of actor-critic models compared to the value or policy-based models, we have considered this type of RL for solving our problem.

We define a game where achieving a higher score corresponds to a more efficient Min Flood solution. In each step, the agent selects a vertex to build backward paths until each terminal satisfies the requirement of $G_w(i)$ distinct complete paths. Let v be the current vertex with gateway proximity n; the agent can choose a neighbor of v with proximity $n-1$. The game starts from a hyper-vertex connected to all terminals and resets to this state upon reaching a gateway. Already selected neighbors of terminals are excluded. As the agent progresses, selected vertices form complete paths, determining the game termination. Since the agent always moves to vertices with lower proximity, infinite loops are avoided. The detailed step logic is given in Algorithm 1.

Algorithm 1 The procedure of each game step in the proposed RL model

Require: A (agent's current action)
Require: a (the agent's history)
if $a.ssz \neq 0$ **then**
  $A \leftarrow (A \ \% \ a.ssz) * 9$
else
  $A \leftarrow 0$
$s \leftarrow$ current state regarding a ▷ Algorithm 2
$a.crnt \leftarrow s[A]$
if $a.src =$ null **then**
  $a.src \leftarrow a.crnt$
$c \leftarrow 0$
for each $v \in a.tnghbr[a.src]$ **do**
  $a.wtrmnl[v] \leftarrow a.wtrmnl[v] - 1$
  if $a.wtrmnl[v] = 0$ **then**
    $c \leftarrow c + 1000$
$r \leftarrow$ reward regarding (A, s, $a.ssz$, c) ▷ Algorithm 3
if $a.ssz = 0$ **or** $a.crnt \in a.gtwy$ **then**
  $a.crnt \leftarrow$ null
  $a.src \leftarrow$ null
else
  $a.aslctd \leftarrow a.aslctd + \{a.crnt\}$
if $a.crnt \in a.gtwy$ **and** $a.src \notin a.tslctd$ **then**
  $a.tslctd \leftarrow a.tslctd + a.src$
Update a based on A and s

To model the game environment, we define nine features for each vertex. Three are binary: being a gateway (f_7), previously selected (f_8), or last in the list (f_9). The remaining features include the vertex ID (f_1), weight (f_2), number of terminal neighbors (f_3), their total weight (f_4), total neighbors (f_5), and total neighbors' weight (f_6). These features collectively guide the agent in selecting optimal paths based on topology, history, and weight considerations.

Algorithm 2 shows how these features are generated and form the environment state.

Rewards in RL are crucial for guiding the agent toward optimal solutions. We define two types of rewards: short-term (step-wise) and long-term (goal-oriented). The long-term reward accounts for two factors: the number of covered terminals and the number of completed paths. Specifically, the agent

Algorithm 2 The procedure of generating the environment states in the proposed RL model

Require: a (the agent's history)
Ensure: $state$ (the current environment state)
$vertices \leftarrow \{\}$
if $a.src =$ null **then**
  for each $v \in a.srcs$ **do**
    if $v \notin a.tslctd$ **then**
      $vertices \leftarrow vertices + \{v\}$
else
  for each $v \in a.nghbr[a.crnt]$ **do**
    if $a.gp[v] = a.gp[a.crnt] - 1$ **then**
      $vertices \leftarrow vertices + \{v\}$
$a.ssz \leftarrow size(vertices)$
$state \leftarrow$ a tuple with $9 \times size(vertices)$ zeros
for $0 \leq v < a.ssz$ **do**
  $v' \leftarrow vertices[v]$
  $state[v * 9] \leftarrow v$
  $state[v * 9 + 1] \leftarrow a.wght[v']$
  $state[v * 9 + 2] \leftarrow size(a.tnghbr[v'])$
  $state[v * 9 + 3] \leftarrow a.twght[v']$
  $state[v * 9 + 4] \leftarrow size(a.nghbr[v'])$
  $state[v * 9 + 5] \leftarrow a.nwght[v']$
  if $v' \in a.gtwy$ **then**
    $state[v * 9 + 6] \leftarrow 1$
  if $v' \in a.aslcd$ **then**
    $state[v * 9 + 7] \leftarrow 1$
  if $v = a.ssz - 1$ **then**
    $state[v * 9 + 8] \leftarrow 1$
return $state$

receives a reward of 100 for completing a valid path and 1000 for successfully covering a terminal, prioritizing terminal coverage due to its higher significance.

Short-term rewards are computed by comparing the selected vertex v with other candidates i in the current list:

- $+5$ if v has a lower weight than i;
- $+10$ if v connects to more terminals than i;
- $+5$ if v connects to higher-weight terminals than i;
- $+1$ if v has more neighbors than i;
- $+2$ if v's neighbors have lower total weight than those of i.

The full reward generation process is outlined in Algorithm 3.

Now that the environment states and the rewards are specified, we can train the RL model based on Algorithm 4.

In this algorithm, the procedure of initiating the agent is called to create the agent. This procedure is described in Algorithm 5.

Moreover, the ending condition of the game can be determined based on Algorithm 6. It must be noted that the game is over once all the terminals are covered or all the terminals' neighbors are examined.

B. The CRL-MTD approach

When the proposed RL model is trained, we can utilized it in an IoT network to effectively mutate the devices and waste the adversary's time. Our proposed MTD method (i.e., CRL-MTD) works based on Software-Defined Networking (SDN)

Algorithm 3 The procedure of generating the reward in the proposed RL model

Require: A (agent's current action)
Require: s (the environment's current state)
Require: ssz (the environment's current state size)
Require: c (the number of covered terminals)
Ensure: r (the agent's reward)
$r \leftarrow 0$
if $ssz \neq 0$ **then**
  for $0 \leq v < ssz$ **do**
    if $s[v * 9 + 1] > s[A + 1]$ **then**
      $r \leftarrow r + 5$
    if $s[v * 9 + 2] < s[A + 2]$ **then**
      $r \leftarrow r + 10$
    if $s[v * 9 + 3] < s[A + 3]$ **then**
      $r \leftarrow r + 5$
    if $s[v * 9 + 4] < s[A + 4]$ **then**
      $r \leftarrow r + 1$
    if $s[v * 9 + 5] > s[A + 5]$ **then**
      $r \leftarrow r + 2$
  if $s[A + 6] = 1$ **then**
    $r \leftarrow r + 1000 \times c + 100$
  if $s[A + 7] = 1$ **then**
    $r \leftarrow r + 10$
return r

Algorithm 4 The procedure of training the proposed RL model

Require: G (the input graph or dataset)
Require: e (the number of training episodes)
Ensure: m (the trained model)
$m \leftarrow$ an instance of RL model
$m.a \leftarrow$ the agent regarding G ▷ Algorithm 5
for $0 \leq i \leq e$ **do**
  $s \leftarrow$ the state regarding $m.a$ ▷ Algorithm 2
  while s is not finished **do** ▷ Algorithm 6
    $A \leftarrow$ the optimal action generated by $m.a$
    Perform a step regarding $m.a$ ▷ Algorithm 1
return m

Algorithm 5 Agent Initialization in the Proposed RL Model

Require: Graph G
Ensure: Initialized agent a
Init a with G_z, G_x, G_w, and empty sets/maps
for each $(v_1, v_2) \in G_e$ **do**
  Append v_2 to $a.nghbr[v_1]$; update $a.nwght[v_1]$
  if $v_2 \in G_x$ **then**
    Append v_2 to $a.tnghbr[v_1]$; update $a.twght[v_1]$
    Add v_1 to $a.srcs$
for each $v \in G_x$ **do**
  $a.wtrmnl[v] \leftarrow G_w(v)$
Initialize $a.gp[v] \leftarrow 0$ if $v \in G_z$ else ∞
for each $src \in G_z$ **do**
  Run Dijkstra-like updates over G_y using $a.nghbr$ and $a.gp$
return a

Algorithm 6 The procedure of determining end of the game in the proposed RL model

Ensure: a (the agent's history)
$covered \leftarrow 0$
for each $v \in a.trmnl$ **do**
  if $a.wtrmnl[v] \geq 0$ **then**
    $covered \leftarrow covered + 1$
$done \leftarrow 0$
if $covered = size(a.trmnl)$ **then**
  $done \leftarrow 1$
else if $size(a.tslctd) = size(a.srcs)$ **then**
  $done \leftarrow 1$
return $done$

concepts, where a central controller manages the traffic flow by coordinating the network switches and the packet-forwarding policies. The controller can communicate with the switches under the OpenFlow protocol. In other words, the controller monitors the network and sets the forwarding rules on the switches, and the switches also inform their status to the controller, all through OpenFlow messages.

Based on CRL-MTD, a trained instance of our proposed RL model resides on the controller. When the network is initiated, the controller discovers the network topology and the devices send their vulnerability exploitation cost (i.e., their weight) to the controller. Then, this information is passed to the RL model. The RL model decides which of the devices are optimal to be mutated. Now, the controller mutates them in specific mutation intervals. To implement this part, the flow rules are set on the switches with a timeout equal to the mutation interval duration. Once an *OFPT_FLOW_REMOVED* message is received, the controller figures out that this is the mutation time. Hence, new IPv6 addresses are assigned to the devices suggested by the RL model, and their related flow entry is updated on the switches. The controller maintains a pool of IPv6 addresses to keep track of the used and new ones.

The complexity order of CRL-MTD depends on the RL model's training complexity. The agent is initiated in $O(\mid G_z \mid \times \mid G_e \mid^2)$ and each running each episode is of $O(\mid G_{zy} \mid^2 \times \mid G_e \mid)$. Hence, the whole training process is of $O(\mid G_{zy} \mid \times \mid G_e \mid \times(\mid G_e \mid + e \mid G_{zy} \mid))$, where e is the number of training episodes. If we consider the upper bound of the number of edges in G, we can say that the time complexity of CRL-MTD is $O(\mid G_{zyx} \mid^5)$.

VI. Evaluation

To evaluate the performance of CRL-MTD, we emulated multiple IoT networks in Mininet using topologies derived from realistic SNDlib datasets [16]. Specifically, the following six datasets were used: *zib54*, consisting of 54 vertices and 81 edges; *norway*, with 27 vertices and 51 edges; *france*, with 25 vertices and 45 edges; *janos-us*, containing 26 vertices and 84 edges; *ta1*, comprising 24 vertices and 55 edges; and *ta2*, which includes 65 vertices and 108 edges. These diverse topologies allow us to evaluate CRL-MTD under various structural conditions and network scales.

We consider 20% of vertices as gateways, 20% as terminals, and the remaining 60% as connectors. To form valid triarchy graphs, we eliminate edges between gateways and between terminals, consistent with our focus on triarchies as sufficient network topologies for problem analysis. Gateway and connector weights are assigned as random integers uniformly distributed in $[1, 10]$, while terminal weights are set to half their topological degree to ensure feasible solutions.

The emulated IoT devices run Ubuntu 18.04 on single-CPU hosts with 2GB RAM, interconnected via Open vSwitches under a POX controller. A separate Internet-based host runs the Mirai botnet [17], which scans the network, compromises vulnerable devices, and launches DDoS attacks from recruited bots targeting critical assets.

To ensure fair evaluation, we conducted multiple emulations and reported average results in terms of efficiency, overhead, and learning ability. We compared our CRL-MTD approach with two credible low-cost MTD methods: SCEMA [9], based on topological degree, and BAP [8], which relies on vertex weight as its main criterion.

CRL-MTD performance is assessed using two metrics: *solution efficiency*, reflecting the optimality of the Min Flood solution generated by the RL model, and *security efficiency*, measuring its protection capability against attacks.

Since the Min Flood problem is NP-complete, computing an exact solution is computationally expensive. Therefore, to evaluate the efficiency of the solution provided by our model, we compare the total weight of selected vertices with that of all vertices. This ratio defines the solution efficiency as formulated in Equation 1.

$$\text{Solution Efficiency}(\lambda) = 100 - \frac{100 \times \sum_{i \in \lambda} G_w(i)}{\sum_{i \in G_{zy}} G_w(i)} \quad (1)$$

The related results are illustrated in Figure 1, which show that the solution efficiency of CRL-MTD is about 15% higher than that of SCEMA and BAP.

After executing the botnet script under CRL-MTD protection, we measured the *attack delay* the time from the first bot recruitment to the final DDoS launch presented in Figure 2. CRL-MTD increases this delay by approximately 111% compared to SCEMA and BAP.

To assess overhead, we measured the *average end-to-end delay*, which grows with the number of mutated devices. This additional delay, considered the *MTD cost*, is shown in Figure 3.

The average results demonstrate that CRL-MTD reduces the extra end-to-end delay cause by the mutations by about 90%, compared to SCEMA and BAP.

Figure 4 illustrates the learning rate of the proposed RL agent trained with different datasets. In all of the trained models, the agent learns how to interact with the environment after about 1000 episodes. The difference between the final score in the stable part is due to the difference between the number of vertices and their degrees in different datasets.

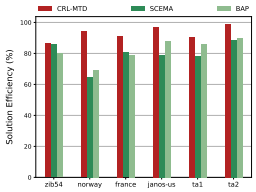

Fig. 1: Comparing CRL-MTD with SCEMA and BAP in terms of solution efficiency.

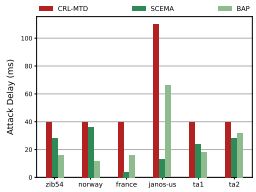

Fig. 2: Comparing CRL-MTD with SCEMA and BAP in terms of attack delay.

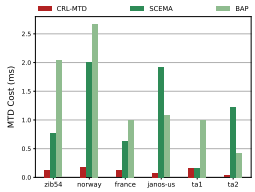

Fig. 3: Comparing CRL-MTD with SCEMA and BAP in terms of MTD cost.

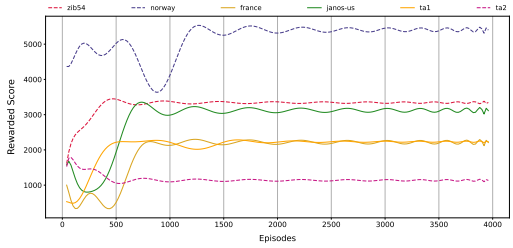

Fig. 4: Comparing the learning ability of the agent trained with different datasets.

VII. Conclusion

To protect the IoT critical assets, we have proposed a low-cost MTD method, called CRL-MTD, in this paper. This method approximately finds the optimal attack path, and then mutates the devices in this path to waste adversary's time. This method is implemented in a software-defined network, the controller of which manages the mutations. To better model the problem, we have introduced a new graph structure, and then it's NP-completeness is proved. Several IoT networks with topologies based on the SNDlib datasets are emulated, and then a Mirai botnet has infected them. The emulation results show that the solution efficiency, the attack delay, and the overhead of CRL-MTD are 15%, 111%, and 90%, respectively, better than that of SCEMA and BAP. We have planned to improve this MTD method by considering a machine-learning model that predicts the optimal mutation intervals.

Acknowledgment

The work in this paper was supported in part by the Federal Ministry of Research, Technology and Space (BMFTR), Germany, through the Project 6GEM+ under Grant 16KIS2409K; and in part by the European Union through the 6G-Path project under Grant 101139172.

References

[1] Y. Qian, D. Wu, W. Bao, and P. Lorenz, "The internet of things for smart cities: Technologies and applications," *IEEE Network*, vol. 33, no. 2, pp. 4–5, 2019.

[2] M. Simpson, S. Woodman, H. Hiden, S. Stein, S. Dowsland, M. Turner, V. L. Hanson, and P. Watson, "A platform for the analysis of qualitative and quantitative data about the built environment and its users," in *2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, 2017, pp. 228–237.

[3] D. Mazzei, G. Baldi, G. Montelisciani, and G. Fantoni, "A full stack for quick prototyping of iot solutions," *Annals of Telecommunications*, vol. 73, pp. 439–449, 2018.

[4] B. Sargunam and S. Anusha, "Iot based mobile medical application for smart insulin regulation," in *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, 2019, pp. 1–5.

[5] J. Landsborough, L. Carpenter, B. Coronado, S. Fugate, K. Ferguson-Walter, and D. Van Bruggen, "Towards self-adaptive cyber deception for defense." in *HICSS*, 2021, pp. 1–10.

[6] A. Javadpour, F. Ja'Fari, T. Taleb, and C. Benzaïd, "Moving target defense for ddos mitigation with shuffling of critical edge (s) connections," *Journal of Information Security and Applications*, vol. 97, p. 104347, 2026.

[7] A. Javadpour, F. Ja'fari, C. Benzaïd, and T. Taleb, "An optimized reinforcement learning based mtd mutation strategy for securing edge iot against ddos attack," *Journal of Information Security and Applications*, vol. 93, p. 104138, 2025.

[8] S. Yoon, J.-H. Cho, D. S. Kim, T. J. Moore, F. Free-Nelson, and H. Lim, "Attack graph-based moving target defense in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1653–1668, 2020.

[9] A. Javadpour, F. Ja'fari, T. Taleb, M. Shojafar, and B. Yang, "Scema: an sdn-oriented cost-effective edge-based mtd approach," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 667–682, 2022.

[10] A. Javadpour, F. Ja'fari, T. Taleb, F. Turkmen, and C. Benzaïd, "Beyond reinforcement learning for network security: A comprehensive survey and tutorial," *Journal of Information Security and Applications*, vol. 96, p. 104294, 2026.

[11] N. Agmon, A. Shabtai, and R. Puzis, "Deployment optimization of iot devices through attack graph analysis," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 192–202.

[12] X. Gao, Y. Zhou, L. Xu, and D. Zhao, "Optimal security protection strategy selection model based on q-learning particle swarm optimization," *Entropy*, vol. 24, no. 12, p. 1727, 2022.

[13] R. Puzis, H. Polad, and B. Shapira, "Attack graph obfuscation," *arXiv preprint arXiv:1903.02601*, 2019.

[14] S. B. Son and D. H. Kim, "Searching for scalable networks in unmanned aerial vehicle infrastructure using spatio-attack course-of-action," *Drones*, vol. 7, no. 4, p. 249, 2023.

[15] N. Misra, N. Narayanaswamy, V. Raman, and B. S. Shankar, "Solving min ones 2-sat as fast as vertex cover," *Theoretical Computer Science*, vol. 506, pp. 115–121, 2013.

[16] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.

[17] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.