

Scalable RL-Based Honeypot Allocation for Slowing IoT Botnet Propagation

Amir Javadpour, Forough Ja'fari and Tarik Taleb

Abstract—Internet of Things (IoT) botnets can expand rapidly across weakly managed devices, making early propagation a useful point for defensive intervention. This paper proposes *Reinforcement Learning-based Honeypot Allocation (RLHA)*, an actor-critic method that formulates honeypot deployment as a finite-horizon, budget-constrained sequential decision problem. RLHA learns an integer-valued allocation map $H : \mathcal{N}_V \setminus \mathcal{N}_C \rightarrow \mathbb{N}_0$ subject to $\sum_i H(i) \leq \delta$, so the policy determines both the number and the locations of decoys. At each decision step, actions are restricted by the remaining budget, while the state combines compromise difficulty, degree, neighborhood vulnerability, reachability, local decoy coverage, and residual budget. For large graphs, clustering reduces the decision set to at most k representative centers and the learned allocations are attached to those centers. The reported implementation is trained with topology-informed shaping and an episode-level cluster-ordering reward; infection rate and DDoS damage are evaluated separately as security outcomes. Small and medium SNDlib connectivity proxies are instantiated in Mininet, whereas large Stanford Gnutella graphs are evaluated through graph-level propagation simulation. In the reported HAAG/AHDS comparisons, RLHA yields lower infection rates and DDoS damage, with a maximum observed relative infection-rate reduction of 41%; this value is a best-case result rather than an average effect. Among seven clustering alternatives, BIRCH provides the most favorable security-runtime balance in the reported experiments. The conclusions are limited to the stated graph proxies, threat model, budget settings, and honeypot-diversion abstraction.

Index Terms—Botnet spread, Malware propagation, Honeypot allocation, Mirai botnet, Reinforcement learning, Clustering algorithm.

I. INTRODUCTION

A. Motivation and Necessity

CONNECTING a massive number of devices together, IoT (Internet of Things) networks are prone to being recruited into botnets, which can trigger major cybersecurity incidents, especially DDoS (distributed denial of service), one of the most frequent and destructive attack categories. Many threat reports have highlighted the urgency of protecting IoT devices from compromise, such as “The number of compromised IoT devices joining botnets to launch DDoS attacks grew by 800,000 from 2022 to the end of June 2023 (Nokia Threat Intelligence Report [1]).”, “The Mirai-based botnets launched more than seven billion DDoS attacks recruiting IoT devices in 2022 (European Union Agency for Cybersecurity, ENISA, Threat Landscape [2]).”, and “A Mirai-variant botnet has launched a DDoS attack with a scale of 1.3 Terabits in the first quarter of 2023 (CloudFlare DDoS Threat Report Q1 [3]).”, to name just a few.

Amir Javadpour is with ICTFICIAL Oy, Espoo, Finland. (e-mail: a.javadpour87@gmail.com).

Forough Ja'fari is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Tarik Taleb is with the Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany

Mirai remains a representative IoT botnet family that demonstrates how rapidly large populations of vulnerable devices can be aggregated and operationalized for DDoS at scale [4]. Similar to the spread of biological viruses, botnets become harder to mitigate as the number of infected devices increases [5]. This motivates mitigation during early-stage propagation, where interventions can still meaningfully delay growth and reduce the eventual bot population.

B. Research Gap

Researchers have mainly used detection, mutation, and deception to slow botnet propagation. Detection methods identify malicious behavior from learned traffic patterns, but they depend on representative training datasets and may not generalize well to unknown or evolving botnets. Mutation or moving target defense techniques can invalidate attacker reconnaissance, yet their deployment cost and operational disruption may be high. Deception techniques, especially honeypots, can lure bots and waste attacker effort without requiring rich labeled datasets [6, 7, 8].

Despite their usefulness, existing deception and honeypot-based defenses still lack a scalable and budget-aware allocation mechanism that can decide where honeypots should be placed to effectively slow propagation in large IoT networks.

RLHA is designed as a budget-aware deception layer for slowing botnet propagation, not as a complete botnet-eradication system. Its goal is to delay reconnaissance and lateral movement within a finite response window, while reducing infection growth and critical-asset degradation. Therefore, RLHA complements defenses such as patching, filtering, authentication hardening, and incident response. The evaluation claims are also limited to graph-connectivity proxies: SNDlib and Stanford Gnutella are used for IoT-scale propagation experiments, not as native IoT traces.

C. Problem Statement and Objectives

We consider an IoT network with limited defensive resources, where only a bounded number of honeypots can be deployed due to cost, management overhead, and operational constraints. The defender must decide where to place honeypots so that botnet propagation is slowed as much as possible while keeping deployment cost low. This placement problem is fundamentally sequential and environment-dependent because the best allocation depends on how propagation unfolds over time and how the attacker traverses the network.

Accordingly, this paper targets the following objectives. The first objective is to learn an allocation strategy that increases propagation resistance, measured through propagation slowdown metrics used in the evaluation. The second objective is to respect cost and budget constraints so the resulting defense remains deployable. The third objective is scalability, meaning the method must remain practical for large topologies without requiring exhaustive search over placements. The fourth objective is data-light operation: the allocation policy should rely on topology and defender-side configuration rather than requiring a labeled traffic dataset for each botnet family.

D. Proposed Approach Overview

In this paper, we propose an RL (reinforcement learning) model called RLHA (reinforcement learning-based honeypot allocation) that

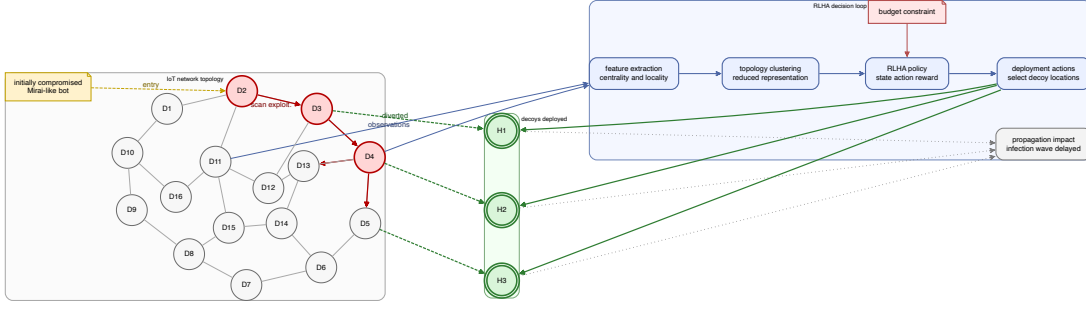


Fig. 1. High-level illustration of RLHA: the defender learns where to deploy honeypots under a limited budget so that botnet propagation is diverted and slowed over the IoT network topology.

seeks an effective strategy for placing honeypots in a network to slow down botnet propagation. Reinforcement learning is a suitable tool for this setting because it directly models sequential decision making under uncertainty and learns policies that maximize long-term reward through interaction with an environment [9, 10, 11]. Using a clustering algorithm, RLHA can be implemented on large-scale networks by reducing the complexity of the decision space while preserving propagation-relevant structure.

E. Novelty and Contributions

The novelty claimed for RLHA is not the individual use of reinforcement learning, honeypots, budget constraints, or clustering. It lies in the decision structure used to connect these elements to botnet-propagation slowdown. Let m denote the number of admissible decision points. A binary placement model chooses whether each point receives a decoy, whereas RLHA chooses a nonnegative integer count $H(i)$ for every point under $\sum_i H(i) \leq \delta$. The number of feasible integer allocations is therefore

$$|\mathcal{H}_{m,\delta}| = \binom{m + \delta}{\delta}, \quad (1)$$

which is not enumerated explicitly. Instead, RLHA constructs an allocation sequentially using $a_t \in \{0, \dots, b_t\}$, where b_t is the remaining budget. Each action changes the residual budget and the local decoy-coverage information observed by later decisions. In the local decoy-coverage information observed by later decisions. In the scalable formulation, clustering also changes the actual decision set from all admissible nodes to at most k representative centers, and the learned integer counts are deployed directly at those centers. Thus, the technical contribution is an integer-allocation, sequential-budget, topology-informed actor-critic formulation with a center-level deployment map, rather than a claim that the constituent techniques are individually new.

The main contributions are as follows.

- 1) A finite-horizon actor-critic formulation for integer-valued honeypot allocation under an explicit deployment budget, with infeasible actions removed by a remaining-budget mask at every decision step.
- 2) A six-component state representation that combines compromise difficulty, degree, neighborhood vulnerability, reachability, local decoy coverage, and remaining budget, together with an explicitly documented feature-shaped training reward.
- 3) A clustering-based center deployment mechanism that bounds the number of sequential decision points on large graphs and permits an ablation across seven clustering methods under the same allocation logic.
- 4) An evaluation with an explicit evidence boundary: Mininet-based experiments on small/medium SNDlib connectivity proxies, graph-level propagation simulation on large Stanford Gnutella proxies, and security claims restricted to the stated Mirai-like propagation and honeypot-diversion model.

F. Illustrative Overview

For an intuitive overview, Figure 1 summarizes the proposed framework. A Mirai-like botnet typically begins from a small set of compromised IoT devices and then expands by scanning and exploiting neighboring nodes. RLHA targets this early propagation stage by learning a budget-aware honeypot allocation policy. Using topology and propagation-related cues, the defender selects a limited set of honeypot locations so that attackers are more likely to interact with decoys than real devices. This diversion wastes attacker effort, slows the infection wavefront, and reduces early botnet growth, while clustering keeps the allocation process feasible on large-scale topologies.

G. Paper Organization

The remainder of this paper is organized as follows. The existing research on slowing down botnets and honeypot allocation is reviewed in Section II. The network model and the behavior of the adversary and the network defender are described in Section III. Section IV explains the proposed RL model and the procedure for making it scalable, along with the required algorithms, and Section V presents performance analysis and related emulation results. Finally, Section VII summarizes the paper and discusses limitations and future work.

II. RELATED WORK

A. Botnet propagation and intervention points

IoT botnets typically follow a progression that starts with initial compromise, continues with reconnaissance and lateral discovery, expands through malware propagation, and culminates in attack execution such as high-volume DDoS. This progression has been repeatedly observed in Mirai-style ecosystems, where large populations of exposed devices with weak credentials or insecure services are recruited and then coordinated for disruptive traffic floods [12, 13]. From a defender perspective, the reconnaissance and propagation stages are particularly important because delaying growth early can reduce the eventual bot population and widen the response window for mitigation and recovery. The literature therefore includes both mechanisms that slow the spread itself and mechanisms that detect early propagation so that containment actions can be triggered before large-scale weaponization Wang et al. [14].

B. Taxonomy of existing defenses for slowing botnets

Prior studies that aim to slow botnet expansion can be grouped into node-centric and network-centric defenses, while detection-oriented work can be seen as an enabling layer for early containment. Node-centric defenses modify or harden individual devices to increase attacker effort or to absorb malicious actions. Network-centric defenses exploit topology, routing, and traffic control to impede scanning, poison attacker knowledge, or divert attacks into decoys. Detection-oriented approaches identify early malicious behaviors and support

rapid quarantine or filtering, but their effectiveness often depends on data availability and generalization to evolving bot families. Table I summarizes these families and the trade-offs that motivate our focus on scalable deception with learning-based allocation.

To make the comparison more informative, the taxonomy in Table I has been extended with three additional evaluation dimensions: implementation effort, scalability, and detection/prevention capability. These dimensions clarify not only what each defense family does, but also how costly it is to deploy, whether it naturally scales from small to large IoT networks, and whether it mainly detects, prevents, delays, or diverts botnet activity.

C. Node-centric approaches

Node-centric defenses aim to slow botnet spread by increasing the time or effort required for compromise and malware execution on individual devices. One line of work exploits implementation weaknesses in bot malware itself by identifying operations that can be forced into slow or blocking behavior. For instance, decoy behaviors can be added to endpoints so that poorly handled network operations in the malware, such as blocking socket reads or timeout handling, incur delays that reduce the effective infection rate Walla and Rossow [19]. Another direction is to expose fake but non-exploitable weaknesses so that bots spend effort on futile exploitation attempts, which can reduce the speed at which the attacker reaches effective attack readiness Hu et al. [20]. Complementary to these, decentralized and collaborative IoT detection has also been proposed, where devices exchange local observations and apply learning models to identify malicious patterns and reduce onward spread Giarretta et al. [21].

D. Network-centric approaches and moving target defenses

Network-centric approaches leverage topology, routing, and control-plane actions to alter attacker visibility or to introduce traps along likely attack paths. Game-based formulations have been used to reason about attacker and defender interactions, for example by introducing deceptive links that can capture an attacker when traversed, with learning used to derive defensive strategies Du et al. [32]. MTD-oriented work includes address and service mutation to invalidate attacker reconnaissance results and force repeated scanning, thereby delaying propagation Luo et al., Wang et al., Javadpour et al. [22, 23, 33]. Other studies combine anomaly detection with adaptive mutation, modeling defense decisions through Markov decision processes to simultaneously slow propagation and reduce attack impact Soussi et al. [25]. While these approaches can be effective, practical deployment must balance mutation cadence against stability and reconfiguration cost, especially in IoT and industrial environments where availability and operational continuity dominate.

In edge-enabled industrial IoT, secure network models have also been proposed to reduce the impact of bot attacks at the network edge. Memos et al. [34] present a secure network model against bot attacks in edge-enabled industrial IoT, emphasizing that defense mechanisms must account for edge placement, industrial service continuity, and constrained devices. RLHA is complementary to this line of work because it focuses on budget-aware deception placement and propagation slowdown rather than only secure network modeling or traffic classification.

E. Early detection and the role of public IoT datasets

A substantial body of research seeks to detect bot activity early so that infected devices can be isolated before botnet growth accelerates. In IoT settings, dataset availability is a recurring bottleneck because device heterogeneity, protocol diversity, and evolving bot families create domain shift between training data and operational environments. Public datasets such as N-BaIoT, Bot-IoT, TON_IoT, and IoT-23 have therefore become common benchmarks for developing and evaluating learning-based detection [15, 16, 17, 18]. These

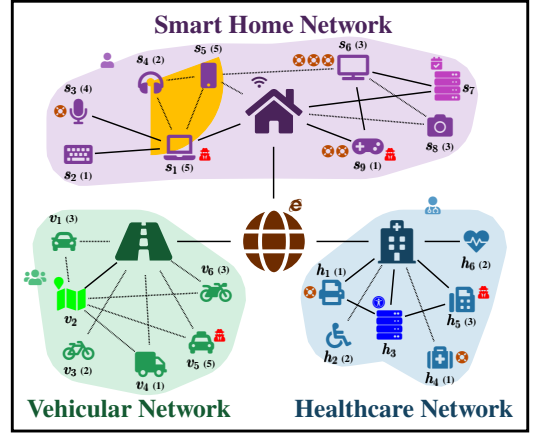


Fig. 2. A sample IoT environment with three local networks.

datasets help standardize evaluation, but data-driven detection still faces challenges in generalizing to new bot variants and to unseen deployment contexts, which motivates complementary defenses that reduce dependence on labeled behavior traces.

Recent ML- and DL-based studies further confirm that data-driven IoT botnet detection remains an active research direction. For example, Kumar et al. [35] investigate machine-learning and deep-learning techniques for securing IoT networks against botnet attacks. Such work strengthens detection capability, but it also reinforces the need for complementary prevention-oriented mechanisms because purely data-driven detection can still suffer from dataset shift, unknown bot variants, and delayed reaction after propagation has already begun.

F. Deception systems and honeypots in IoT practice

Deception has a long history in cyber defense, and honeypots remain a central mechanism because they can both divert attackers and provide high-value telemetry for incident response and threat intelligence [26]. In IoT threat landscapes, honeypots have been used not only for detection but also for studying compromise campaigns at scale, including Telnet-based intrusion waves and multi-architecture malware collection and analysis [13]. Empirical evidence from controlled studies with skilled participants has also shown that decoy systems can measurably alter attacker behavior and impede forward progress, supporting the argument that deception can be an effective delay mechanism rather than merely a passive monitoring tool [27]. At the same time, modern surveys emphasize that attacker adaptation and honeypot detectability are key failure modes, which makes placement, diversity, and realism central to practical effectiveness [36].

G. Honeypot allocation and decoy placement optimization

Beyond isolated honeypot deployment, recent studies examine where decoys should be placed to maximize interception or disruption under limited budgets. Earlier approaches often use strategic games on attack graphs, while newer methods apply scalable optimization, graph learning, and reinforcement learning. Examples include clustering over dynamic attack-graph snapshots [28], graph-based placement prediction from topology and attacker behavior [29], decoy-type selection using attacker technique knowledge bases [30], and multi-agent or hierarchical RL for coordinated allocation [31].

Although these studies advance decoy placement, most focus on interception, attack-path coverage, or decoy selection rather than IoT botnet propagation slowdown. RLHA addresses this gap by learning a budget-aware integer honeypot allocation that directly targets propagation dynamics while using clustering to keep the decision process scalable.

TABLE I
TAXONOMY OF DEFENSES RELEVANT TO SLOWING IoT BOTNET PROPAGATION, TOGETHER WITH THEIR PRACTICAL TRADE-OFFS AND OPERATIONAL CHARACTERISTICS

Defense Category			Practical Assessment		Operational Characteristics		
Family	Deployment locus	Typical lever	Strengths	Limitations	Cost / effort	Scalability	Detection / prevention capability
Early detection and containment [15, 16, 17, 18]	sensors, gateways, monitors	anomaly or signature detection, quarantine, filtering	can suppress propagation if triggered early; integrates naturally with monitoring and SOC workflows	depends on representative datasets; suffers from domain shift and weak coverage for unknown or evolving bot behaviors	medium to high; requires data pipelines, feature engineering, and model maintenance	medium; gateway deployment scales reasonably well but performance is traffic-sensitive	mainly detection and containment after suspicious behavior emerges
Node-centric slowdown Walla et al., Giaretta et al. [19, 20, 21]	endpoint devices	hardening, deception hooks, fake vulnerabilities, local detection	directly increases attacker effort at compromise and infection stages	heterogeneous firmware, limited device resources, and operational overhead complicate deployment	high; often requires device-level modification or endpoint support	low to medium; difficult to deploy consistently across heterogeneous IoT fleets	prevention and slowdown at individual devices
Network-centric containment and MTD Luo et al., Wang et al., Osei et al., Soussi et al. [22, 23, 24, 25]	network control plane	address mutation, service randomization, traffic redirection	reduces attacker certainty and disrupts scanning and exploitation paths	reconfiguration overhead, stability constraints, and mutation-cadence tuning may limit practicality	medium to high; requires control-plane integration and co-ordination	medium; scalable when SDN or programmable network control is available	prevention, disruption, and containment
Deception and honeypots [26, 27, 13]	decoy assets and instrumentation	luring and engagement to waste attacker effort, plus intelligence collection	less dependent on labeled datasets; useful even against evolving or previously unseen tactics	effectiveness depends on placement quality, realism of decoys, and attacker adaptation	low to medium; depends on decoy realism, maintenance, and deployment design	medium; scalable when placement and management are automated	diversion, slowdown, telemetry, and partial prevention
Decoy placement optimization [28, 29, 30, 31]	attack graphs and network topology	game-theoretic, optimization, RL, or graph-learning-based placement	provides principled allocation under budget constraints and can address structurally complex graphs	may face scalability issues, dynamic-topology challenges, and dependence on actionable state representations	medium; depends on graph construction quality and optimization cost	medium to high; improves with clustering or learning-based abstractions	mostly prevention/diversion through improved placement
This work (RLHA)	IoT topology and deployment budget	RL-based honeypot allocation with clustering for scalable placement	low-cost, data-light, and directly focused on propagation slowdown under explicit budget constraints	depends on faithful propagation modeling and realistic deployment assumptions	low to medium; requires topology abstraction and deployable decoys	high; decision points are bounded by the number of clusters	prevention-oriented slowdown through budget-aware deception

H. Industry guidance and operational constraints

Real IoT and industrial IoT deployments are constrained by long device lifecycles, limited patchability, heterogeneous firmware, and strict availability requirements. Therefore, standards and guidance emphasize secure default configurations, vulnerability disclosure, update mechanisms, logging, and defense-in-depth risk management [37, 38, 39, 40]. These constraints support the need for low-cost, topology-aware deception layers that can be deployed at the network level without requiring firmware changes on every device.

I. Positioning of this work

Existing approaches for slowing botnets through detection and MTD face practical limitations such as dataset dependence, reconfiguration cost, and sensitivity to unknown bot variants. Deception-based defenses reduce reliance on labeled botnet traces and can directly increase attacker effort through engagement, but their effectiveness depends critically on where decoys are placed and how they scale to large topologies. While attack-graph allocation and learning-based placement have advanced substantially, there remains a gap for a scalable, budget-aware method explicitly designed to slow IoT botnet propagation. RLHA is developed to address this gap by learning an allocation policy tied to propagation slowdown while using clustering to keep decisions feasible on large IoT networks.

J. Technical distinction from prior decoy-placement work

The closest methodological families include attack-graph decoy placement, centrality-driven placement, graph-learning placement, and hierarchical or multi-agent allocation. RLHA does not claim to supersede these families in general. Its technical distinction is the

object and constraint structure of the decision process. The learned allocation is

$$H : \mathcal{N}_V \setminus \mathcal{N}_C \rightarrow \mathbb{N}_0, \quad \sum_{i \in \mathcal{N}_V \setminus \mathcal{N}_C} H(i) \leq \delta, \quad (2)$$

so the action specifies how many decoys are attached to a decision point, not only whether a decoy is present. For a node i , multiple attached decoys alter the diversion probability and expected engagement delay, while the remaining budget constrains every subsequent action. Moreover, the state evolves with the allocation because later decisions observe a reduced budget and updated local decoy-coverage information. In the scalable variant, clustering defines the actual feasible decision set $\{c_1, \dots, c_K\}$ and the final center-attached deployment rather than acting only as a visualization or feature-preprocessing step. These properties give RLHA a propagation-oriented, integer-allocation, sequential-budget structure. Direct empirical superiority is claimed only against methods for which comparable results are actually reported in Sec. V.

K. Detailed comparative interpretation of Tables I and II

The extended comparison highlights three practical advantages of RLHA. First, compared with detection-centered studies, including ML/DL-based IoT botnet detection, RLHA does not require labeled traces of each botnet family before it can affect propagation. Detection methods are valuable for identifying suspicious behavior, but their reaction is usually triggered after traffic evidence appears. RLHA instead acts as a prevention-oriented slowdown layer by placing decoys before or during early propagation. Second, compared with node-centric hardening and endpoint-level deception, RLHA avoids requiring firmware-level modification of every heterogeneous IoT device. The method operates through a topology-level allocation

TABLE II
REPRESENTATIVE STUDIES RELATED TO BOTNET SLOWDOWN, DECEPTION, AND HONEYPOT ALLOCATION

Reference	Scope	Primary goal	Recon	Prop	Attack	Deception	Learning or game	Scalability note
Walla and Rossow [19]	node	slowdown	✓	✓		✓		endpoint-focused
Hu et al. [20]	node	slowdown			✓	✓		endpoint-focused
Giaretta et al. [21]	IoT	detection	✓	✓			✓	decentralized sensing
Du et al. [32]	network	capture and delay	✓			✓	✓	topology-based trapping
Luo et al. [22]	IoT	delay scanning	✓	✓		✓		uses mutation and decoys
Wang et al. [23]	network	delay scanning	✓	✓		✓		redirect before mutation
Soussi et al. [25]	5G and IoT	slowdown and mitigate		✓	✓		✓	demo framework
Ngo et al. [28]	enterprise	placement	✓	✓	✓	✓	✓	clustering over dynamic graphs
Osman et al. [29]	general	placement	✓	✓	✓	✓	✓	graph learning for placement
Zambianco et al. [30]	enterprise	decoy selection	✓	✓	✓	✓		graph optimization using ATT&CK
Hoang et al. [31]	general	allocation	✓	✓	✓	✓	✓	hierarchical multi-agent RL
[35]	IoT	detection	✓	✓	✓		✓	ML/DL detection; no explicit honeypot allocation
[34]	edge IIoT	secure network model	✓	✓	✓			edge-enabled industrial setting; prevention-oriented architecture
This work	IoT	slowdown and placement	✓	✓		✓	✓	clustering for large topologies

map and therefore better matches environments in which full endpoint control is unrealistic. Third, compared with general decoy-placement and attack-graph optimization methods, RLHA explicitly learns an integer-valued allocation under a remaining-budget state. This allows one decision point to receive multiple honeypots when that point is structurally important, instead of treating placement only as a binary decision.

RLHA is intended as a scalable, budget-aware deception layer for slowing botnet propagation rather than replacing intrusion detection, secure edge architectures, or moving-target defense. As summarized in Table I, it balances deployment cost, scalability, and prevention capability. Its center-based clustering further reduces allocation complexity on large graphs while retaining the propagation-delay benefits of deception.

III. NETWORK AND THREAT MODELS

A. IoT network as a vertex-weighted graph

We model an IoT network as an undirected vertex-weighted graph, where vertices and edges represent IoT devices and their connectivity, respectively. Devices may communicate through wired or wireless links. A wired link between two endpoints is represented as an edge. For wireless connectivity, the neighbor set of a device equipped with a wireless interface is determined by signal strength and effective coverage, which induces additional edges between devices that can effectively communicate. In the sample IoT environment shown in Figure 2, the coverage area of s_4 is highlighted in orange to illustrate how wireless reachability induces edges.

The signal of s_4 covers s_1 and s_5 , creating corresponding graph edges that represent feasible probing paths during botnet propagation. While Figure 2 provides the physical-network view, Figure 3 summarizes the threat model by showing Internet-connected devices, initial bots, critical assets, honeypots, and wireless links.

B. Internet reachability and propagation entry points

Some IoT devices, such as s_1 , are directly Internet-connected, while others, such as s_6 , rely on Internet-connected relays. Consequently, Internet-originated attacks are expected to enter the local network through reachable devices and propagate toward non-Internet-connected nodes through compromised intermediate devices, as illustrated in Figure 3.

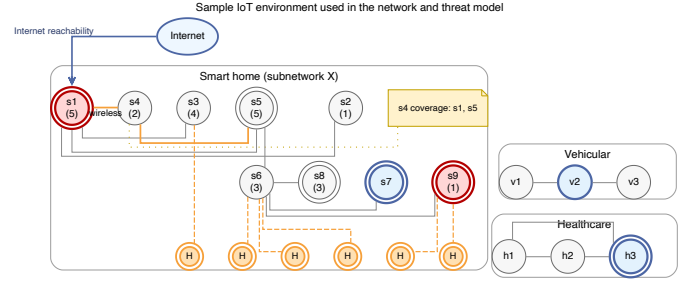


Fig. 3. A structured sample IoT environment with three local networks and the visual encoding used in our network and threat model. Internet-connected devices are marked with a double boundary, initial bots are highlighted in red, critical assets are highlighted in light colors, honeypots are shown as orange decoys attached to non-critical nodes, and wireless reachability induced by s_4 is indicated by orange links.

C. Critical assets and service dependency

In an IoT network, there may be critical assets serving key services, such as vital signs monitoring in healthcare networks [41], providing location and meter data in vehicular networks [42], and synchronization in smart home networks [43]. Adversaries are enticed to attack these assets because degrading them can undermine the whole network. The critical assets in Figure 3, shown in light colors, are s_7 , v_2 , and h_3 .

D. Vulnerability severity and compromise-time weights

Each device may contain vulnerabilities whose severity can be assessed using CVSS (Common Vulnerability Scoring System) [44]. We abstract this exposure through an integer compromise-time weight: lower values indicate devices that are easier and faster to compromise, while higher values indicate more resistant devices. The exact numeric value is less important than the relative ordering among devices. We also assume that critical assets are protected from recruitment as bots and can only be degraded by distributed attacks such as DDoS. The compromise-time values in the illustrative network are shown in parentheses in Figure 3.

For clarity, the compromise-time weight used in RLHA is an abstract relative difficulty indicator rather than a direct transformation of the CVSS base score. In practice, such a weight may be estimated

from vulnerability exploitability, service exposure, patch status, configuration weakness, historical security evidence, device exposure to the Internet, or the presence of compensating controls such as access control and monitoring. Lower values indicate devices that are easier to compromise and therefore more likely to accelerate early botnet growth, whereas higher values indicate devices that are harder to compromise and less likely to become early propagation accelerators. This interpretation is used in the evaluation and avoids assuming a one-to-one mapping between CVSS severity and compromise time.

E. Formal network model and illustrative subnetwork

Accordingly, we model an IoT network as a quintuple $\mathcal{N} = (\mathcal{N}_V, \mathcal{N}_E, \mathcal{N}_C, \mathcal{N}_I, \mathcal{N}_T)$, where $\mathcal{N}_V$ is the set of devices, $\mathcal{N}_E$ is the set of direct links shown as pairs of vertices with $\mathcal{N}_E \subseteq \{(i, j) \mid i, j \in \mathcal{N}_V\}$, $\mathcal{N}_C$ is the set of critical assets, $\mathcal{N}_I$ is the set of Internet-connected devices with $\mathcal{N}_C, \mathcal{N}_I \subseteq \mathcal{N}_V$, and $\mathcal{N}_T$ is the set of pairs indicating the devices' compromising time. Specifically, $\mathcal{N}_T = \{(i, j) \mid i \in \mathcal{N}_V \cap \mathcal{N}_C, j \in \mathbb{N}\}$.

The sample environment in Figure 2 and Figure 3 contains multiple local networks. For clarity, we denote the sample smart home subnetwork by $\mathcal{X}$ and list its elements explicitly. If we call the sample smart home network shown in Figure 3 as $\mathcal{X}$, we have:

$$\begin{aligned}\mathcal{X}_V &= \{s_1, s_2, \dots, s_9\}, \\ \mathcal{X}_E &= \{(s_1, s_2), (s_1, s_3), (s_1, s_4), (s_1, s_5), (s_4, s_5), \\ &\quad (s_5, s_6), (s_6, s_7), (s_6, s_8), (s_6, s_9)\}, \\ \mathcal{X}_C &= \{s_7\}, \\ \mathcal{X}_I &= \{s_1, s_5, s_7, s_8, s_9\}, \\ \mathcal{X}_T &= \{(s_1, 5), (s_2, 1), (s_3, 4), (s_4, 2), (s_5, 5), (s_6, 3), (s_8, 3), (s_9, 1)\}.\end{aligned}$$

F. Defender model and honeypot interaction abstraction

The network defender in our security model has complete knowledge of the network, meaning $\mathcal{N}$ is known, and deploys at most δ honeypots. Each honeypot is directly connected to a device that is not considered to be a critical asset. Furthermore, the wasted time each honeypot causes is a fixed value, say σ . Consequently, the defender of $\mathcal{N}$ is modeled as a triple $\mathcal{D}^{\mathcal{N}} = (\mathcal{D}_\delta^{\mathcal{N}}, \mathcal{D}_\sigma^{\mathcal{N}}, \mathcal{D}_H^{\mathcal{N}})$, where $\mathcal{D}_\delta^{\mathcal{N}}$ is the maximum number of honeypots the defender can deploy, $\mathcal{D}_\sigma^{\mathcal{N}}$ is the time a single honeypot can waste, and $\mathcal{D}_H^{\mathcal{N}}$ is the set of pairs that indicates how many honeypots are assigned to each device. In other words, $\mathcal{D}_H^{\mathcal{N}} = \{(i, j) \mid i \in \mathcal{N}_V \cap \mathcal{N}_C, j \in \mathbb{N}_\delta\}$, where $\mathbb{N}_\delta$ is the set of natural numbers not higher than δ .

To capture how honeypots introduce delay, we assume that when a bot probes a target device i , the connection attempt is resolved uniformly at random over the set consisting of i and the honeypots attached to i . If the attempt lands on a honeypot, the bot wastes σ time units and does not compromise a real device. Otherwise, the attempt targets the real device and the compromise process proceeds according to the device compromise-time weight in $\mathcal{N}_T$. This abstraction models the intended effect of deception as engagement delay while keeping the propagation process analytically and algorithmically tractable.

For network $\mathcal{X}$ in Figure 3, where honeypots are shown as orange decoys, we have:

$$\begin{aligned}\mathcal{D}_\delta^{\mathcal{X}} &= 7, \\ \mathcal{D}_\sigma^{\mathcal{X}} &= 1, \\ \mathcal{D}_H^{\mathcal{X}} &= \{(s_1, 0), (s_2, 0), (s_3, 1), (s_4, 0), (s_5, 0), (s_6, 3), (s_8, 0), (s_9, 2)\}.\end{aligned}$$

G. Threat-model scope and deployment assumptions

The proposed threat model intentionally abstracts several operational details in order to isolate the honeypot-allocation problem. First, the assumption that the defender has complete knowledge of $\mathcal{N}$ is most appropriate for managed IoT, enterprise IoT, industrial IoT, or campus IoT deployments where asset inventories, monitoring systems, or SDN controllers provide a reasonably accurate network

view. It is less appropriate for unmanaged consumer IoT settings, where devices may join and leave without central registration. In such settings, RLHA should be combined with topology discovery or uncertainty-aware learning.

Second, attaching a honeypot to a non-critical device is a deployment abstraction. In practice, this can correspond to SDN-based redirection, address aliasing, gateway-level forwarding, service-level decoy exposure, or virtualized emulation that makes the decoy appear reachable through the protected location. It does not require physically adding a new device beside every protected endpoint.

Third, the uniform diversion assumption is used as a baseline tractable model. If $H(i)$ honeypots are associated with node i , the baseline diversion probability is $\rho(i) = H(i)/(H(i) + 1)$. More realistic deployments can use

$$\rho(i) = \frac{\eta_i H(i)}{1 + \eta_i H(i)}, \quad 0 \leq \eta_i \leq 1, \quad (3)$$

where η_i captures decoy realism, routing effectiveness, and the attacker's ability to fingerprint honeypots. The present experiments use the baseline abstraction, while sensitivity to lower diversion effectiveness is left for future work.

Fourth, critical assets are modeled as hardened or monitored services that can be degraded by DDoS but do not become bots. If critical assets can also be compromised and recruited, the propagation model must be extended with an additional compromised-critical state. The claims in this paper should therefore be interpreted under the stated non-recruitment assumption for critical assets.

H. Parameterized robustness dimensions

To make violations of the baseline assumptions explicit, we define four perturbation dimensions. First, under partial topology knowledge the defender acts on an observed edge set $\hat{\mathcal{N}}_E \subseteq \mathcal{N}_E$. A simple visibility index is

$$q = \frac{|\hat{\mathcal{N}}_E \cap \mathcal{N}_E|}{|\hat{\mathcal{N}}_E|}, \quad 0 \leq q \leq 1, \quad (4)$$

where $q = 1$ recovers the baseline complete-view assumption. Edge deletion can be random or biased toward unmonitored segments to test non-random missingness.

Second, constrained deployment is represented by an admissible set $\mathcal{F} \subseteq \mathcal{N}_V \setminus \mathcal{N}_C$. If $i \notin \mathcal{F}$, the only feasible action at that location is $H(i) = 0$. This converts administrative, physical, or SDN-control limitations into an explicit action mask rather than assuming that every non-critical node is deployable.

Third, decoy effectiveness is generalized from the uniform baseline to

$$\rho_i(t) = \frac{\eta_i(t) H(i)}{1 + \eta_i(t) H(i)}, \quad 0 \leq \eta_i(t) \leq 1, \quad (5)$$

where $\eta_i(t)$ jointly captures redirection reliability and the probability that a decoy remains believable to the attacker. A decreasing $\eta_i(t)$ can model fingerprinting or adaptation after repeated interactions.

Fourth, if a critical asset can be recruited rather than only degraded, its state must be extended, for example, to $z_i \in \{\text{clean, compromised, degraded}\}$ and the propagation process must permit transitions into the compromised state. This changes both the bot population and the damage calculation and is therefore not equivalent to the baseline assumption.

These perturbations define concrete robustness experiments, but they are not silently treated as completed validations. Unless a result is explicitly reported in Sec. V, the empirical conclusions of this paper refer to the baseline model ($q = 1$, unconstrained non-critical attachment, stationary diversion effectiveness, and non-recruitable critical assets).

TABLE III
KEY PARAMETERS IN THE ATTACKER AND DEFENDER MODELS

Symbol	Meaning
δ	Maximum number of honeypots deployable by the defender
σ	Wasted time incurred when a probe is diverted to a honeypot
λ	Probe budget per time unit for each bot
γ	Time horizon until the botnet triggers the distributed attack stage

I. Adversary model and propagation dynamics

We consider botnet growth followed by a distributed attack stage, with the adversary potentially located either inside or outside the local network. A subset of devices is initially compromised as bots, regardless of the initial infection mechanism.

Once compromised, each bot scans and probes neighboring devices, performing up to λ probes per time unit. Propagation is therefore constrained by network connectivity: a device can be compromised only if a path of infected nodes exists from an initially infected bot to that device. Infection may occur directly or through a third-party malware loader.

The adversary commands the bots to launch distributed attacks against accessible critical assets after γ time units from the beginning of the botnet spread. We assume that the adversary knows at least one critical asset in the network. We represent the attacker model as a quadruple $\mathcal{A}^N = (\mathcal{A}_\lambda^N, \mathcal{A}_\gamma^N, \mathcal{A}_B^N, \mathcal{A}_C^N)$, where $\mathcal{A}_\lambda^N$ is the number of probes in each time unit by a single bot, $\mathcal{A}_\gamma^N$ is the duration of the botnet lifecycle, $\mathcal{A}_B^N$ is the set of initially infected bots with $\mathcal{A}_B^N \subseteq \mathcal{N}_V \cap \mathcal{N}_C$, and $\mathcal{A}_C^N$ is the set of critical assets that the adversary is aware of with $\mathcal{A}_C^N \subseteq \mathcal{N}_C$.

Considering that the initially infected bots are shown as red nodes in Figure 3, we have:

$$\begin{aligned}\mathcal{A}_\lambda^N &= 3, \\ \mathcal{A}_\gamma^N &= 30, \\ \mathcal{A}_B^N &= \{s_1, s_9\}, \\ \mathcal{A}_C^N &= \{s_7\}.\end{aligned}$$

J. Parameter summary

For ease of interpretation, Table III summarizes the key attacker and defender parameters used in the threat model.

K. Notation summary

The notations used in this paper for modeling the network, the defender, and the adversary are presented in Table IV.

IV. PROPOSED SECURITY METHOD

The main problem this paper tries to solve is two-fold: "How many honeypots are optimal, in terms of cost and security, to be deployed in a network?" and "Where must the honeypots be located in a network to achieve the optimized security?". This section presents our proposed actor-critic RL model, called RLHA (RL-based Honeypot Allocation), for solving these two questions and a clustering-based technique to make it scalable.

A. Problem formulation and objectives

Let $\mathcal{N} = (\mathcal{N}_V, \mathcal{N}_E, \mathcal{N}_C, \mathcal{N}_I, \mathcal{N}_T)$ denote the IoT network defined in Section III. The defender selects an allocation map $H : \mathcal{N}_V \setminus \mathcal{N}_C \rightarrow \mathbb{N}_0$, where $H(i)$ denotes the number of honeypots attached to device i . The allocation must satisfy the budget and feasibility constraints

$$\sum_{i \in \mathcal{N}_V \setminus \mathcal{N}_C} H(i) \leq \delta, \quad H(i) = 0 \quad \forall i \in \mathcal{N}_C. \quad (6)$$

TABLE IV
THE NOTATIONS USED FOR REPRESENTING THE NETWORK, THE DEFENDER, AND THE ADVERSARY

Notation	Description
$\mathcal{N}$	The IoT network
$\mathcal{N}_V$	The set of devices in $\mathcal{N}$
$\mathcal{N}_E$	The set of pairs indicating the links in $\mathcal{N}$
$\mathcal{N}_C$	The set of critical assets in $\mathcal{N}$
$\mathcal{N}_I$	The set of Internet-connected devices in $\mathcal{N}$
$\mathcal{N}_T$	The set of pairs indicating the compromising time of the nodes in $\mathcal{N}$
$\mathcal{D}^N$	The defender of $\mathcal{N}$
$\mathcal{D}_\delta^N$	The maximum number of honeypots that can be deployed by $\mathcal{D}^N$
$\mathcal{D}_\sigma^N$	The time a single honeypot deployed by $\mathcal{D}^N$ can waste
$\mathcal{D}_H^N$	The set of pairs indicating the number of honeypots assigned to each node of $\mathcal{D}^N$'s network
$\mathcal{A}^N$	The adversary attacking $\mathcal{N}$
$\mathcal{A}_\lambda^N$	The number of probes a single bot controlled by $\mathcal{A}^N$ performs during a single time unit
$\mathcal{A}_\gamma^N$	The duration of $\mathcal{A}^N$'s botnet lifecycle
$\mathcal{A}_B^N$	The set of initially infected bots by $\mathcal{A}^N$
$\mathcal{A}_C^N$	The set of critical assets $\mathcal{A}^N$ is aware of

Our security objective is to obtain a budget-feasible allocation that slows botnet propagation, while the implemented RL training objective is a topology-informed proxy for that security goal. Specifically, the actor-critic agent is trained using the feature-shaped local reward and, in the clustered setting, the episode-level cluster-ordering reward defined in Algorithm 5. Infection rate and DDoS damage are then used to evaluate whether the learned allocation achieves the intended security effect. This separation prevents the training objective from being described as if it directly optimized propagation outcomes when those outcomes are not executed inside the reported training loop.

For a presentation-oriented overview, Figure 4 summarizes RLHA as a step-by-step network story. The left column lists the main stages of the method, while the right column shows a consistent network snapshot in each stage to illustrate what changes on the topology as the method progresses, from feature extraction and clustering to center-based allocation and final deployment of decoys.

B. MDP definition for RLHA

RLHA is formulated as a finite-horizon Markov decision process (MDP) in which the agent allocates honeypots sequentially across nodes. At step t , the agent observes a state vector s_t that summarizes the current node and the remaining budget, selects an integer action a_t representing the number of honeypots assigned to that node, receives a reward r_t , and moves to the next node deterministically. The action space is

$$a_t \in \{0, 1, \dots, \delta\}, \quad (7)$$

with feasibility enforced by the remaining budget at that step as implemented in Algorithm 5. The transition is independent of the chosen action in the sense that the next decision point advances to the next node index (or next center when clustering is enabled). An episode terminates when either the deployed honeypots reach the budget δ or all decision points have been visited, consistent with Algorithm 4. The immediate reward corresponds to the short-term shaping rules, and the episode-level reward implements the cluster-aware ordering objective described in subsection IV-E.

The next decision point advances deterministically according to the predefined node or center ordering. However, the full MDP state transition is action-dependent because each selected action changes the remaining budget and can change the neighborhood-coverage feature used in later states. Therefore, RLHA has a deterministic ordering process but action-dependent budget and local-coverage updates.

RLHA step-by-step (clean): method chart with network intuition

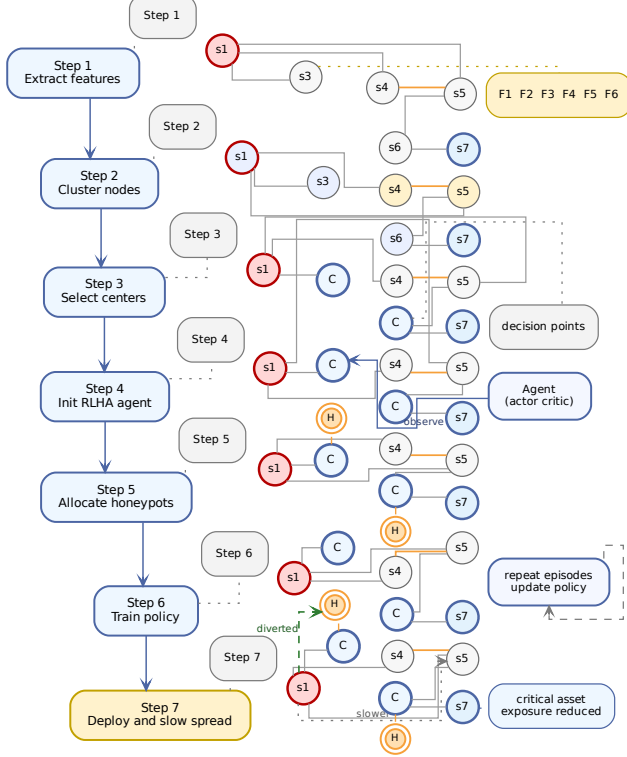


Fig. 4. In each stage, the network snapshot highlights the corresponding concept: feature extraction from topology-aware signals, clustering for scalability, selecting cluster centers as decision points, initializing the actor-critic agent, allocating honeypots to centers under the budget constraint, training over repeated episodes, and deploying decoys that divert probing attempts and slow propagation toward critical assets.

C. Implemented actor-critic objective and security-metric separation

Let $\pi_\theta(a_t | s_t)$ denote the actor, $V_\phi(s_t)$ the critic, and b_t the remaining deployment budget before step t :

$$b_t = \delta - \sum_{\tau < t} a_\tau, \quad \mathcal{A}_t^{\text{feas}} = \{0, 1, \dots, b_t\}. \quad (8)$$

The actor is sampled only over $\mathcal{A}_t^{\text{feas}}$. Hence, the -1000 infeasibility branch retained in Algorithm 5 is a defensive guard and is not reached under the normal masked-policy execution.

To make the training signal traceable to Algorithm 5, write the normalized state as $s_t = (s_{t,0}, \dots, s_{t,5})$, let τ denote the learning threshold, and define $p_t = a_t/b_t$ for $b_t > 0$ (and $p_t = 0$ when $b_t = 0$). For a feasible action, the implemented local shaping reward is

$$\begin{aligned} r_t^{\text{loc}} = & -a_t + 2 \mathbf{1}(|p_t - (1 - s_{t,0})| \leq \tau) + 10 \mathbf{1}(|p_t - s_{t,1}| \leq \tau) \\ & + 5 \mathbf{1}(|p_t - (1 - s_{t,2})| \leq \tau) + 10 \mathbf{1}(|p_t - (1 - s_{t,3})| \leq \tau) \\ & + \mathbf{1}(|p_t - (1 - s_{t,4})| \leq \tau), \end{aligned} \quad (9)$$

where $\mathbf{1}(\cdot)$ is the indicator function. At episode termination, Algorithm 5 additionally assigns an ordering reward r_T^{ord} that counts pairwise agreement between the ranking of cluster sizes and the ranking of allocated honeypot counts. The reward passed to the learner is therefore

$$r_t = r_t^{\text{loc}} + \mathbf{1}_{t=T} r_T^{\text{ord}}, \quad (10)$$

and the implemented discounted objective is

$$J_{\text{train}}(\pi) = \mathbb{E}_\pi \left[\sum_{t=1}^T \gamma_{\text{RL}}^{t-1} r_t \right]. \quad (11)$$

Using temporal-difference error

$$\Delta_t = r_t + \gamma_{\text{RL}} V_\phi(s_{t+1}) - V_\phi(s_t), \quad (12)$$

the actor is updated in the direction $\Delta_t \nabla_{\theta} \log \pi_\theta(a_t | s_t)$ and the critic minimizes the squared TD error.

Infection rate $\text{IR}(H)$ and DDoS damage $\text{DD}(H)$ are external security-evaluation metrics for the experiments reported in this paper; they are not fed back as terminal training rewards. This distinction is required because the cumulative shaped reward is not monotone with infection reduction in the reported learning analysis. Consequently, RLHA's current training objective is interpreted as a topology-informed proxy for the security objective. A propagation-outcome terminal reward would define a different training objective and would require retraining before its performance could be claimed.

D. RL model

Reinforcement learning (RL) learns decision policies through interaction with an environment. At each step, an agent observes a state, selects an action, receives a reward or penalty, and updates its policy to maximize cumulative reward over repeated episodes. We adopt an actor-critic model, where the actor selects the allocation action and the critic estimates its expected value.

In RLHA, honeypot allocation is modeled as a sequential decision problem. At each step, the agent observes topology-aware features of one candidate node and the remaining budget, then selects an integer action indicating how many honeypots should be assigned to that node. Actions exceeding the remaining budget are masked as infeasible. The node order is deterministic: for ordered candidates $n_1, n_2, \dots, n_k$, the agent moves from n_i to n_{i+1} . However, the full state transition is action-dependent because each action changes the remaining budget and may update the local honeypot-coverage feature used in later states.

The reward function encourages effective and low-cost allocations. Positive rewards are assigned when actions match propagation-relevant features such as vulnerability, degree, neighborhood weakness, and reachability, while negative rewards penalize excessive deployment or budget violations. Thus, RLHA learns how many honeypots to allocate to each node in order to slow botnet propagation under the defender's budget constraint.

We have considered six node-based and a single network-based features to represent an environment state: the node weight (F_1), the node's degree centrality (F_2), the summation of the node's neighbors' weight (F_3), the summation of the shortest distance between a node and the other nodes in the network (F_4), the minimum number of honeypots that are deployed on the neighbor nodes (F_5), and the number of available honeypots to be deployed (F_6). For readability, Table V summarizes these features and their intended role. Before introducing these features in detail, it must be noted that we have considered the proportion of the quantities instead of the exact numerical value of features. This method helps the agent perceive the next states, even though they are not still shown. Let us give an example to show the benefit of this method. Recalling the sample smart home network in Figure 2, $\mathcal{X}$, in the first step of the game, the agent is provided with the features of s_1 , which are $(5, 4, 12, 12, 0, 7)$, and it is asked to select a number between 0 to 7 indicating the number of honeypots to be assigned to s_1 . However, the agent has no idea about the other nodes in the network to estimate the importance of the current node compared to the others. Now, suppose that we have divided the value of the first five features by the summation of the values of these features for the next nodes, including the current one. Hence, the agent is provided with $(5/24, 4/15, 12/54, 12/124, 0, 7)$, because the sum of nodes' weights, degrees, neighbors' weights, and distances is 24, 15, 54, 124, respectively. For F_5 the sum of deployed honeypots for all the nodes is zero; we provide zero. Now, the agent knows that the current node has $5/24$ and $4/15$ of weight and degree centrality in its network of the next nodes, respectively. So, if there are 7 honeypots left, they must be distributed between the current and the next nodes based on these portions.

TABLE V
STATE FEATURES USED IN RLHA AND THEIR ROLE IN ALLOCATION DECISIONS

Feature	Definition and intuition
F_1	Node weight, given by compromise-time in $\mathcal{N}_T$; lower weights indicate easier-to-compromise devices that can accelerate early growth.
F_2	Degree centrality of the node; higher degree increases the number of reachable neighbors and can speed up propagation.
F_3	Sum of neighbors' weights; small neighbor-weight sum indicates a vulnerable neighborhood that can amplify spread.
F_4	Sum of shortest-path distances from the node to other nodes; smaller values indicate more central nodes in terms of reachability.
F_5	Minimum number of honeypots deployed on neighbor nodes so far; captures local defensive coverage around the current node.
F_6	Remaining honeypot budget at the current step, equal to δ minus the number of deployed honeypots.

In the terminology of this paper, F_2 is implemented as the node degree over the considered graph, not as a normalized graph-theoretic centrality score unless explicitly normalized during state construction. The phrase degree centrality is therefore used in the operational sense of degree-driven exposure, and the normalized state value is obtained through the tail-sum normalization in Algorithm 3. In RLHA, the first five features are provided to the agent in normalized form using tail-sums over the remaining decision points, as detailed in Algorithm 3, so the agent can interpret the relative importance of the current node compared to the nodes that are yet to be processed.

For the reward function, we consider both short-term and long-term rewards. Short-term rewards are assigned after each step, while the long-term reward is given at the end of an episode and is described in subsection IV-E. The short-term rewards are defined as follows:

- Each deployed honeypot incurs a cost penalty of -1 .
- Allocations favoring lower-weight, easier-to-compromise nodes receive $+2$.
- Allocations reflecting higher node degree receive $+10$, since highly connected nodes can accelerate propagation.
- Allocations favoring nodes with lower neighbors' weight sums receive $+5$.
- Allocations based on node-distance information receive $+10$, reflecting the structural importance of nodes involved in shorter propagation paths.
- Allocations favoring nodes with low neighboring honeypot coverage receive $+1$.
- Selecting more honeypots than the remaining budget receives a penalty of -1000 .

These short-term rewards guide the learning process but do not guarantee an optimal allocation.

The short-term shaping terms and the cluster-ordering reward define the implemented training signal. Infection rate and DDoS damage are evaluated separately as security outcomes and are not used as terminal rewards in the reported training loop. Any mismatch between cumulative reward and infection rate is therefore treated as an objective-alignment limitation; see Sec. IV-C.

E. Clustering technique

IoT networks are commonly large-scale, and the model described in subsection IV-D may consume an extremely long time to find the optimal honeypot allocation solution. Hence, we have utilized clustering algorithms to divide the network into limited groups of similar nodes, called the center nodes, and then present only these centers to the agent to find the solution. To identify the similar nodes as a cluster, we have considered their weights and central degrees and

the summations of their neighbors' weights. Focusing on the center nodes instead of the whole nodes makes RLHA scalable, and it can be deployed for securing large-scale networks.

F. Cluster-level decision points and scalability rationale

For large-scale IoT networks, treating every node as a decision point can make training inefficient. RLHA therefore clusters non-critical nodes into k groups and uses the cluster centers as representative decision points. Honeypots are allocated only to these centers, while the long-term reward favors centers representing larger clusters. This reduces the sequential decision space to at most k points while preserving relevant propagation structure.

G. From center-level decisions to network deployment

Clustering reduces the number of decision points by representing each cluster with a center node. RLHA then allocates honeypots only over the set of centers, and deployment follows a center-attached abstraction. Specifically, if the clustering procedure yields clusters $\{C_1, \dots, C_k\}$ with corresponding center nodes $\{c_1, \dots, c_k\}$, RLHA outputs an allocation $\hat{H}$ defined on the centers, where $\hat{H}(c_m)$ is the number of honeypots assigned to center c_m . The final network-level deployment is obtained by attaching $\hat{H}(c_m)$ honeypots directly to c_m for each m and assigning zero honeypots to all non-center nodes.

This design is consistent with the interaction abstraction used in subsection III-F: when a bot probes a protected location, its attempt is diverted with higher probability as the number of attached honeypots increases. Attaching decoys to center nodes makes the defense scalable, because each center represents a group of similar devices and influences propagation through highly connected or structurally representative locations, as illustrated in Figure 4. While alternative mappings that distribute honeypots within each cluster are possible, the center-attached mapping keeps deployment simple and aligns the learning objective with a small, stable action space.

Any clustering algorithm, such as Bisecting K-means and Hierarchical algorithms, can be utilized to scale RLHA. Here, we describe our clustering techniques based on K-means (Algorithm 1). In the first part of this algorithm (line 1 to line 13), a triple is assigned to each node, containing its weight, degree, and sum of neighbors' weights, and then they are stored in a variable called *points*. Now, the clustering algorithm randomly selects k of these points as the initial centers (line 14) and updates them repeatedly until the convergence criterion is met (line 18 to line 35). In each iteration of this updating loop for each of the remaining nodes, the cluster with the closest distance is selected, and they join it (line 19 to line 27). Once all the nodes are assigned to the clusters, the center nodes are updated based on the new distances within the clusters. *clusters* is the return value of this procedure, a map of key-value pairs. The keys are the center nodes, and the values are the nodes assigned to the related cluster.

We update the reward function according to the clustering concept used in RLHA. The agent receives the long-term reward once the current episode is finished. The value of this reward is equal to the five times the number of center node pairs, say (c_1, c_2) , that if c_1 appears after c_2 when the center nodes are sorted based on their cluster size, c_1 also appears after c_2 when the centers are sorted based on the number of their assigned honeypots. In other words, this long-term reward tries to lead the agent toward assigning more honeypots to the center nodes in larger clusters to protect more devices.

Now, let us completely describe the scalable RLHA. The first step of training the agent is to initiate its history record. This process is explained in Algorithm 2. This algorithm mainly intends to initiate the variables of the agent's history once the training phase starts. In the loop from line 14 to line 26, the shortest path algorithm, such as Dijkstra, calculates the distances between the nodes (line 21). The main complexity of our proposed model is due to this line, which is of $O(k^4)$, where k is the number of clusters. For small- or medium-scale networks, the number of clusters equals the number of nodes. But, for large-scale ones, k is a fixed limited value. We have considered

Algorithm 1 The procedure of clustering the nodes based on K-means in the proposed scalable RL model (RLHA)

Require: $\mathcal{N}$ (the input network)
Require: $\mathcal{D}^{\mathcal{N}}$ (the input defender)
Require: $conv$ (the clustering convergence criterion)
Require: k (the number of clusters)
Ensure: $clusters$ (the clusters and their center node)

```

1:  $points \leftarrow$  empty map
2: for each  $n \in \mathcal{N}_V - \mathcal{N}_C$  do
3:    $weight \leftarrow \mathcal{N}_T[n]$ 
4:    $degree \leftarrow 0$ 
5:    $nei \leftarrow 0$ 
6:   for each  $e \in \mathcal{N}_E$  do
7:     if  $e[0] = n$  and  $e[1] \notin \mathcal{N}_C$  then
8:        $degree \leftarrow degree + 1$ 
9:        $nei \leftarrow nei + \mathcal{N}_T[e[1]]$ 
10:    else if  $e[1] = n$  and  $e[0] \notin \mathcal{N}_C$  then
11:       $degree \leftarrow degree + 1$ 
12:       $nei \leftarrow nei + \mathcal{N}_T[e[0]]$ 
13:     $points[n] \leftarrow (weight, degree, nei)$ 
14:  $centers \leftarrow$  a random subset of  $\mathcal{N}_V - \mathcal{N}_C$  with size  $k$ 
15:  $clusters \leftarrow$  empty map
16: for each  $c \in centers$  do
17:    $clusters[c] \leftarrow \{c\}$ 
18: while  $conv$  is not met do
19:   reset  $clusters$  to empty sets for the current iteration
20:   for each  $p \in points$  do
21:      $mdist \leftarrow \infty$ 
22:      $mc \leftarrow NULL$ 
23:     for each  $c \in centers$  do
24:        $dist \leftarrow$  distance between  $points[c]$  and  $points[p]$ 
25:       if  $dist < mdist$  then
26:          $mdist \leftarrow dist$ 
27:          $mc \leftarrow c$ 
28:    $clusters[mc] \leftarrow clusters[mc] + p$ 
29:   for each  $c \in centers$  do
30:      $newc \leftarrow$  node closest to the centroid of  $clusters[c]$ 
31:     if  $c = newc$  then
32:       continue
33:    $centers \leftarrow centers - \{c\}$ 
34:    $centers \leftarrow centers + \{newc\}$ 
35:    $clusters[newc] \leftarrow clusters[c]$ 
36:   remove  $clusters[c]$ 
37: return  $clusters$ 

```

$k = 50$. Hence, this line does not exponentially increase the execution time.

When shortest paths are computed from each of the k centers over the original graph using Dijkstra's algorithm, the dominant distance-computation cost is $O(k(|\mathcal{N}_E| + |\mathcal{N}_V| \log |\mathcal{N}_V|))$. If distances are instead computed only on a dense center-level graph, the corresponding cost is $O(k^3)$ using Floyd-Warshall. The practical complexity therefore depends on the shortest-path routine and whether distances are evaluated on the original graph or on a center-level graph.

The algorithm for generating the environment states is shown in Algorithm 3. Being at the i^{th} step, the features of the i^{th} center nodes are provided to the agent (line 1). The first five parts of a state indicate the proportion of the current center node's features to the sum of that feature for all the centers (line 3 to line 7). Finally, the last part of a state is the total number of available honeypots, which is equal to the total number of honeypots minus the number of deployed ones (line 8)

An episode of the game finishes once the number of deployed honeypots reaches the maximum number of honeypots ($\mathcal{D}_\delta^{\mathcal{N}}$) or when the agent passes k steps, where k is the number of clusters. The latter condition indicates that the agent has explored all the possible locations for placing the honeypots. The procedure of determining whether or not the game has finished is shown in Algorithm 4.

Considering the long-term reward and the short-term rewards mentioned in subsection IV-D, to figure out which action is the optimal one in each state, we compare the value of F_1 to F_4 with the value of action divided by the number of remained honeypots. We

Algorithm 2 The procedure of initiating the agent in the proposed scalable RL model (RLHA)

Require: $\mathcal{N}$ (the input network)
Require: $\mathcal{D}^{\mathcal{N}}$ (the input defender)
Require: $conv$ (the clustering convergence criterion)
Require: k (the number of clusters)
Require: th (the agent's learning threshold)
Ensure: a (the agent's history)

```

1:  $a \leftarrow$  An instance of RL agent
2:  $a.nodes, a.edges \leftarrow \mathcal{N}_V - \mathcal{N}_C, \mathcal{N}_E$ 
3:  $a.step, a.th, a.depl, a.delta \leftarrow 0, th, 0, \mathcal{D}_\delta^{\mathcal{N}}$ 
4:  $a.clusters \leftarrow$  Algorithm 1( $\mathcal{N}, \mathcal{D}^{\mathcal{N}}, conv, k$ )
5:  $a.centers \leftarrow \{\}$ 
6: for each  $c \in a.clusters$  do
7:    $a.centers \leftarrow a.centers + \{c\}$ 
8:  $a.weight, a.tweight \leftarrow$  empty map, empty map
9:  $a.degree, a.tdegree \leftarrow$  empty map, empty map
10:  $a.nei, a.tnei \leftarrow$  empty map, empty map
11:  $a.dist, a.tdist \leftarrow$  empty map, empty map
12:  $a.honey, a.thoney \leftarrow$  empty map, empty map
13:  $a.minh, a.tminh \leftarrow$  empty map, empty map
14: for each  $n \in a.centers$  do
15:   for each  $w \in \mathcal{N}_T$  do
16:     if  $w[0] = n$  then
17:        $a.weight[n] \leftarrow w[1]$ 
18:       break
19:    $a.dist[n] \leftarrow 0$ 
20:   for each  $n' \in a.centers$  do
21:      $d \leftarrow$  shortest distance between  $n$  and  $n'$ 
22:      $a.dist[n] \leftarrow a.dist[n] + d$ 
23:    $a.honey[n] \leftarrow 0$ 
24:    $a.thoney[n] \leftarrow 0$ 
25:    $a.minh[n] \leftarrow 0$ 
26:    $a.tminh[n] \leftarrow 0$ 
27: for each  $n \in a.centers$  do
28:    $a.degree[n] \leftarrow 0$ 
29:    $a.nei[n] \leftarrow 0$ 
30:   for each  $d \in \mathcal{N}_E$  do
31:     if  $d[0] = n$  then
32:        $a.degree[n] \leftarrow a.degree[n] + 1$ 
33:        $a.nei[n] \leftarrow a.nei[n] + \text{weight of } d[1] \text{ if } d[1] \notin \mathcal{N}_C$ 
34:     else if  $d[1] = n$  then
35:        $a.degree[n] \leftarrow a.degree[n] + 1$ 
36:        $a.nei[n] \leftarrow a.nei[n] + \text{weight of } d[0] \text{ if } d[0] \notin \mathcal{N}_C$ 
37:  $weight, degree, nei, dist \leftarrow 0, 0, 0, 0$ 
38: for each  $n \in \text{reverse}(a.centers)$  do
39:    $weight \leftarrow weight + a.weight[n]$ 
40:    $a.tweight[n] \leftarrow weight$ 
41:    $degree \leftarrow degree + a.degree[n]$ 
42:    $a.tdegree[n] \leftarrow degree$ 
43:    $nei \leftarrow nei + a.nei[n]$ 
44:    $a.tnei[n] \leftarrow nei$ 
45:    $dist \leftarrow dist + a.dist[n]$ 
46:    $a.tdist[n] \leftarrow dist$ 
47: return  $a$ 

```

Algorithm 3 The procedure of generating the environment states in the proposed scalable RL model (RLHA)

Require: a (the agent's history)
Ensure: $state$ (the current environment state)

```

1:  $n \leftarrow a.centers[a.step]$ 
2:  $state \leftarrow$  a tuple with 6 zeros
3:  $state[0] \leftarrow a.weight[n]/a.tweight[n]$ 
4:  $state[1] \leftarrow a.degree[n]/a.tdegree[n]$ 
5:  $state[2] \leftarrow a.nei[n]/a.tnei[n]$ 
6:  $state[3] \leftarrow a.dist[n]/a.tdist[n]$ 
7: if  $a.tminh[n] > 0$  then
8:    $state[4] \leftarrow a.minh[n]/a.tminh[n]$ 
9: else
10:    $state[4] \leftarrow 0$ 
11:  $state[5] \leftarrow a.delta - a.depl$ 
12: return  $state$ 

```

Algorithm 4 The procedure of determining the end of a game in the proposed scalable RL model (RLHA)

Require: a (the agent's history)
Ensure: $done$ (the game-ending flag)
1: $done \leftarrow \text{FALSE}$
2: **if** $a.\text{delta} = a.\text{depl}$ **then**
3: | $done \leftarrow \text{TRUE}$
4: **else if** $a.\text{step} = \text{size}(a.\text{centers})$ **then**
5: | $done \leftarrow \text{TRUE}$
6: **return** $done$

Algorithm 5 The procedure of generating the reward in the proposed scalable RL model (RLHA)

Require: a (the agent's history)
Require: $action$ (the agent's current action)
Require: $state$ (the environment's current state)
Ensure: $reward$ (the agent's reward)
1: $sreward \leftarrow 0$
2: **if** $state[5] \geq action$ **then**
3: | $sreward \leftarrow -1 \times action$
4: | **if** $state[5] > 0$ **then**
5: | | $p \leftarrow action/state[5]$
6: | **else**
7: | | $p \leftarrow 0$
8: | **if** $p - a.th \leq 1 - state[0] \leq p + a.th$ **then**
9: | | $sreward \leftarrow sreward + 2$
10: | **if** $p - a.th \leq state[1] \leq p + a.th$ **then**
11: | | $sreward \leftarrow sreward + 10$
12: | **if** $p - a.th \leq 1 - state[2] \leq p + a.th$ **then**
13: | | $sreward \leftarrow sreward + 5$
14: | **if** $p - a.th \leq 1 - state[3] \leq p + a.th$ **then**
15: | | $sreward \leftarrow sreward + 10$
16: | **if** $p - a.th \leq 1 - state[4] \leq p + a.th$ **then**
17: | | $sreward \leftarrow sreward + 1$
18: **else**
19: | $sreward \leftarrow -1000$
20: $lreward \leftarrow 0$
21: $done \leftarrow \text{Algorithm 4 } (a)$
22: **if** $done$ **then**
23: | $hbase \leftarrow \text{sort } a.\text{centers} \text{ based on } a.\text{honey}[c]$
24: | $cbase \leftarrow \text{sort } a.\text{centers} \text{ based on } |a.\text{clusters}[c]|$
25: | **for each** $0 \leq i < \text{size}(a.\text{centers}) - 1$ **do**
26: | | **for each** $i < j < \text{size}(a.\text{centers})$ **do**
27: | | | $c1 \leftarrow hbase.\text{index}(cbase[i])$
28: | | | $c2 \leftarrow hbase.\text{index}(cbase[j])$
29: | | | **if** $c1 < c2$ **then**
30: | | | | $lreward \leftarrow lreward + 5$
31: **return** $sreward + lreward$

assume a threshold for this comparison that varies from 0.1 to 0.9. Algorithm 5 shows the detail of the rewarding function in RLHA. The short-term reward is calculated in the first part of this algorithm (line 1 to line 16). If $state[5]$ is higher than or equal to $action$, the agent has not exceeded the maximum number of honeypots to locate. As a result, in this case, each part of the short-term rewards is calculated. Otherwise, the agent receives a significant punishment (line 16). Since the long-term reward is received by the agent only when the game is over, we have a conditional statement to check the end of the game (line 19). Next, the centers are sorted based on the deployed honeypots and the cluster size, and the result is stored in $hbase$ and $cbase$, respectively (line 20 and line 21). Now, we need a nested loop over the center nodes to calculate the number of centers that appear after them (line 22 to line 27). At last, the summation of short- and long-term rewards is returned (line 28).

Finally, the proposed scalable RL model is trained based on Algorithm 6. In this algorithm, first an RL model is instantiated (line 1), and then its agent is initiated based on Algorithm 2 (line 2). We must loop over the training process until the number of passed episodes reaches its specified input value (line 3). Another loop is also required for playing the game until it finishes (line 4 to line 30). When a single game episode is completed, the game history is

Algorithm 6 The procedure of training the proposed scalable RL model (RLHA)

Require: $\mathcal{N}$ (the input network)
Require: $\mathcal{D}^{\mathcal{N}}$ (the input defender)
Require: $conv$ (the clustering convergence criterion)
Require: k (the number of clusters)
Require: th (the learning threshold)
Require: e (the number of training episodes)
Ensure: m (the trained model)
1: $m \leftarrow$ an instance of RL model
2: $m.a \leftarrow \text{Algorithm 2 } (\mathcal{N}, \mathcal{D}^{\mathcal{N}}, conv, k, th)$
3: **for** $0 \leq i \leq e$ **do**
4: | **while** **TRUE** **do**
5: | | $done \leftarrow$ the game ending flag regarding $m.a$
6: | | **if** $done$ **then**
7: | | | **break**
8: | | $s \leftarrow \text{Algorithm 3 } (m.a)$
9: | | $A \leftarrow$ an action selected by the masked actor policy
10: | | $r \leftarrow \text{Algorithm 5 } (m.a, A, s)$
11: | | $n \leftarrow m.a.\text{centers}[m.a.\text{step}]$
12: | | $m.a.\text{depl} \leftarrow m.a.\text{depl} + A$
13: | | $m.a.\text{honey}[n] \leftarrow A$
14: | | $N \leftarrow m.a.\text{centers}$
15: | | $E \leftarrow m.a.\text{edges}$
16: | | **for each** $n' \in N$ **do**
17: | | | **if** $(n, n') \in E$ **or** $(n', n) \in E$ **then**
18: | | | | **if** $m.a.\text{honey}[n] \geq m.a.\text{minh}[n']$ **then**
19: | | | | | **continue**
20: | | | | $m.a.\text{minh}[n'] \leftarrow m.a.\text{honey}[n]$
21: | | | | **if** $N.\text{index}(n') < N.\text{index}(n)$ **then**
22: | | | | | **continue**
23: | | | | **for each** $n'' \in N$ **do**
24: | | | | | **if** $N.\text{index}(n') < N.\text{index}(n'')$ **then**
25: | | | | | | **continue**
26: | | | | | $t \leftarrow m.a.\text{tminh}[n'']$
27: | | | | | $t \leftarrow t + m.a.\text{minh}[n']$
28: | | | | | $m.a.\text{tminh}[n'] \leftarrow t$
29: | | | $m.a.\text{step} \leftarrow m.a.\text{step} + 1$
30: | | | Update $m.a$ based on A, s , and r
31: | | Reset allocation history
32: **return** m

reset to remove the allocated honeypots for the next episodes (line 31). Inside the game loop (line 5 to line 30), the ending condition of the game is determined, and if it is over, the loop is broken (line 5 to line 7). Then, the action selected by the masked actor policy and its related reward are generated (line 9 and line 10). If the agent acts A , it means that A honeypots are assigned to the current center nodes. As a result, the number of deployed honeypots and the map storing the deployed honeypots assigned to each center are updated (line 12 and line 13). To update the variables related to F_5 (i.e., the minimum number of deployed honeypots assigned to the neighbors), we have to perform a nested loop on the current centers' neighbors and their neighbors (line 16 to line 28). After the agent's history is completely updated, we increment the number of passed steps (line 29) and update the agent to learn the triple of action, state, and reward (line 30).

The pseudocode above specifies the intended implementation logic. The deployed honeypot counter is increased by the selected action A , the agent initialization call uses the same arguments as the agent-initialization procedure, the zero-denominator case in F_5 normalization is explicitly handled, and long-term reward sorting is performed using actual honeypot counts and cluster sizes rather than the sizes of map objects.

H. Mathematical formulation of RLHA

This subsection provides a complete mathematical view of RLHA that complements the step-by-step story in Figure 4. The defender chooses an integer allocation map $H : \mathcal{N}_V \setminus \mathcal{N}_C \rightarrow \mathbb{N}_0$, where $H(i)$

denotes the number of honeypots attached to device i . The allocation is budget-feasible if it satisfies

$$\sum_{i \in \mathcal{N}_V \setminus \mathcal{N}_C} H(i) \leq \delta, \quad H(i) = 0 \quad \forall i \in \mathcal{N}_C. \quad (13)$$

To model the intended deception effect, we use the same abstraction as in subsection III-F. When a bot probes a target device i , the connection is resolved uniformly at random over the set consisting of the real device and its attached honeypots. Therefore, the probability of being diverted to a honeypot is

$$\rho(i) = \frac{H(i)}{H(i) + 1}. \quad (14)$$

If diversion occurs, the bot wastes σ time units and no real device is compromised in that probe. This yields an expected wasted time of $\rho(i)\sigma$ per probe directed to i . RLHA aims to learn an allocation H that increases the cumulative wasted time during reconnaissance and propagation while respecting the deployment budget, which is precisely the high-level effect illustrated in Figure 4.

RLHA implements the allocation as a sequential decision process. Let the decision order be a fixed permutation of non-critical nodes, denoted by $(i_1, i_2, \dots, i_T)$ where $T = |\mathcal{N}_V \setminus \mathcal{N}_C|$. Let b_t denote the remaining honeypot budget at step t , with $b_1 = \delta$. At step t , the agent chooses an integer action a_t satisfying

$$a_t \in \{0, 1, \dots, b_t\}, \quad (15)$$

and updates the allocation and budget as

$$H(i_t) \leftarrow a_t, \quad b_{t+1} \leftarrow b_t - a_t. \quad (16)$$

The episode terminates when either $b_t = 0$ or $t = T$, matching the stopping logic in Algorithm 4. The policy is trained to maximize the expected return

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=1}^T r_t \right], \quad (17)$$

where the immediate reward r_t is computed using the short-term shaping rules in Algorithm 5. When clustering is enabled, an additional episode-level reward is added as described in subsection IV-E to encourage assigning more honeypots to centers representing larger clusters.

In the scalable variant, non-critical nodes are partitioned into k clusters, denoted by $\{\mathcal{C}_1, \dots, \mathcal{C}_k\}$, with center nodes $\{c_1, \dots, c_k\}$. RLHA then runs the sequential allocation only over the center nodes, reducing the number of decision points from T to k . The deployed honeypots are attached to the selected centers, and the resulting allocation map is

$$H(c_m) = a_m \quad \forall m \in \{1, \dots, k\}, \quad H(i) = 0 \quad \forall i \notin \{c_1, \dots, c_k\}, \quad (18)$$

which matches the deployment intuition in Figure 4 where decoys are placed on representative nodes to influence propagation at scale.

I. Complete numerical example on the illustrative subnetwork

This subsection provides a complete numerical walk-through on the illustrative smart-home subnetwork $\mathcal{X}$ introduced in Section III. We use $\delta = 7$ and the non-critical node set $\mathcal{X}_V \setminus \mathcal{X}_C = \{s_1, s_2, s_3, s_4, s_5, s_6, s_8, s_9\}$, where s_7 is the critical asset. The decision order follows the node identifiers, namely $(s_1, s_2, s_3, s_4, s_5, s_6, s_8, s_9)$.

The raw topology-aware features used by RLHA are listed in Table VI. Here, F_1 is the compromise-time weight, F_2 is the degree excluding critical neighbors, F_3 is the sum of neighbor weights excluding critical nodes, and F_4 is the sum of shortest-path distances from the node to all other non-critical nodes. The feature F_5 depends on the current deployment history, and F_6 is the remaining budget.

TABLE VI
RAW FEATURE VALUES FOR THE ILLUSTRATIVE SMART-HOME SUBNETWORK $\mathcal{X}$ (EXCLUDING THE CRITICAL ASSET s_7)

Node	F_1	F_2	F_3	F_4
s_1	5	4	12	12
s_2	1	1	5	18
s_3	4	1	5	18
s_4	2	2	10	14
s_5	5	3	10	11
s_6	3	3	9	13
s_8	3	1	3	19
s_9	1	1	3	19

TABLE VII
ONE COMPLETE EPISODE ON $\mathcal{X}$ WITH TAIL-NORMALIZED STATES, REMAINING BUDGET, AND ACTIONS. THE STATE SHOWN IS $(\tilde{F}_1, \tilde{F}_2, \tilde{F}_3, \tilde{F}_4, \tilde{F}_5, b_t)$.

Step	Node	State (rounded)	b_t	Action	Short-term reward
1	s_1	(0.208, 0.250, 0.211, 0.097, 0.000, 7)	7	0	0
2	s_2	(0.053, 0.083, 0.111, 0.161, 0.000, 7)	7	0	10
3	s_3	(0.222, 0.091, 0.125, 0.191, 0.000, 7)	7	1	9
4	s_4	(0.143, 0.200, 0.286, 0.184, 0.000, 6)	6	0	10
5	s_5	(0.417, 0.375, 0.400, 0.177, 0.000, 6)	6	0	0
6	s_6	(0.429, 0.600, 0.600, 0.255, 0.000, 6)	6	3	14
7	s_8	(0.750, 0.500, 0.500, 0.500, 1.000, 3)	3	0	1
8	s_9	(1.000, 1.000, 1.000, 1.000, 1.000, 3)	3	2	-2

At step t , RLHA normalizes features using tail-sums over the remaining decision points. Let $\mathcal{R}_t = \{i_t, i_{t+1}, \dots, i_T\}$. For $\ell \in \{1, 2, 3, 4\}$, the normalized features are

$$\tilde{F}_\ell(i_t) = \frac{F_\ell(i_t)}{\sum_{j \in \mathcal{R}_t} F_\ell(j)}. \quad (19)$$

For the neighbor-coverage feature, define

$$m_t(i) = \begin{cases} \min_{u \in \mathcal{N}(i) \cap \{i_1, \dots, i_{t-1}\}} H(u), & \mathcal{N}(i) \cap \{i_1, \dots, i_{t-1}\} \neq \emptyset, \\ 0, & \text{otherwise,} \end{cases} \quad (20)$$

and normalize it as

$$\tilde{F}_5(i_t) = \begin{cases} \frac{m_t(i_t)}{\sum_{j \in \mathcal{R}_t} m_t(j)}, & \sum_{j \in \mathcal{R}_t} m_t(j) > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (21)$$

Finally, the remaining-budget feature is $\tilde{F}_6(i_t) = b_t$, where b_t is updated by Equation 16.

We now show a complete episode. Consider the deployment outcome $H(s_3) = 1$, $H(s_6) = 3$, and $H(s_9) = 2$, with all other non-critical nodes receiving zero honeypots, which respects $\sum_i H(i) \leq \delta$. The per-step states, actions, remaining budget, and short-term rewards are summarized in Table VII. The short-term rewards are computed using Algorithm 5 with a fixed illustrative threshold value $th = 0.2$. This episode-level walk-through is consistent with the network intuition in Figure 4, where the policy allocates decoys to a small set of influential locations.

At the end of this episode, the deployed allocation is

$$H(s_3) = 1, \quad H(s_6) = 3, \quad H(s_9) = 2, \\ H(i) = 0, \quad \forall i \in \{s_1, s_2, s_4, s_5, s_8\}. \quad (22)$$

and the remaining budget is $b_{T+1} = 1$. In the scalable variant, the same episode structure applies over center nodes only, and the additional episode-level reward in subsection IV-E encourages allocating more honeypots to centers that represent larger clusters, consistent with the center-focused steps in Figure 4.

J. Complexity and scalability analysis

Let $n = |\mathcal{N}_V \setminus \mathcal{N}_C|$ be the number of non-critical nodes and k be the number of clusters used in the scalable version. Without clustering, each episode visits all n candidate nodes, and distance-based features may require repeated shortest-path computations, which becomes costly on large graphs.

With clustering, RLHA reduces the decision space from n nodes to k center nodes. Clustering is performed once using compact node features, and subsequent allocation decisions are made only over the selected centers. Therefore, each episode has length k instead of n , and distance-related computations are limited to representative nodes. Since k is fixed to a small value for large networks, the training and inference cost no longer grows directly with the full network size, making RLHA scalable for large IoT topologies.

V. EVALUATION RESULTS

The performance of RLHA is evaluated on network-connectivity proxies used to instantiate controlled propagation scenarios; these graphs are not presented as native IoT deployment traces. We use SNDlib [45] to represent small and medium-scale topologies and Stanford peer-to-peer Gnutella networks [46] to represent large-scale connectivity. Mininet is used as the execution environment for topology instantiation and controlled traffic generation [47], with Open vSwitch providing the virtual switching substrate [48]. The detailed characteristics of the selected topologies are reported in Table VIII.

The evaluation treats SNDlib and Stanford Gnutella as graph-connectivity proxies rather than native IoT deployment traces. SNDlib provides small and medium connected network graphs suitable for controlled emulation, while the Stanford Gnutella graphs provide large sparse topologies for scalability stress testing. Accordingly, the claims in this paper are limited to propagation-slowing behavior over the evaluated graph structures and threat model, not to universal effectiveness across all possible IoT deployments.

A. Rationale for dataset characteristics and scale categories

The dataset characteristics in Table VIII were selected because they directly affect propagation, allocation complexity, and scalability. The number of nodes determines the number of possible devices that can become infected or protected. The number of edges determines the reconnaissance surface and the number of feasible propagation paths available to bots. Therefore, both node count and edge count are necessary to interpret infection speed, honeypot coverage, and runtime.

The selected datasets were grouped into small-, medium-, and large-scale settings according to the number of nodes and the resulting feasibility of full node-level allocation. Small and medium SNDlib graphs allow detailed emulation and visual inspection of propagation behavior, whereas Stanford Gnutella graphs provide stress tests for scalability because they contain thousands to tens of thousands of nodes. Although these graphs are not IoT-native deployment traces, they are used as connectivity proxies to evaluate how RLHA behaves under heterogeneous graph sizes and densities. This distinction is important: the purpose of Table VIII is not to claim that every topology is a direct replica of an IoT deployment, but to evaluate whether the allocation method remains effective when the underlying network connectivity changes substantially.

B. Experimental setup and evaluation protocol

Across all topologies, dataset edges define device connectivity and feasible reconnaissance and propagation paths. Critical assets are selected as 10% of the nodes, i.e., $|\mathcal{N}_C| = 0.1|\mathcal{N}_V|$, and are connected to randomly selected normal devices to model service dependencies. A critical asset is considered degraded by DDoS when at least half of its neighbors are compromised as bots, following the overload rule in Section III.

TABLE VIII
THE MAIN FEATURES OF THE SMALL AND MEDIUM-SCALE AND LARGE-SCALE DATASETS USED FOR EVALUATING RLHA

Category	Name	No. of Nodes	No. of Edges
SNDlib	nobel-eu	28	41
	di-yuan	11	42
	zib54	54	80
	Norway	27	51
	sun	27	51
	dfn-bwin	10	45
	France	25	45
	dfn-gwin	11	47
	nobel-us	14	21
	brain	161	166
	pioro40	40	89
	pdh	11	34
	polska	12	18
	abilene	12	15
	janos-us	26	42
	cost266	37	57
	giul39	39	86
	ta2	65	108
	Atlanta	15	22
	germany50	50	88
	india35	35	80
	NewYork	16	49
	janos-us-ca	39	61
Stanford	nobel-Germany	17	26
	geant	22	36
	ta1	24	51
	p2p-Gnutella04	10,876	39,994
	p2p-Gnutella05	8,846	31,839
	p2p-Gnutella06	8,717	31,525
	p2p-Gnutella08	6,301	20,777
	p2p-Gnutella09	8,114	26,013
	p2p-Gnutella24	26,518	65,369
	p2p-Gnutella25	22,687	54,705
	p2p-Gnutella30	36,682	88,328
	p2p-Gnutella31	62,586	147,892

Device compromise-time weights are sampled as integers from one to six to represent heterogeneous vulnerability levels. The honeypot engagement delay is fixed at $\sigma = 1$, the defender budget is $\delta = 100$, and the scalable RLHA configuration uses at most $k = 50$ clusters for large-scale networks. The adversary follows Mirai-like scanning and propagation behavior [49], where infected bots probe neighboring devices and attempt exploitation. The initial infection ratio is swept from 10% to 90% of the network size, i.e., $0.1|\mathcal{N}_V| \leq |\mathcal{A}_B^N| \leq 0.9|\mathcal{N}_V|$.

Small and medium topologies are evaluated in a Mininet-based testbed [47] using Open vSwitch [48] and a POX controller, while all compared methods are tested under identical topology, node-weight, critical-asset, and initial-infection settings. To reduce sensitivity to random placement, reported curves and bars are averaged over multiple independent runs, where node weights, critical-asset placement, and initial infected sets are re-sampled while keeping the topology fixed.

For scalability reasons, small and medium SNDlib graphs are instantiated in Mininet, while the large Stanford Gnutella graphs are evaluated through graph-level propagation simulation using the same infection, diversion, and asset-degradation rules. This avoids the impractical overhead of instantiating tens of thousands of virtual hosts. The reported configurations use repeated random seeds for descriptive averaging. Because the present figures and tables do not report seed-level confidence intervals or paired-test statistics, the plotted differences are interpreted descriptively and are not labeled statistically significant; Sec. V-F states this evidence boundary explicitly.

The reported main experiments use the budget and initial-infection settings listed in Table IX. Normalized budgets and additional early-stage infection ratios are retained as required sensitivity dimensions

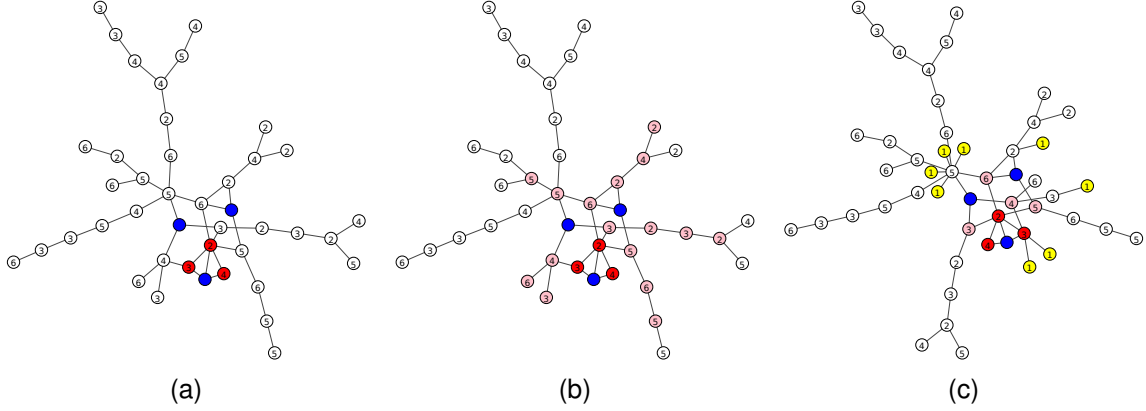


Fig. 5. An emulated network fitting the SNDlib-giul39 dataset containing three critical assets, three initially infected bots, and thirty six other normal devices in three states: (a) the botnet is not yet propagating, (b) the botnet has started its propagation and no honeypots are deployed, and (c) the botnet has started its propagation and honeypots are deployed.

TABLE IX
SCENARIO PARAMETERS USED IN EVALUATION

Parameter	Value and description
$ \mathcal{N}_C / \mathcal{N}_V $	0.1, fraction of nodes treated as critical assets
Asset degradation rule	A critical asset is degraded if at least half of its neighbors are bots
Compromise-time weights	Discrete integer values from one to six, sampled per node
σ	1, honeypot engagement delay per diverted probe
δ	100, maximum number of deployable honeypots
k	At most 50 clusters for the scalable configuration
$ \mathcal{A}_B^N / \mathcal{N}_V $	Swept from 0.1 to 0.9, initial infection ratio
Threat behavior	Mirai-like scanning and exploitation dynamics [49]
λ	3, probes per bot per time unit, consistent with the illustrative adversary setting
γ	30, time units until the distributed attack stage
Number of independent runs	30 runs per configuration; reported values are averages over independent random seeds

rather than described as completed experiments unless the corresponding numerical results are included.

C. Implementation details for reproducibility

The actor and critic are implemented as two-layer feed-forward networks with hidden sizes 64 and 32 and ReLU activations. The actor outputs a categorical distribution over integer allocation actions, and infeasible actions larger than the remaining budget are masked before sampling. The critic outputs a scalar state-value estimate. The actor and critic learning rates are both set to 10^{-3} , the RL discount factor is $\gamma_{RL} = 0.95$, and the agent is trained for 500 episodes per topology and parameter setting. Exploration is performed by sampling from the masked actor distribution during training, while evaluation uses the highest-probability feasible action. Unless otherwise stated in a result caption or table, the repeated-run protocol uses 30 random seeds for descriptive averaging; inferential statistics are not claimed unless their intervals or test statistics are explicitly reported.

D. Baselines and fairness

The reported experiments compare RLHA with HAAG and AHDS under identical budgets, topologies, and threat parameters. HAAG represents attack-graph-based allocation, whereas AHDS follows an importance/degree-driven deployment strategy. RLHA is evaluated without clustering on small and medium graphs and with seven clustering methods on large graphs.

TABLE X
COMPARED AND DEFINED HONEYPOT ALLOCATION METHODS

Method	Core allocation signal	Results shown
AHDS	Degree/importance-driven automated placement	Yes
HAAG	Attack-graph-oriented allocation	Yes
RLHA	Actor-critic allocation with topology-aware state and budget mask	Yes
Random	Uniform feasible placement	No
Degree	Descending node degree	No
Vulnerability-aware	Lower compromise-time weight first	No
Betweenness	Descending shortest-path betweenness	No

For completeness, four additional heuristic baselines are defined for future matched evaluation: random, degree-based, vulnerability-aware, and betweenness-based placement. Since these controls are not included in the current quantitative results, no performance advantage over them is claimed.

E. Metrics

RLHA aims to slow botnet propagation, so we report infection dynamics over time and under varying initial infection ratios. The primary metric is the infection rate, computed at each time interval as

$$\text{Infection Rate} = \frac{100 \times \text{Number of bots}}{\text{Total number of normal devices}}. \quad (23)$$

To connect propagation slowing to protection of critical services, we also report DDoS damage, defined as

$$\text{DDoS Damage} = \frac{\text{Number of degraded assets}}{\text{Total number of critical assets}}. \quad (24)$$

Scalability is measured using the required time for finding an allocation strategy in large-scale topologies, and learning performance is assessed by sweeping the learning threshold and by reporting the reward score accumulated by the agent under each clustering option.

The relative infection-rate reduction reported in the abstract and conclusion is computed as

$$\Delta_{IR} = \frac{IR_{\text{baseline}} - IR_{\text{RLHA}}}{IR_{\text{baseline}}} \times 100\%. \quad (25)$$

The maximum reported reduction of 41% corresponds to the largest value of Δ_{IR} observed across the evaluated topology, budget, and initial-infection configurations against the strongest non-RLHA baseline in that configuration.

F. Statistical scope and claim discipline

The reported curves and bars are descriptive summaries of repeated runs. The available figures and tables do not provide seed-level confidence intervals, paired test statistics, or configuration-wise p -values; therefore, this paper does not label the observed differences as statistically significant. In particular, the 41% value is the maximum observed relative infection-rate reduction computed by Eq. 25; it is not an estimate of the mean or median improvement across the complete configuration grid.

For inferential comparison, the appropriate unit is the paired seed-level outcome under identical topology, node weights, critical-asset placement, initial infection set, budget, and threat parameters. If x_s and y_s denote RLHA and baseline outcomes under matched seed s , a complete inferential report should include confidence intervals for method-level outcomes, the paired differences $d_s = x_s - y_s$, a paired test such as the Wilcoxon signed-rank test when distributional assumptions are not justified, and multiplicity control across repeated configurations. Such values are not reconstructed from aggregate plots, because doing so would not recover the original sampling variability.

Figure 5 illustrates a representative emulation instance based on SNDlib-giul39. The red, pink, white, blue, and yellow circles indicate initially infected bots, devices compromised during propagation, uncompromised devices, critical assets, and deployed honeypots, respectively, and the numbers inside circles represent node weights.

G. Case study visualization on SNDlib-giul39

Figure 5 provides an interpretable snapshot of the propagation process and the effect of deception. The comparison between Figure 5b and Figure 5c illustrates the operational meaning of honeypot engagement delay: decoys absorb a portion of probing attempts and reduce the number of successful compromises that would otherwise spread along high-connectivity regions of the graph. This case study complements the aggregate curves by showing how a limited number of decoys can disrupt the growth pathways that lead to critical-asset overload.

Figure 5a shows the initial stage where the server compromises the initial bots but propagation has not yet started. After spread begins, bots scan neighbors and infect vulnerable devices. If the defender deploys no honeypots (Figure 5b), sixteen normal devices become compromised in this instance, while in the presence of honeypots (Figure 5c) the growth is reduced to four compromised devices.

The results reported in this section are averaged over multiple independent emulations per configuration, and all methods are compared under identical scenario settings as described in subsection V-B and subsection V-D.

H. Security performance

Figure 6 compares RLHA with HAAG and AHDS in terms of infection rate over time. The infection rate increases as time progresses, reflecting botnet growth. Under the same conditions, RLHA consistently yields a lower infection rate than both baselines, indicating that the learned allocation slows propagation more effectively. The improvement is most pronounced in the mid-propagation regime, where early diversions delay the formation of a large bot population.

To assess the effect of the clustering choice within RLHA, Figure 8 reports infection rate trajectories under different clustering algorithms. The results show that BIRCH yields the lowest infection rate among the compared clustering methods in this setting, suggesting that its cluster representation leads RLHA to allocate honeypots to more propagation-influential locations.

I. Clustering ablation across algorithms

The clustering module affects which nodes become centers and therefore which locations receive honeypots under the center-attached

deployment described in subsection IV-G. The infection-rate trajectories in Figure 8 indicate that clustering choice can change the effective defensive footprint even under the same honeypot budget. In our experiments, BIRCH yields the lowest infection rate among the compared clustering options, suggesting that its hierarchical summarization provides representative centers that better preserve propagation-relevant structure.

A practical way to summarize the trade-off between security and scalability is to jointly interpret the infection-rate curves and the allocation-time results in Figure 10. Clustering methods that produce favorable centers can simultaneously reduce runtime and improve propagation slowing, which is consistent with the end-to-end story in Figure 4.

The robustness of each method to different initial infection ratios is evaluated in Figure 7. As the number of initially infected devices grows, all methods face faster growth, but the RLHA curve remains below the HAAG and AHDS curves across the displayed sweep, indicating lower infection levels in those plotted settings.

J. Sensitivity to adversary foothold size

Sweeping the initial infection ratio tests whether a defense remains effective when the adversary starts with a larger foothold. In the plotted configurations, infection rates rise as the initial foothold grows, while the RLHA curve remains below the HAAG and AHDS curves across the displayed sweep. This behavior is aligned with RLHA using multiple topology-aware signals rather than relying on a single entry-point dependent abstraction, which reduces sensitivity to where the adversary starts.

The reported initial-infection sweep spans 10%–90%. The 10% setting is the lowest foothold represented in the reported experiments, while the larger ratios progressively stress the defense under substantial pre-existing compromise. Lower initial ratios such as 1%, 2%, and 5% would provide a more focused test of the very early propagation regime and are therefore identified as an additional sensitivity requirement rather than being implied by the existing sweep.

Although RLHA is designed to slow propagation, its practical objective is to protect critical assets from distributed attacks. Figure 9 reports DDoS damage under different initial infection ratios. RLHA shows lower DDoS damage than the plotted baselines across the displayed sweep; this is consistent with the mechanism by which reduced ordinary-device compromise lowers the probability that critical assets satisfy the degradation rule.

K. Scalability performance

The scalable RLHA configuration reduces sequential decision points by clustering nodes and learning only over cluster centers, as described in subsection IV-E and subsection IV-G. Figure 10 compares the required time to obtain an allocation strategy under different clustering algorithms. The results highlight the infeasibility of training without clustering on large-scale topologies and show that BIRCH yields the lowest allocation time among the compared clustering options in this experiment.

L. Runtime breakdown and scalability discussion

The scalability results can be understood by separating center-level decision making from full-network size. Without clustering, the sequential allocation would visit all nodes as decision points and repeatedly compute topology-aware quantities at network scale, which becomes prohibitive in large graphs. Clustering bounds the number of decision points to at most k centers and restricts distance-related computations to this representative set, so the overall allocation time becomes dominated by clustering and center-level feature preparation rather than by full-network sequential allocation.

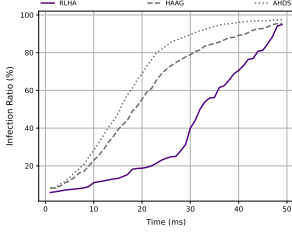


Fig. 6. Comparing RLHA with HAAG and AHDS regarding the infection rate over time.

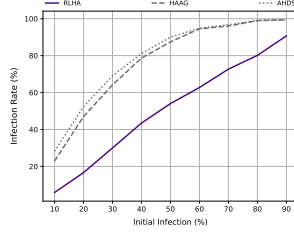


Fig. 7. Comparing RLHA with HAAG and AHDS under different initial infection ratios regarding the infection rate.

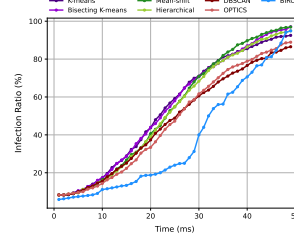


Fig. 8. Comparing different clustering algorithms used by RLHA in terms of infection rate over time.

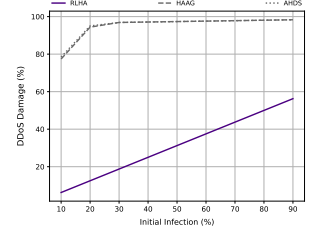


Fig. 9. Comparing RLHA with HAAG and AHDS regarding DDoS damage under different initial infection ratios.

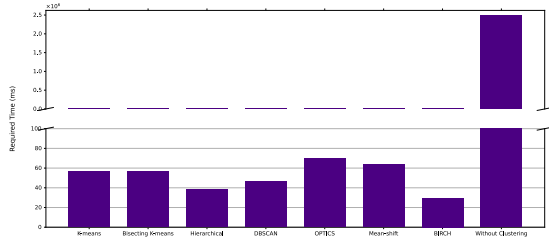


Fig. 10. Comparing different clustering algorithms used by RLHA in terms of the required time for finding an allocation strategy.

M. Extended analysis of Figures 10–12

Figures 7–10 provide complementary views of RLHA under increasing adversarial pressure and increasing scalability requirements. Figure 7 shows that RLHA maintains a lower infection rate across the full initial-infection sweep. This is important because a larger initial foothold reduces the defender’s opportunity to block individual entry points; therefore, the consistent gap indicates that RLHA is not only overfitting to a single gateway or a single initial bot location. Instead, the learned policy benefits from multiple topology-aware signals, including degree, neighbor vulnerability, and distance-based reachability.

Figure 9 connects infection control to service protection. The lower DDoS damage indicates that slowing ordinary device compromise also reduces the probability that critical assets become surrounded by enough bots to satisfy the degradation rule. This is consistent with the intended operational effect: RLHA does not merely reduce an abstract infection count, but also lowers the downstream impact on critical services. Finally, Figure 10 explains why clustering is necessary for deployment on large graphs. Without reducing the decision space, allocation time grows with the number of candidate nodes. By contrast, center-level learning bounds the sequential decision process by k , making BIRCH particularly attractive because it provides a favorable security-runtime balance in the reported experiments.

N. Learning performance

The reward shaping in RLHA depends on the learning threshold, which affects how actions are matched to state proportions inside the reward computation. Figure 11b reports infection rate for different learning threshold values and clustering options. The results show that the learning threshold can noticeably affect security performance, and that BIRCH remains among the strongest choices across the reported thresholds.

To further inspect learning behavior, Figure 11a reports the cumulative reward scores obtained by the agent under different clustering choices. The highest reward score does not always correspond to the lowest infection rate, which indicates that reward maximization and propagation slowing are aligned but not identical. This observation

motivates refining reward shaping so that the learned score more directly tracks propagation-level outcomes, especially in large-scale networks where clustering can change the state distribution.

O. Reward alignment with propagation-level outcomes

The combination of Figs. 11b and 11a shows that the training score is not a monotone surrogate for infection reduction: the clustering configuration with the highest cumulative reward is not necessarily the configuration with the lowest infection rate. This is expected from Eq. 11, because the implemented reward is assembled from local topology proportions and cluster-ordering terms, whereas infection rate depends on the global stochastic propagation process. Consequently, we do not interpret reward maximization itself as proof of security improvement. Security performance is assessed separately through infection rate and DDoS damage. A future variant may execute the propagation simulator at episode termination and feed a security-outcome term back into training, but such a variant constitutes a different training objective and requires new experiments rather than a textual relabeling of the current results.

P. Discussion and consistency with the proposed method

The results are consistent with the RLHA design presented in Section IV. First, the reported RLHA curves improve the evaluated security metrics relative to HAAG and AHDS in the plotted configurations; the method uses multiple topology-aware signals rather than a single centrality proxy. Second, the scalable configuration maintains effectiveness because clustering reduces the decision space while preserving propagation-relevant structure through representative centers, and the center-attached deployment ensures that the learned decisions translate directly to honeypot placement with bounded operational complexity. Third, the learning-threshold analysis shows that reward shaping parameters matter and should be tuned jointly with the clustering choice.

The evaluation claims are therefore descriptive unless uncertainty intervals and paired tests are explicitly shown. Repeated seeds reduce dependence on a single random placement, but repetition alone is not equivalent to reporting statistical significance. The inferential procedure required for such claims is specified in Sec. V-F.

VI. LESSONS LEARNED AND PRACTICAL IMPLICATIONS

The experiments show that, for IoT botnets, an effective defense does not always require immediate eradication. Creating early propagation friction can slow botnet growth and increase the defender’s response window. By diverting reconnaissance and exploitation attempts toward decoys, RLHA achieves a best-case observed infection-rate reduction of up to 41% under the reported configurations and reduces DDoS damage against critical assets. This maximum is not interpreted as the average effect across all configurations.

As illustrated in Fig. 11c, RLHA works as a propagation throttle by redirecting attack flows toward honeypot decoys. The figure also

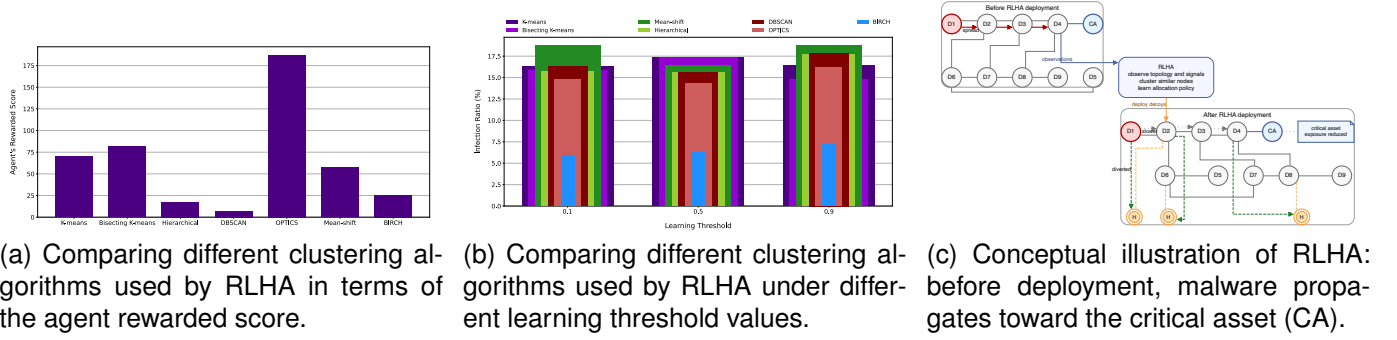


Fig. 11. RLHA clustering analysis and conceptual illustration: (a) agent rewarded score under different clustering algorithms; (b) infection-rate performance under different learning-threshold values; (c) conceptual operation of RLHA for diverting and slowing malware propagation toward a critical asset.

shows that clustering is central to the method, since center selection determines where allocation decisions are applied and shapes the achievable slowdown under the same budget.

The results also show that effective placement requires multiple propagation-relevant signals rather than a single proxy such as degree centrality. By combining vulnerability, exposure, reachability, and neighborhood information, RLHA becomes less sensitive to the adversary's initial foothold. Clustering further improves scalability, but it is also a modeling choice because different clustering methods produce different center nodes and therefore different security-runtime trade-offs. In our experiments, BIRCH provides the strongest balance.

Finally, training reward should not be treated as the only selection criterion, because high reward scores may not always imply lower infection rates. Learning parameters should therefore be tuned using operational metrics such as infection rate and DDoS damage. Future work will consider propagation-aware clustering, dynamic re-optimization of $H(\cdot)$, heterogeneous honeypot costs, constrained deployment points, and adaptive adversaries under partial observability.

A. Limitations and validity of claims

The demonstrated evidence in this paper has a deliberately bounded scope. It supports propagation-slowing behavior for the evaluated SNDlib and Stanford Gnutella connectivity proxies, the stated Mirai-like propagation rules, the reported budget/configuration settings, and the baseline honeypot-diversion abstraction. Small and medium SNDlib scenarios are instantiated in Mininet; large Gnutella scenarios are graph-level propagation simulations and should not be interpreted as large-scale Mininet testbed validation.

The following properties are *not* demonstrated by the current result set: robustness to incomplete or biased topology knowledge; dynamic device/link churn; heterogeneous honeypot costs; arbitrary administrative restrictions on attachment locations; time-varying decoy detectability or attacker fingerprinting; fully adaptive attacker policies; or propagation in which critical assets can themselves be recruited as bots. Sec. III-H defines how several of these perturbations enter the model so that they can be tested in a controlled manner. They remain robustness-validation requirements rather than established properties of RLHA.

B. Practical deployability in real-world IoT environments

Practical deployment of RLHA requires consideration of partial network visibility, heterogeneous IoT devices, constrained decoy-placement locations, and honeypot detectability. RLHA can use topology information obtained from asset inventories, passive monitoring, SDN controllers, or gateways, while feasibility masks can restrict allocation to deployable locations. Honeypots may be implemented as virtual decoys, gateway-level emulations, containerized services, or software-defined redirection targets.

Because sophisticated attackers may fingerprint decoys, practical deployment should also consider decoy diversity and service-fingerprint randomization. RLHA should therefore be viewed as a modular deception-planning layer whose effectiveness still requires validation in operational IoT testbeds with heterogeneous devices, partial topology knowledge, and deployment constraints.

VII. CONCLUSION

This paper presented RLHA, an actor-critic framework for budget-aware honeypot allocation aimed at slowing IoT botnet propagation. RLHA learns an integer-valued allocation over non-critical decision points, restricts each action by the remaining budget, updates later state information according to the selected allocation, and uses clustering to reduce large graphs to a bounded set of center-level decisions. The implemented training signal consists of topology-informed local shaping and an episode-level cluster-ordering term; infection rate and DDoS damage are evaluated separately as security outcomes rather than being treated as unreported terminal rewards.

Under the stated Mirai-like propagation and honeypot-diversion model, the reported SNDlib and Stanford Gnutella experiments show lower infection rates and DDoS damage for RLHA than for the plotted HAAG and AHDS baselines. Small and medium SNDlib scenarios are instantiated in Mininet, while the large Gnutella scenarios are graph-level propagation simulations. The maximum observed relative infection-rate reduction is 41% and should be interpreted as a best-case configuration rather than a mean or median effect. Among the seven clustering alternatives examined, BIRCH provides the most favorable security-runtime balance in the reported experiments.

The current evaluation does not establish robustness to partial topology knowledge, dynamic network changes, heterogeneous honeypot costs, constrained deployment locations, decoy detectability, adaptive attackers, or compromised critical assets. These factors are parameterized in the threat model for future evaluation. The reported results should therefore be interpreted descriptively, while stronger statistical claims require paired seed-level analyses with confidence intervals and matched significance tests. Future work will focus on outcome-aligned reward design and robustness evaluation under these operational conditions.

ACKNOWLEDGMENT

The research work is supported in part by the Federal Ministry of Research, Technology, and Space (BMFT), Germany, through the Project 6GEM+ under Grant 16KIS2411; and in part by the European Union's Horizon Europe research and innovation programme under the 6G-Path project (Grant No. 101139172).

REFERENCES

- [1] Nokia, "Nokia threat intelligence report finds malicious iot botnet activity has sharply increased," <https://www.nokia.com/about-us/news/releases/2023/06/07/>

- nokia-threat-intelligence-report-finds-malicious-iot-botnet-activity-has-sharply-increased/, 2023. [Accessed: Sep 2023].
- [2] ENISA, “Enisa threat landscape 2022,” <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022/@download/fullReport>, 2022. [Accessed: Sep 2023].
 - [3] O. Yoachimik and J. Pacheco, “Ddos threat report for 2023 q1,” <https://blog.cloudflare.com/ddos-threat-report-2023-q1/>, 2023. [Accessed: Sep 2023].
 - [4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, “Understanding the mirai botnet,” in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
 - [5] S. F. Shetu, M. Saifuzzaman, N. N. Moon, and F. N. Nur, “A survey of botnet in cyber security,” in *2019 2nd International Conference on Intelligent Communication and Computational Techniques (ICCT)*. IEEE, 2019, pp. 174–177.
 - [6] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Professional, 2002.
 - [7] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, “A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems,” *IEEE Communications Surveys and Tutorials*, vol. 23, no. 4, pp. 2351–2383, 2021.
 - [8] A. Javadpour, F. Ja’fari, T. Taleb, and C. Benzaid, “Reinforcement learning-based slice isolation against ddos attacks in beyond 5g networks,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3930–3946, 2023.
 - [9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.
 - [10] A. Javadpour, F. Ja’fari, T. Taleb, M. Shojafar, and C. Benzaid, “A comprehensive survey on cyber deception techniques to improve honeypot performance,” *Computers & Security*, vol. 140, p. 103792, 2024.
 - [11] A. Javadpour, F. Ja’fari, T. Taleb, F. Turkmen, and C. Benzaid, “Beyond reinforcement learning for network security: A comprehensive survey and tutorial,” *Journal of Information Security and Applications*, vol. 96, p. 104294, 2026.
 - [12] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “Ddos in the iot: Mirai and other botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.
 - [13] Y. M. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “Iotpot: Analysing the rise of iot compromises,” in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. USENIX Association, 2015. [Online]. Available: <https://www.usenix.org/conference/woot15/workshop-program/presentation/pa>
 - [14] H. Wang, H. He, W. Zhang, W. Liu, P. Liu, and A. Javadpour, “Using honeypots to model botnet attacks on the internet of medical things,” *Computers and Electrical Engineering*, vol. 102, p. 108212, 2022.
 - [15] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, “N-baiot: Network-based detection of iot botnet attacks using deep autoencoders,” 2018. [Online]. Available: <https://arxiv.org/abs/1805.03409>
 - [16] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, “Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset,” *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.
 - [17] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, “Ton-iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems,” *IEEE Access*, vol. 8, pp. 165 130–165 150, 2020.
 - [18] S. Garcia, A. Parmisano, and M. J. Erquiaga, “Iot-23: A labeled dataset with malicious and benign iot network traffic,” 2020. [Online]. Available: <https://zenodo.org/records/4743746>
 - [19] S. Walla and C. Rossow, “Malpity: Automatic identification and exploitation of tarpit vulnerabilities in malware,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 590–605.
 - [20] Z. Hu, Y. Hu, and B. Dolan-Gavitt, “Towards deceptive defense in software security with chaff bugs,” in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, 2022, pp. 43–55.
 - [21] L. Giarretta, A. Lekssays, B. Carminati, E. Ferrari, and S. Girdzijauskas, “Meta-soma: Decentralized and collaboratively early-stage detection of iot botnets,” 2023.
 - [22] X. Luo, Q. Yan, M. Wang, and W. Huang, “Using mtd and sdn-based honeypots to defend ddos attacks in iot,” in *2019 Computing, Communications and IoT Applications (ComComAp)*. IEEE, 2019, pp. 392–395.
 - [23] S. Wang, Q. Pei, Y. Zhang, X. Liu, and G. Tang, “A hybrid cyber defense mechanism to mitigate the persistent scan and foothold attack,” *Security and Communication Networks*, vol. 2020, pp. 1–15, 2020.
 - [24] A. B. Osei, S. R. Yeginati, Y. Al Mtawa, and T. Halabi, “Optimized moving target defense against ddos attacks in iot networks: When to adapt?” in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 2782–2787.
 - [25] W. Soussi, M. Christopoulou, G. Xilouris, E. M. de Oca, V. Lefebvre, G. Gür, and B. Stiller, “Closed-loop security orchestration in the telco cloud for moving target defense,” in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–3.
 - [26] J. Pawlick, E. Colbert, and Q. Zhu, “A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy,” *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–28, 2019.
 - [27] K. J. Ferguson-Walter, M. M. Major, C. K. Johnson, and D. H. Muhleman, “Examining the efficacy of decoy-based and psychological cyber deception,” in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 1127–1144. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/ferguson-walter>
 - [28] H. Q. Ngo, M. Guo, and H. Nguyen, “Catch me if you can: Effective honeypot placement in dynamic ad attack graphs,” 2023. [Online]. Available: <https://arxiv.org/abs/2312.16820>
 - [29] M. Osman, T. Nadeem, A. Hemida, and C. Kamhoua, “Optimizing honeypot placement strategies with graph neural networks for enhanced resilience via cyber deception,” in *Proceedings of CoNEXT 2023: The 19th International Conference on Emerging Networking Experiments and Technologies*, 2023.
 - [30] M. Zambianco, C. Facchinetti, and D. Siracusa, “A proactive decoy selection scheme for cyber deception using mitre att&ck,” *Computers & Security*, vol. 148, p. 104144, 2025.
 - [31] H. D. Hoang, T.-N. To, and N. D. Hoang Son, “Hawkeyes: An intelligent honeypot allocation strategy for cyber deception using reinforcement learning,” *Computer Networks*, vol. 276, p. 111982, 2025.
 - [32] Y. Du, Z. Song, S. Milani, C. Gonzales, and F. Fang, “Learning to play an adaptive cyber deception game,” in *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems. Auckland, New Zealand*, vol. 6, 2022.
 - [33] A. Javadpour, F. Ja’fari, C. Benzaid, 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.
 - [34] V. A. Memos, K. E. Psannis, and Z. Lv, “A secure network model against bot attacks in edge-enabled industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 7998–8006, Nov. 2022.
 - [35] C. S. Kumar, M. Bhavana, S. N. Rehman, P. Lokesh, and S. TejaPrakashReddy, “Securing iot networks against botnet attacks with machine learning and deep learning techniques,” in *2025 International Conference on Data Science and Business Systems (ICDSBS)*, Chennai, India, 2025, pp. 1–6.
 - [36] A. Javadpour, M. Ja’fari *et al.*, “A comprehensive survey on cyber deception techniques to improve honeypot performance,” *Computers & Security*, vol. 140, p. 103792, 2024.
 - [37] K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. N. Megaw, E. Nadeau, B. Piccarreta, and B. Guttman, “Considerations for managing internet of things (iot) cybersecurity and privacy risks,” NIST Interagency Report 8228, 2019. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/ir/2019/nist.ir.8228.pdf>
 - [38] M. Fagan, J. Marron, K. Brady, B. Cuthill, K. N. Megaw, R. Herold, E. Lemay, and K. Scarfone, “Iot device cybersecurity capability core baseline,” NIST Interagency Report 8259A, 2020. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8259a.pdf>
 - [39] ETSI, “Etsi en 303 645: Cyber security for consumer internet of things: Baseline requirements,” European Standard, 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_en/303600_303699/303645/02.01.01_60/en_303645v020101p.pdf
 - [40] International Society of Automation, “Isa iec 62443 series of standards: Security for industrial automation and control systems,” 2018. [Online]. Available: <https://www.isa.org/standards-and-publications/isa-iec-62443-series-of-standards>
 - [41] M. Papaioannou, M. Karageorgou, G. Mantas, V. Sucasas, I. Essop, J. Rodriguez, and D. Lymberopoulos, “A survey on security threats and countermeasures in internet of medical things (iomt),” *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 6, p. e4049, 2022.
 - [42] A. Verma, R. Saha, G. Kumar, and T.-h. Kim, “The security perspectives of vehicular networks: a taxonomical analysis of attacks and solutions,” *Applied Sciences*, vol. 11, no. 10, p. 4682, 2021.
 - [43] H. Chi, C. Fu, Q. Zeng, and X. Du, “Delay wrecks havoc on your smart home: delay-based automation interference attacks,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 285–302.
 - [44] FIRST, “Common vulnerability scoring system version 3.1: Specification document,” Forum of Incident Response and Security Teams, 2019. [Online]. Available: <https://www.first.org/cvss/specification-document>
 - [45] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessälly, “SNDlib 1.0—Survivable Network Design Library,” *Networks*, vol. 55, no. 3, pp. 276–286, 2010. [Online]. Available: <http://www3.interscience.wiley.com/journal/122653325/abstract>
 - [46] J. Leskovec and A. Krevl, “Snap datasets: Stanford large network dataset collection,” <https://snap.stanford.edu/data/>, 2014. [Accessed: Sep 2023].
 - [47] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets ’10)*, Monterey, CA, USA, Oct. 2010. [Online]. Available: <https://conferences.sigcomm.org/hotnets/2010/papers/a19-lantz.pdf>
 - [48] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The design and implementation of open vSwitch,” in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. [Online]. Available: <https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff>
 - [49] 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.