



Kent Academic Repository

Li, Haoxiang, Huang, Yuxiang, Brierley, Calvin, Wang, Yichao, Arief, Budi, Pope, James and Oikonomou, George (2026) *WiP Paper: Decentralised TinyML-Based Ransomware Containment for Resource-Constrained IIoT Networks*. In: EWSN '26: Proceedings of the 2026 International Conference on Embedded Wireless Systems and Networks. . pp. 194-199. International Conference on Embedded Wireless Systems and Networks (EWSN)

Downloaded from

<https://kar.kent.ac.uk/116435/> The University of Kent's Academic Repository KAR

The version of record is available from

<https://doi.org/10.3217/n14h-pd76>

This document version

Author's Accepted Manuscript

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in **Title of Journal** , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our [Take Down policy](https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies) (available from <https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies>).

WiP Paper: Decentralised TinyML-Based Ransomware Containment for Resource-Constrained IIoT Networks

Haoxiang Li
University of Bristol
United Kingdom
haoxiang.li@bristol.ac.uk

Yuxiang Huang
University of Bristol
United Kingdom
ethan.huang@bristol.ac.uk

Calvin Brierley
University of Kent
United Kingdom
c.r.brierley@kent.ac.uk

Yichao Wang
University of Kent
United Kingdom
y.wang-301@kent.ac.uk

Budi Arief
University of Kent
United Kingdom
b.arief@kent.ac.uk

James Pope
University of Bristol
United Kingdom
james.pope@bristol.ac.uk

George Oikonomou
University of Bristol
United Kingdom
g.oikonomou@bristol.ac.uk

Abstract

Ransomware is emerging as a real threat to Industrial Internet of Things (IIoT) networks, where resource-constrained devices cannot run traditional endpoint protections. Such ransomware spreads silently through normal data-plane traffic before activating, and once infection reaches network scale, recovery costs become prohibitive, making containment during propagation essential. Existing intrusion response systems rely on centralised controllers whose multi-hop command latency is unable to keep pace, while decentralised execution must cope with imperfect on-device detection and purely local observations. We present **Decentralised Containment System (DConS)**, a light-weight intrusion response system in which each IIoT node autonomously contains ransomware through on-device TinyML inference using only local and one-hop neighbour observations. Experimental results show that DConS can contain propagation, even when detector's true-positive rate drops to 0.70, and when propagation outpaces the response cycle, while restoring up to 65% of the connectivity disrupted by containment. The entire system fits within ~7.2 KB Flash and ~470 B RAM, demonstrating the feasibility of our proposed solution.

CCS Concepts

• **Security and privacy** → *Mobile and wireless security*; • **Computer systems organization** → **Embedded systems**; • **Computing methodologies** → *Neural networks*.

Keywords

Ransomware Containment, IIoT, resource-constrained device, TinyML, MARL

1 Introduction

Industrial Internet of Things (IIoT) deployments increasingly rely on Wireless Sensor Networks (WSN) of resource-constrained devices that self-organise into tree-based routing topologies via protocols such as Routing Protocol for Low-Power and Lossy Networks (RPL) [1]. RPL organises nodes into a tree rooted at the border

router, where each node forwards traffic toward the root through a preferred parent. Such devices cannot run traditional endpoint protections, which far exceed IoT microcontroller unit (MCU) budgets. Prior work has shown that ransomware can propagate inconspicuously across RPL networks through normal data packets [2]. Once infection reaches network scale, recovery costs become prohibitive, making containment during the propagation phase essential.

Existing Intrusion Response Systems (IRS) typically adopt centralised architectures [3] whose multi-hop command latency cannot keep pace with rapid propagation. On the other hand, decentralised execution typically requires each node to act on its own state and one-hop neighbours under noisy detection, demanding policies that balance infection suppression and network utility [4].

To address these issues, we propose **Decentralised Containment System (DConS)**, a fully decentralised IRS that leverages TinyML [5] to enable each resource-constrained node to autonomously contain ransomware infection through rapid on-device inference, without a central controller. DConS performs two operations on each node: (i) progressive communication containment along RPL's tree structure to block ransomware propagation paths, and (ii) reparent triggering to repair connectivity disrupted by containment. Under tight resource constraints, fast propagation, and noisy infection detection, DConS provides strong containment while incurring lower connectivity overhead. Learning such on-device policies under partial observability is non-trivial. We address this by learning policies offline under Centralised Training with Decentralised Execution (CTDE) [6], where a centralised critic uses global infection state during training to guide each node's locally-observable policy toward coordinated behaviour. We then compress the trained policies for IoT MCUs: distillation maps them to smaller student networks using deployment-available features, followed by int8 quantisation [7] for integer-only inference within KB-scale memory.

Our paper's primary contributions are:

- We design DConS, a light-weight, fully decentralised intrusion response system that enables cost-effective ransomware containment through purely local TinyML inference on resource-constrained IIoT devices.

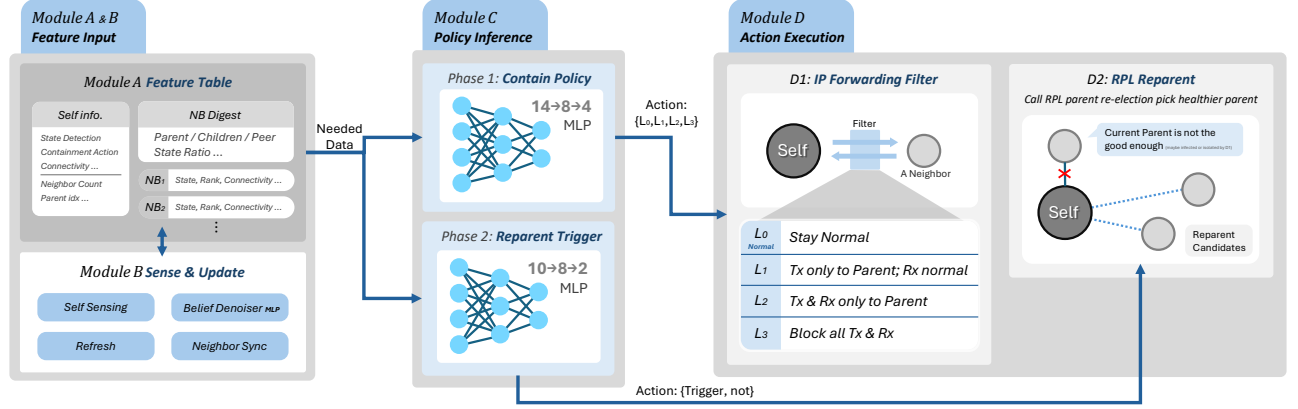


Figure 1: System architecture of DConS. Modules A and B aggregate local and neighbour features into a unified Feature Table; Module C performs two-stage TinyML inference for containment level selection (Phase 1) and reparent triggering (Phase 2); Module D executes the resulting actions via IP forwarding filters and RPL local repair.

- We present a CTDE-to-MCU training pipeline that learns coordinated containment policies under partial observability and distils them into compact int8 student models suitable for on-device inference.
- We provide experimental evidence that DConS is able to contain propagation, even when detector’s true-positive rate drops to 0.70, and when propagation outpaces the response cycle, recovering up to 65% of disrupted connectivity, and fitting within 7.2 KB Flash and 470 B RAM.

2 Related Work

Ransomware has evolved from encrypting IT systems to disrupting operational technology, and is increasingly projected to target the devices of IIoT systems [8]. A recent prototype by Huang et al. [2] establishes IIoT ransomware as a concrete near-term threat. It propagates autonomously through embedded payloads in normal data packets on IEEE 802.15.4 devices, with no observable changes in traffic, CPU, or power. Since propagation evades all conventional detection signals, on-device detection is critical. Centralised architectures such as SVELTE [9] suffer from multi-hop latency. Huang et al. [10] proposed a light-weight micro-auditing solution that provides viable on-device detection signals. However, the inherent stealthiness of such ransomware limits even specialised detectors to imperfect accuracy, underscoring the need for autonomous containment that remains effective under noisy detection.

Containment strategies remain largely under-explored. Bashendy et al. [11] found that existing IRS measures rely on either manual operation or hard-coded actions. Recent reinforcement learning (RL)-based approaches [4, 12, 13] have advanced adaptive response but uniformly assume centralised controllers on server-class hardware, while SDN-based isolation [14] faces the same latency bottleneck. RLtools [15] has demonstrated deep RL inference on MCU-class hardware with sub-millisecond latency, but this has been limited to robotics control. To our knowledge, no prior work has successfully deployed RL-based response policies on resource-constrained IIoT nodes for decentralised, autonomous ransomware containment.

3 System Design

Figure 1 provides an overview of DConS, organised as four modules across feature input, policy inference, and action execution, executed as two sequential phases per cycle. Phase 1 *Containment Policy* selects a progressive data-plane restriction level on the node’s traffic, and Phase 2 *Reparent Trigger* decides whether to invoke RPL parent re-election when containment isolates the node from its parent or the parent itself is infected. It provides a topology-level action that lies outside Phase 1’s per-link action space.

Both phases read from a unified on-device Feature Table (Module A), maintained by the Sense & Update module (Module B) which acquires local detector signals, denoises them, and synchronises neighbour state (see details in Section 3.1). The Policy Inference module (Module C, Section 3.2) executes the two phases as two int8-quantised Multi-Layer Perceptron (MLP) networks. Their outputs are translated by the Action Execution module (Module D, Section 3.3) into IP-layer forwarding filter rules and `rpl_local_repair()` invocations.

3.1 Feature Table and Sensing Pipeline

The inputs for models in Module C come from a unified on-device Feature Table (Module A), shown in the left hand side of Figure 1. This table aggregates the node’s own infection state, current containment level, topological position, and its neighbours’ state detection signals and connectivity features. During inference, the models only read from the table to assemble their input vectors and do not perform any feature aggregation or computation themselves. Updates to the Feature Table are also decoupled from model inference, allowing different pieces of information to be acquired asynchronously. The Feature Table is maintained by the Sense & Update module (Module B), which performs local information acquisition, detection-signal correction, neighbour information synchronisation, and data preprocessing.

The Feature Table captures three views the policies need: (i) the node’s own status: its current infection belief, Destination-Oriented Directed Acyclic Graph (DODAG) rank, and the containment level it is currently applying; (ii) a bounded summary of each known

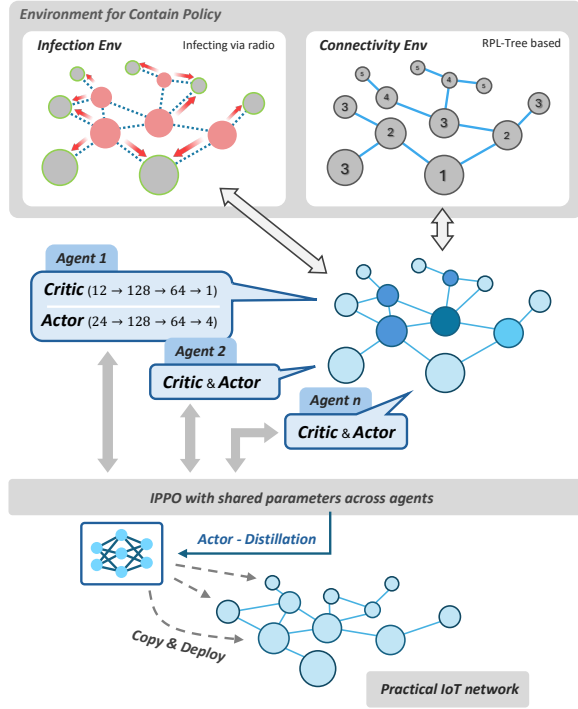


Figure 2: CTDE MARL training architecture: agents share Actor/Critic weights and interact with a dual-component environment modelling both infection propagation and RPL-tree connectivity.

neighbour: infection state, rank, the containment level they are applying, and connectivity flags; and (iii) a small set of pre-computed aggregates over the neighbour set: parent infection state, mean and ratio of children’s infection states, and whether the parent link is currently blocking this node. These features are exactly those a node can obtain locally or from one-hop neighbour exchange, requiring no global state at inference, while still covering the three signals a containment decision needs: own infection evidence, neighbour infection pressure, and the parent/children relations that determine which links a containment action cuts.

The parent is referenced by an index into the neighbour array. Children and other neighbours are tracked with two 16-bit bitmasks, so an RPL topology change costs only an index or bit update.

Within each cycle, Module B refreshes the Feature Table by reading the local infection detector, refreshing topological information from the RPL stack, synchronising a state digest with one-hop neighbours, and computing aggregate statistics over the neighbour set such as neighbour’s infected ratio. Topology refresh and aggregate computation are routine. The rest of this subsection focuses on detector denoising and neighbour information synchronisation.

We assume each node is equipped with a binary infection detector, whose design is out of scope. Its output is timely but noisy, and feeding raw detections into the policy would let single-tick errors drive containment decisions. Module B therefore produces two additional signals from the raw stream, each targeting a different error mode:

- A *temporal belief* that smooths out single-tick flips by accumulating evidence over time, so a transient false positive does not immediately escalate containment. It is implemented as a Bayesian log-odds accumulator $L_t = \lambda L_{t-1} + \text{evidence}(d_t)$, where $\text{evidence}(d_t)$ is the detection’s log-likelihood ratio and $\lambda = 0.9$ gives an effective memory of $\approx 1/(1 - \lambda) = 10$ cycles, mapped to an 8-bit probability via a sigmoid lookup table.
- A *spatially denoised belief* that corrects for the case where a node’s own detector disagrees with its neighbourhood — e.g. a lone positive amid healthy neighbours is more likely a false positive than a real infection. A $6 \rightarrow 4 \rightarrow 1$ MLP (BeliefDenoiser, 33 parameters), supervised at training time, takes the node’s own and neighbours’ beliefs and outputs a corrected estimate.

Three signals—raw detection, temporal belief, and spatial denoised—are simultaneously provided to the inference model, and the policy learns which signal to weight under different conditions.

Neighbours’ beliefs and connectivity flags are exchanged via dedicated TSCH MAC-layer broadcast frames in designated slots each cycle. Because TSCH exchange operates at the MAC layer, even a node with full data-plane isolation can still send and receive exchange frames normally. Meanwhile, the TSCH exchange channel does not constitute a ransomware propagation path: The 3-byte payload uses fixed-length positional reads with no executable content, and MAC-layer frames are processed by the radio driver before reaching the IP stack.

3.2 Policy Inference

Module C runs the two decisions per cycle: C1 picks a containment level ($14 \rightarrow 8 \rightarrow 4$ MLP, 156 parameters), and C2 decides whether to trigger reparenting ($10 \rightarrow 8 \rightarrow 2$ MLP, 106 parameters). The input and output sizes follow from the feature set and the action space, while a single narrow hidden layer suffices to reproduce the teacher policy within the MCU budget.

We face two design challenges. First, the right containment action at a node depends on what its neighbours simultaneously decide. Over-containment in a cluster fragments the network, whereas under-containment lets propagation through. So we cannot train each node independently. Second, a node at deployment only sees its own and one-hop neighbour state, but learning a globally sensible policy from purely local observations is hard without some form of training-time supervision. We address these challenges with a Multi-Agent Reinforcement Learning (MARL) [16] formulation trained under CTDE [6], as shown in Figure 2.

3.2.1 Training. We train in a Gymnasium-based simulator [17] of an RPL network under ransomware intrusion, with infection following a Susceptible-Infected (SI)-like model: in an IIoT network composed of a 50-node RPL topology, 20% of nodes are randomly selected as initial infections, and the ransomware propagates each round to direct neighbours within its radio range. Each node is equipped with a binary detector that outputs real-time infection states, but these results are noisy.

The training is performed under Independent Proximal Policy Optimization (IPPO) [18], with shared parameters across agents. Training each agent on its deployment-time observation alone

yields high-variance value estimates and slow learning. CTDE addresses this by giving the Critic access to privileged global state during training (noise-free infection states and the full connectivity graph), so that value estimates accurately guide the Actor toward globally coordinated behaviour. At deployment the Critic is discarded and only the locally-observable Actor remains.

The Actor receives 24-dimensional local observations. In addition to the features provided by the Feature Table during deployment, auxiliary features such as subtree size and average neighbour rank are included during training to enrich the policy’s topology awareness. The reward comprises three categories of signals. *Containment signals* penalise new infection events and malicious packet propagation, and reward the proportion of healthy nodes at the end of an episode; *Connectivity signals* reward the maintenance of up-link and downlink connectivity; *Operational Cost signals* penalise over-containment of healthy neighbours and frequent switching of containment levels.

With the Phase 1 Containment Actor trained, we then train the Phase 2 reparent policy. The trained Containment Actor is frozen and embedded in the environment to execute containment actions. The Phase 2 model jointly learns two decisions: which neighbour would make the best replacement parent, and whether the switch is worthwhile. To handle variable neighbour counts, it uses a Pointer Network that scores each of K neighbours from the node’s 24-dimensional observation and the neighbour’s 9-dimensional features; a switch is triggered only when the top-scoring candidate beats the current parent by a margin. The reward credits connectivity gains from the switch.

3.2.2 Distillation. The teacher models exceed the deployment budget of MCUs and must be distilled into minimal student models via behaviour cloning. The C1 student takes as input a 14-dimensional subset of the teacher’s 24-dimensional observations, retaining only features that are readily obtainable on resource-constrained devices. It reproduces the teacher’s containment decisions with 96.1% behaviour cloning accuracy. The C2 student splits the teacher’s joint WHEN+WHO decision: the student only learns when to trigger reparent (WHEN), while the selection of which new parent (WHO) is delegated to RPL’s default `rpl_local_repair()` mechanism, thereby simplifying the variable-length Pointer Network into a fixed 10-dimensional input MLP and significantly reducing parameter count and inference complexity. It captures the teacher’s reparent timing with a recall of 0.95.

3.2.3 Deployment. Training under IPPO with parameters shared across agents yields a single Actor that all nodes execute. Since this Actor reads only a node’s own state and its one-hop neighbour features, it is independent of the network’s size and topology, and one set of weights serves a network of any scale. We therefore deploy it by distillation and direct transfer, as illustrated at the bottom of Figure 2: the distilled Actor is compiled into each node’s firmware as frozen constant weights, and every node runs the same fixed-weight inference locally. At run time, each node repeats a fixed per-cycle sequence: sensing, a neighbour exchange, Phase 1 containment, a second exchange, and Phase 2 reparenting.

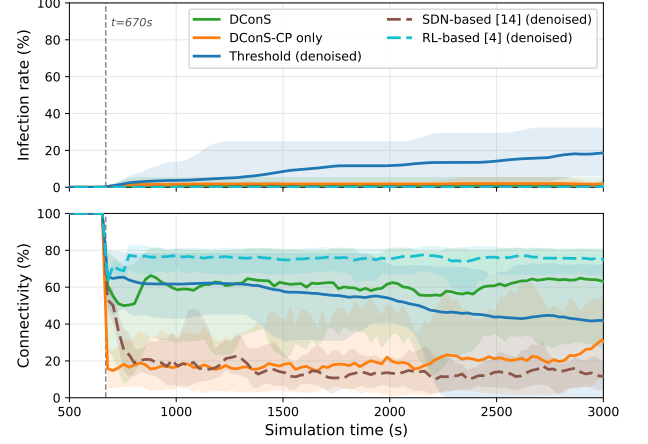


Figure 3: Temporal dynamics at TPR = 0.80. Containment is activated at $t=670$ s (dashed line), by which time 20% of nodes have already been infected.

3.3 Action Execution

Module D translates the inference outputs of Module C into actual network actions. D1 installs hooks on the IPv6 forwarding path that compare each packet’s peer address against the cached parent and apply ACCEPT/DROP according to the current containment level. All levels key on the parent address, which RPL already maintains, avoiding additional per-neighbour state. The four levels follow a progressive restriction:

- L0: all data-plane traffic flows normally.
- L1: Tx to parent only; Rx unrestricted.
- L2: Tx and Rx with parent only.
- L3: all data-plane traffic blocked.

ICMPv6 control messages (DIO/DAO) are consumed by the RPL engine before reaching the filter point, and are therefore unaffected by any containment level.

D2 actively invokes RPL’s `rpl_local_repair()` based on the Reparent Trigger output, allowing the node to switch parents in response to containment-induced isolation or parent infection without waiting for the next scheduled DODAG maintenance round.

4 Experiments and Results

All experiments are conducted on the Cooja simulator running full Contiki-NG [19] firmware, using a 49-node IIoT lab topology ($80\text{ m} \times 60\text{ m}$, graph density $\approx 12.6\%$). Ransomware propagation is modelled after the PoC of [2], in which malicious code is embedded within application-layer data transmissions and spreads to wireless neighbours. On-device detection is abstracted as a noisy binary detector parameterised by true-positive rate (TPR) and false-positive rate (FPR). We compare against a Threshold baseline that shares the same temporal belief tracker and issues containment by applying fixed thresholds to the local detection only. We further compare against two representative centralised approaches adapted to our setting: an SDN-based scheme [14] and a RL-based [4]. Both share the same denoised detection input as DConS but execute their response decisions through a central controller rather than locally. We also include a variant, DConS-CP only, which retains only

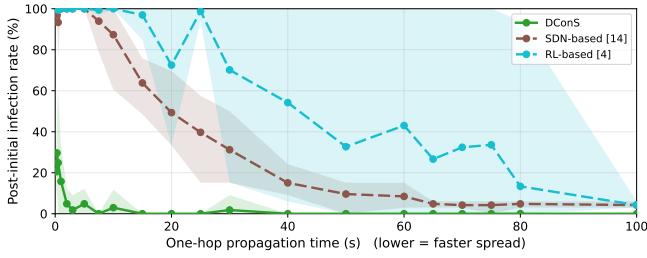


Figure 4: Infection rate vs. one-hop propagation time. Lower x-values mean faster propagation. (TPR = 0.90, FPR = 0.10, 20% initial infections).

the Phase 1 containment policy and disables Phase 2 reparenting, isolating the contribution of topology repair.

4.1 Performance evaluation

Figure 3 illustrates the temporal behaviour of the five policies at a representative operating point. In our experiment, DConS is initially disabled and activated network-wide once any node first detects an infection. Activation occurs at $t = 670$ s (dashed line), with 20% already infected. We report the *post-initial infection rate*, the fraction of newly infected nodes after activation among those still susceptible at that moment, measuring DConS’s ability to contain propagation. We also track *connectivity*, the fraction of non-root nodes whose UDP traffic reaches the RPL root, the end-to-end utility that containment directly affects. After activation, all methods except the Threshold baseline contain propagation. They differ sharply in the connectivity they preserve: the centralised RL-based policy retains the most ($\sim 75\%$). Full DConS attains connectivity ($\sim 60\%$) using only local, on-device inference, and clearly outperforms its DConS-CP only variant and the SDN-based baseline.

4.1.1 Robustness to propagation speed. Figure 4 reports the new infections accumulated by the end of a 3,000 s simulation, as the ransomware’s one-hop propagation time is varied (smaller values on the left denote faster spread). DConS contains propagation across almost the entire range, rising above zero only under the most extreme spread and returning to near-complete containment thereafter. The centralised baselines contain infection only when propagation is slow and collapse as it accelerates. Each decision must travel up to a central controller and back before a response takes effect, and this round trip is slow enough for the ransomware to spread further in the meantime. DConS instead decides and acts locally, so it can run at a much higher frequency and react before propagation outpaces it.

4.1.2 Robustness to detector quality. Figure 5 shows performance across a range of detector operating points (TPR $\in [0.60, 1.00]$, FPR $\in [0.00, 0.16]$). To isolate the effect of detector quality from propagation speed, these experiments use a relatively slow propagation setting (one-hop time $\gg$ execution cycle). At TPR ≥ 0.90 all methods suppress infection. The differences emerge at lower TPR. The threshold baseline collapses at TPR ≤ 0.80 , and the RL-based baseline, although strong under accurate detection, breaks down once TPR ≤ 0.70 . DConS limits infection to at most 6% at TPR $\geq$

Table 1: DConS subsystem resource footprint

Component	Flash [†]	Static RAM [‡]
Feature Table (Module A)	948 B	120 B
Table updater (Module B)	1,284 B	—
BeliefDenoiser (33 params)	237 B	—
C1 Containment (156 params)	376 B	—
C2 Reparent (106 params)	294 B	—
IP filter (Module D)	299 B	3 B
Neighbour sync	475 B	36 B
Main process	3,465 B	314 B
DConS total	~ 7.2 KB	~ 470 B

[†] Flash: Non-volatile storage of model-related parameters and constants placed in `.text/.rodata`. [‡] Static RAM: Temporary memory footprint during model inference, used for intermediate variables and buffers in `.bss/.data`.

0.70, and degrades only to 18–22% at TPR = 0.60, remaining the most robust to detector noise.

The connectivity heatmaps expose the cost each method pays for this containment. The SDN-based baseline contains infection at every operating point, but only by isolating nodes so aggressively that connectivity is pinned near 15% throughout. The RL-based baseline preserves high connectivity when detection is accurate, yet loses it together with containment once TPR drops. DConS avoids both failure modes: it sustains 37–68% connectivity while keeping infection low, and the gap over DConS-CP only (6–45%) confirms that Phase 2 reparenting restores paths disrupted by containment. False-positives manifest as connectivity loss from over-containment, not as containment failure. DConS-CP only’s connectivity therefore degrades with FPR, whereas DConS offsets this cost through re-parenting and remains FPR-insensitive.

4.2 Resource Footprint

The entire DConS subsystem adds approximately 7.2 KB of Flash and 470 B of RAM over the Contiki-NG protocol stack baseline, with per-module breakdown shown in Table 1. The three int8 models contain 295 parameters in total (33, 156, and 106 for BeliefDenoiser, C1, and C2 respectively) and require only 268 multiply-accumulate operations per inference. DConS is therefore light enough for typical IoT MCUs.

5 Conclusion and Future Work

We presented DConS, a decentralised intrusion response system that contains ransomware in resource-constrained IoT networks through on-device TinyML inference. Each node decides and acts using only its own state and one-hop neighbour observations. DConS pairs a containment policy with an RPL reparent trigger, both trained with CTDE-based multi-agent reinforcement learning and distilled for resource-constrained MCUs. DConS contains propagation under noisy detection and fast infection speed, while Phase 2 reparenting recovers most of the connectivity that containment costs. The entire system fits within ~ 7.2 KB Flash and ~ 470 B RAM.

Several directions remain for future work. First, we plan to replace the abstract binary detector with concrete on-device detectors and to evaluate DConS under correlated or biased detection errors

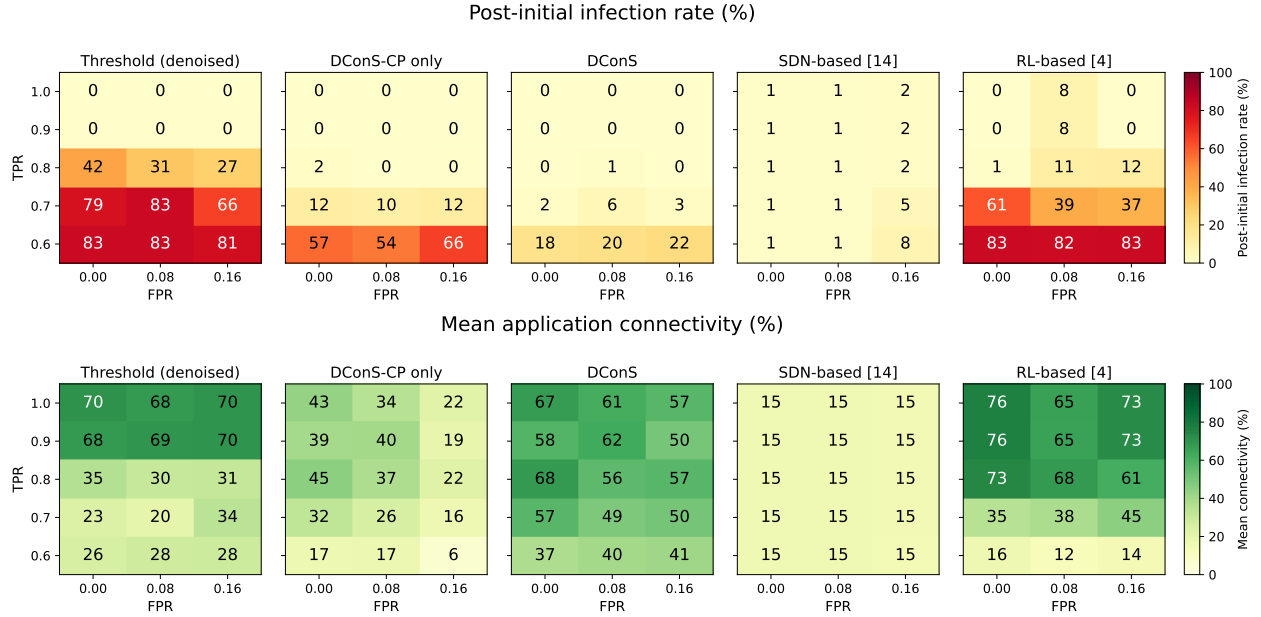


Figure 5: Sensitivity to detector quality. Top: post-initial infection rate; bottom: mean application connectivity. Columns from left to right: Threshold (denoised), DConS-CP only, DConS, SDN-based [14], RL-based [4]. Colour encodes metric value across TPR (y-axis) and FPR (x-axis).

rather than the independent and identically distributed noise assumed here. Second, containment outcomes are strongly influenced by where the initial infected nodes are located in the RPL tree; root-adjacent infections are substantially harder to contain than leaf-adjacent ones. Thus, we plan a systematic study of infection placements, complementing the randomised placements used in our current evaluation.

Acknowledgments

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC), Grant Numbers EP/X036871/1 and EP/X036707/1.

References

- [1] Montdher Alabadi, Adib Habbal, and Xian Wei. 2022. Industrial Internet of Things: Requirements, Architecture, Challenges, and Future Research Directions. *IEEE Access* 10 (2022).
- [2] Yuxiang Huang, Calvin Brierley, Adel ElZemity, James Pope, Jiteng Ma, Antonio Di Buono, Budi Arief, and George Oikonomou. 2025. Ransomware in Resource-Constrained Industrial IoT Networks: There Actually is a Threat. In *Proc. DCOSS-IoT*. 763–770.
- [3] M. Surendar and A. Umamakeswari. 2016. InDReS: An Intrusion Detection and response system for Internet of Things with 6LoWPAN. In *2016 Int'l Conf. on Wireless Communications, Signal Processing & Networking (WiSPNET)*. 1903–1908.
- [4] Kim Hammar and Rolf Stadler. 2024. Learning Near-Optimal Intrusion Responses Against Dynamic Attackers. *IEEE Transactions on Network and Service Management* 21, 1 (2024), 1158–1177.
- [5] Nikolaos Schizas, Aristeidis Karras, Christos Karras, and Spyros Sioutas. 2022. TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review. *Future Internet* 14, 12 (2022).
- [6] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.
- [7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Procs. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [8] Muna Al-Hawawreh, Mamoun Alazab, Mohamed Amine Ferrag, and M. Shamim Hossain. 2024. Securing the Industrial Internet of Things against ransomware attacks: A comprehensive analysis of the emerging threat landscape and detection mechanisms. *Journal of Network and Computer Applications* 223 (2024), 103809.
- [9] Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks* 11, 8 (2013), 2661–2674.
- [10] Yuxiang Huang, Calvin Brierley, James Pope, Jiteng Ma, Antonio Di Buono, Budi Arief, and George Oikonomou. 2025. Micro-Auditing for Ransomware Detection in Resource-Constrained Industrial IoT Networks. In *The 4th Workshop on Security and Privacy in Connected Embedded Systems*. Association for Computing Machinery (ACM).
- [11] May Bashendy, Ashraf Tantawy, and Abdelkarim Erradi. 2023. Intrusion response systems for cyber-physical systems: A comprehensive survey. *Computers & Security* 124 (2023), 102984.
- [12] Trung V. Phan and Thomas Bauschert. 2022. DeepAir: Deep Reinforcement Learning for Adaptive Intrusion Response in Software-Defined Networks. *IEEE Transactions on Network and Service Management* 19, 3 (2022).
- [13] Hao Chen, Yingxu Lai, Jing Liu, and Hanxiao Wanyan. 2024. Interpretable Cross-Layer Intrusion Response System Based on Deep Reinforcement Learning for Industrial Control Systems. *IEEE Transactions on Industrial Informatics* 20, 7 (2024), 9771–9781.
- [14] David Candal-Ventureira, Pablo Fondo-Ferreiro, Felipe Gil-Castañeira, and Francisco Javier González-Castaño. 2020. Quarantining Malicious IoT Devices in Intelligent Sliced Mobile Networks. *Sensors* 20, 18 (2020).
- [15] Jonas Eschmann, Dario Albani, and Giuseppe Loianno. 2024. RLtools: A Fast, Portable Deep Reinforcement Learning Library for Continuous Control. *Journal of Machine Learning Research* 25, 301 (2024), 1–19.
- [16] Lucian Buesoni, Robert Babuska, and Bart De Schutter. 2008. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 2 (2008), 156–172.
- [17] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, et al. 2024. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032* (2024).
- [18] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviychuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. 2020. Is independent learning all you need in the StarCraft multi-agent challenge? *arXiv preprint arXiv:2011.09533* (2020).
- [19] George Oikonomou, Simon Duquenoey, Atis Elsts, Joakim Eriksson, Yasuyuki Tanaka, and Nicolas Tsiftes. 2022. The Contiki-NG Open Source Operating System for Next Generation IoT Devices. *SoftwareX* 18 (2022), 101089.