Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.DOI

Cognitive and Scalable Technique for Securing IoT Networks Against Malware Epidemics

SAI MANOJ P D[1], (Member, IEEE), XIAOJIE GUO[2], HOSSEIN SAYADI[3], (MEMBER, IEEE), CAMERON NOWZARI[1], (MEMBER, IEEE), AVESTA SASAN[1], (MEMBER, IEEE) SETAREH RAFATIRAD[2], (MEMBER, IEEE) LIANG ZHAO[2], (MEMBER, IEEE), AND HOUMAN HOMAYOUN[4], (SENIOR MEMBER, IEEE)

[1]Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA (e-mail: {spudukot,cnowzari,asasan}@gmu.edu)
[2]Department of Information Science and Technology, George Mason University, Fairfax, VA 22030 USA (e-mail: {xguo7,srafatir,lzhao9}@gmu.edu)
[3]Department of Computer Engineering and Computer Science, California State University, Long Beach, CA 90840 USA (e-mail: hossein.sayadi@csulb.edu)
[1]Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA (e-mail: hhomayoun@ucdavis.edu)

Corresponding author: Sai Manoj P D (e-mail: spudukot@gmu.edu).

ABSTRACT The sheer volume of IoT networks being deployed today presents a major "attack surface" and poses significant security risks at a scale never encountered before. In other words, a single IoT device/node that gets infected with malware has the potential to spread the malicious activities across the network, eventually ceasing the network functionality or compromising the network. Simply detecting and quarantining the malware in IoT networks does not guarantee preventing malware propagation. On the other hand, use of traditional control theory for malware confinement is not effective, as most of the existing works do not consider real-time malware control strategies that can be implemented using uncertain infection information from the nodes in the network or have the containment problem decoupled from network performance. In response, in this work, we propose a two-pronged approach with malware detection at node-level, and confinement of malware at network-level. We deploy a recently proposed lightweight runtime malware detector at the node-level that employs Hardware Performance Counter (HPC) values for malware detection. This node-level malware information is combined with the malware propagation information and then fed during runtime to a stochastic predictive controller to confine the malware propagation without hampering the network performance. Synthesizing the node-level malware information with the model predictive containment strategy leads to achieving an average network throughput of nearly 200% of that of IoT network without any defense, and up to 160% of that of network with commonly employed state-of-the-art heuristic approaches for malware confinement. Furthermore, to scale with ever-increasing network topology sizes, we introduce a novel multi-attribute graph translation that can predict the network topology and node state information when provided with a snapshot of topology and node-level malware infection. The proposed multi-attribute graph translation has $<$5.88 Root Mean Square Error (RMSE) compared to the model predictive containment strategy and has shown nearly constant graph translation time and limited resource utilization independent of the network size.

INDEX TERMS Malware epidemics; Control theory; Malware detection; IoT security; Malware confinement

I. INTRODUCTION

The grand vision of the Internet-of-Things (IoT) boasts a fully connected global network connecting every imaginable thing together. The benefits of such a vision are currently seen spanning across many application domains including mobile, health, smart homes, smart grids, and defense. Amelioration of miniature computing devices into the consumer and industrial markets with enabled connectivity to the Internet towards smart and intelligent features lead to an upsurge in the size of networks through which the devices are linked and

communicate with each other [1], [2]. Unfortunately, with the massive amount of potential benefits offered by IoT devices, there comes an equal amount of potential vulnerabilities and security risks that have never seen before [3], [4], [5], [6], [7], as the security is often neglected in IoT devices' design [8], [9], [3].

From the adversaries' perspective, the feasibility for malware[1] propagation via connected network with none/weakly built defense measures, and a vast connectivity makes the IoT devices a potential target for cyber-attacks [3], [4], [10], [5], [6], [7], [11], [12]. These attacks can be targeted at various IoT devices such as routers, surveillance cameras, and mobile phones. It is highly possible that the malware spreads across the network as soon as one of the devices/nodes is infected, and the entire network immediately becomes compromised [13], [14], [9]. In April 2017, an attack named 'Bricker-Bot' was launched in the USA, where IoT devices such as routers stopped functioning for the users who never changed their default 'usernames' and 'passwords' for their devices. Similar attack occurred in Germany with the hacking of Deutsche Telekom network in November 2016, resulted in widespread Internet blackout for three days. The amount of attacks on IoT devices are augmenting tremendously. According to latest McAfee threat reports [8], there is nearly 38% increase in new malware in the year 2018 (more than 800 million samples in 2018), and nearly 15% of the existing enterprises being attacked [15]. Unfortunately, given the magnitude of deployed and emerging IoT networks, it becomes impractical to quarantine the infected systems as the malware would have propagated to other devices already. More specifically, the accentuating size and popularity of these networks further exacerbates the challenge of securing IoT devices and restricting the malware propagation, as we no longer have the option to just 'restart' the entire system, as the cost of restarting a massive network may easily exceed the cost of potential malware existing in the network. The propagation of malware through the IoT networks not only leads to infecting multiple IoT devices, but also can significantly degrade the network performance such as throughput. Preserving the network connectivity for the functionality and communication can be seen as potential threat in IoT networks leading to spread of malware in IoT network and edge devices [16], [17], [18], [19], [8]. On the contrary side, it is non-trivial to maintain the connectivity (throughput) to facilitate the communication and preserve the functionality while proposing effective solutions to improve the security and/or performance. Thus, detecting threats and minimizing the traffic/network access from the compromised nodes is non-trivial to protect the users from different cyber-threats including DDoS [16], [17], [18], [19], [20].

Coupling the above-discussed issues of malware attacks on a single IoT node, and their propagation through the network reveals some significant issues that have to be addressed before realizing large-scale global deployments of IoT networks. In this work, we propose a unified solution that addresses the challenge of malware detection and propagation in an IoT network by: a) deploying an effective yet lightweight runtime malware detection on IoT devices; and b) confining the propagation of malware in the IoT network with imperfect infection obtained from node-level malware information while preserving the network connectivity and overall performance. Albeit, the malware confinement can be performed effectively using a stochastic optimal control technique, the scalability is limited to networks with few tens of nodes. To further improve it, a novel multi-attributed graph translation method is proposed based on multi-attributed graph translation generative adversarial nets.

A. ASSOCIATED RESEARCH CHALLENGES

Solving the aforementioned problem involves multiple research challenges that are highlighted in this section.

1) Limited Resource Availability

Unlike general network systems, IoT devices are designed only with fundamental cyber-physical functions in mind such as sensing and actuation with minimal computational, storage, and communication capabilities. As such, carrying out compute-intensive operations for malware detection or storing the malicious patterns to detect cyber-attacks during runtime are impractical in these devices. For traditional computing systems, several techniques have been explored for malware detection including dynamic binary instrumentation [21], anomaly detection [22], information flow tracking [23], [24], [25], taint-analysis and symbolic execution [26], and VM introspection [27]. There also exist traditional approaches such as semantic [28], [29], [30] and signature-based [31], [32], [33] solutions including off-the-shelf anti-viruses as well. However, most of these techniques are slow, and require large computational resources and memory [34], [35], [36], [37], making them infeasible to be adopted in IoT and resource constrained devices. Furthermore, the emergence of new malware threats often requires patching or updating off-the-shelf software-based malware detection solutions (such as anti-virus) and incurs a large amount of memory, hardware resources, as well as network communication bandwidth. Therefore, IoT devices in general cannot accommodate resource intensive solutions, thus are vulnerable to security threats. As such, it is crucial to deploy a lightweight malware detector for resource constrained IoT systems. In response to the latency and processing overheads incurred by existing malware detectors, Hardware-assisted Malware Detection (HMD) technique[2] is proposed [35]. *In this work, we adopt a recently proposed lightweight HMD solution [36] on IoT nodes to detect the malware.* It needs

[1]Malware, also known as malicious software, is a piece of code designed to perform various malicious activities, such as destroying or manipulating the data, stealing information, running destructive or intrusive programs on devices to perform Denial-of-Service (DoS) attack, and gaining root access without the consent of user.

[2]In HMD, signatures of the underlying hardware events are utilized to detect malicious applications (malware).

to be noted that HMD is solely adopted based on the experimental setup and other mechanism can be chosen depending on the devices utilized. Also, we emphasize that HMD [36] is neither contribution nor the proposed solution is bound to HMD proposed in [36]. Deploying a lightweight malware detector will benefit by reducing the resource consumption and also aids when designing a malware confinement solution, as will be discussed later. It needs to be noted that HMD is solely adopted based on the experimental setup and other mechanism can be chosen depending on the devices utilized. Other malware detectors that consider different attributes of the node and/or network can also be considered as long as it provides the estimate of presence of malware.

2) Malware Propagation through Communication Links

In addition to the malware detection at node-level in an IoT network, the propagation of malware poses additional challenges in IoT networks. The connectivity between IoT devices both in terms of close proximity, and sheer numbers make them vulnerable to contamination. None/minimal security measures in the networks result in compromise of other device's/node's security[3] through propagation. If a perfect real-time malware detection is feasible, an effective strategy is to simply detect and quarantine the malware to avoid malware propagation into the network. However, constantly evolving malware is showing that no detection technique will be instantaneous and deliver a perfect yield, as many malware are purposely designed to avoid detection. For instance, the work in [38] has shown the possibility of spreading a light bulb worm which allows a reprogrammed bulb to re-flash nearby counterparts.

In addition, due to limited computing resources at the node-level, sophisticated and highly accurate malware detection solutions cannot be afforded [36], [39], [34], [35]. Consequently, deploying a semi-accurate malware detection method and a proactive approach such as immediately disconnecting the links in the network, while can confine malware, but also impact network performance and throughput, or even can result in communication loss, which is detrimental to the network performance. As a result, a more effective and methodical technique to confine malware propagation through communication links is needed, that balances the desire to contain the malware while preserving the network functionality and performance in the presence of semi-accurate malware detectors.

3) Scalability of Traditional Malware Confinement Methods

Malware confinement by formulating as a stochastic control problem and optimization, though effective (solution to previous challenge of malware confinement), requires malware propagation information to be predicted feed-forward in time to ensure that the trade-off between performance and malware infection is met. This incurs significant latencies, and thus it is limited to few tens of nodes due to involved complexity. The large execution time and memory consumption of simulation-based models motivate us to consider data-driven-based models in machine learning (ML) domain [40]. Motivated by the most recent progress of self-supervised learning in ML community, cutting-edge deep learning methods are able to learn the transformation mapping of complex data from one status to another [41]. For example, image translation methods transfer a painting from one style to another based on convolutional neural networks and generative adversarial nets [42]. Once the transformation mapping is learned, the prediction is instantaneous, just the same as that of conventional supervised learning method. For self-supervised learning, although deep generative models have proven their effectiveness on the generation of continuous data such as images, and videos, the prediction and generation for discrete data such as graphs are still open questions and are extremely challenging.

However, the problem of malware confinement requires constant prediction of the malware propagation status across the network, which is to predict a graph based on the current topology. The recently introduced generative deep learning models for graphs are typically unconditioned generative models, which typically only synthesize additional graphs directly following the distributions of the observation graphs and has no control over modes of the graphs being generated. However, in our problem, we need to predict the future graph status based on the current graph status and node status, thus a model that can generate a graph by conditioning on another is required, which cannot be achieved by the existing graph learning models.

B. OVERVIEW OF PROPOSED SOLUTION

In this work, we propose a two-pronged approach for effective malware detection and confinement in IoT networks.

First, we address the problem of malware detection on the IoT devices by deploying effective hardware-assisted malware detection proposed in a recent work [36], under the constraints of optimum latency, power and silicon footprint. This is adapted in this work based on the devices we chose in the experiments i.e., devices that host on-chip Hardware performance counters (HPCs) registers. However, other malware detection techniques can be adopted, as the stochastic controller is not dependent on the HMD, rather it requires an estimate of malware at node-level. From our experimental evaluations, we employ a lightweight rule-based JRip classifier at node-level to analyze the microarchitectural events and classify the malware and benign applications. The advantages of employing such HMD are its low processing overheads along with high detection performance [36]. Regardless of the best efforts on deploying highly accurate malware detectors, malware propagation is still feasible in IoT network, as no "perfect" malware detector exist.

Further, to confine the malware propagation in the network, the outputs of deployed node-level malware detector

[3]Compromised node refers to an IoT device that has been infected by malware.

are fed to the proposed malware epidemic controller. It needs to be noted that to solve the stochastic control problem for malware confinement, it requires an estimate of node-level malware existence from the malware detector, rather than accurate detection information. Furthermore, as accurate information might not be always feasible, using stochastic optimization utilizing the estimate is more beneficial. While there exists vast research on epidemic processes control in general, there are very few works that consider tractable real-time feedback control solutions working with the stochastic dynamics of malware spreading [43]. Works such as [44] performs the optimal control to minimize the spread of the malware, but only considers spreading characteristics rather than node state, which also plays a non-trivial role in malware spreading. Instead, the only known works that primarily study control of the stochastic dynamics are [45], [46], [47]. In [45], [46], it is shown that for the standard continuous-time (Susceptible-Infected-Susceptible) SIS epidemic process, there exists a finite threshold for varying the healing parameter of the process, above which the process can be controlled to the disease-free equilibrium exponentially quickly, and below which it cannot. In addition, these works mathematically characterize a sub-optimal controller for the processes and characterize its sub-optimality. However, the computations required to implement such controllers are known to be NP-hard and inapproximable, indicating that they are not applicable for real-time confinement. Instead, a tractable fixed rolling horizon controller is developed in [47] that considers performance constraint can be seen as a solution addressing similar challenge, however, a mechanism to incorporate node-level information is not available.

In this work, instead of directly solving the stochastic optimal control problem, which requires efficient methodologies for estimating and predicting the future malware propagation in the network graphs, a similar functionality is mimicked by our novel graph translation method, in which the future propagation graph topology and node status (malware probability) is predicted based on the current topology and node-level malware information. Different from graph generation problem, graph translation considers learning the propagation process from one graph state to another state. Existing methods [48] only deal with the topology translation, which is not suitable for our problem, where the node attributes (states) and propagation controlling parameters have influence on the propagation process. Specifically, we propose a multi-attributed deep graph translator (MA-DGT) to learn the propagation process from the initial graph to infected graphs, where the translation process is controlled by node states, and topology.

C. CONTRIBUTIONS

The contributions of this work can be outlined in a four-fold manner as follows:

- We develop a unified framework for combining the models of malware spreading processes on IoT networks explicitly with their direct adverse effects on network performance and formulate a stochastic optimal control problem for malware confinement while maintaining the network connectivity.
- We show how the output of a node-level malware detector can be deployed to generate imperfect estimates of infection state information as an input to proposed rolling horizon optimal controller for epidemic containment.
- We propose a heuristic end-to-end detection and defense strategy for IoT networks to solve the malware confinement problem. We demonstrate its superior performance against commonly used heuristic containment strategies.
- Lastly, to address the scalability concerns, a novel deep learning-based graph translation method is proposed based on graph convolutional neural networks and generative adversarial nets. Once being trained, the proposed method runs instantly and much faster than deriving the solution for stochastic optimal problem i.e. malware confinement.[4]

II. PROBLEM FORMULATION

Here, we introduce the IoT network architecture and the malware model, based on which we formulate the stochastic optimal control problem for malware confinement.

A. IOT NETWORK ARCHITECTURE

The network architecture and connectivity to various IoT nodes is shown in Figure 1. The network comprises of multiple heterogeneous nodes, each having an on-board low-end microprocessor to perform simple operations such as attenuation, routing, and noise removal. The heterogeneous IoT nodes are placed randomly in a $L \times M$ space, (L is the length and M is the width), with each node connected to its neighboring IoT node(s) within range R via Bluetooth. The nodes are of type broadcasting stations, routers, or sensory nodes. Each of these nodes are equipped with a lightweight ML classifier to differentiate the malware and benign applications during runtime. This requirement is for the considered experimental setup but is not a requirement nor constraint to deploy our proposed solution. The process of malware detection on IoT nodes is illustrated in the left zoom-out box of Figure 1. A runtime malware detection is adopted from a recent work [36], where it is considered the devices that has embedded processor and utilizes ML for malware detection, similar to [34], [35], [36], [49]. However, one can accommodate other malware detection techniques, as long as it can provide an estimate of node-level infection. In this work, we consider all the nodes to be of same priority (all the devices are equally important) and the nodes can host a microprocessor to execute the adopted malware detection technique. The rationale for such assumptions is that future IoT networks consist of smart devices. As aforementioned, other lightweight techniques can be considered for malware detection depending on the utilized device's specifications.

[4]The source code is available at https://github.com/xguo7/MA-GT-GAN

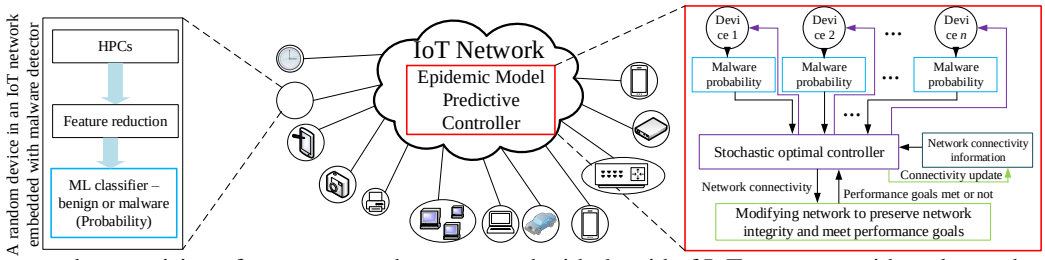

Figure 1: IoT network comprising of numerous nodes connected with the aid of IoT gateways with malware detector deployed on each node and a centralized malware confiner

At time-instant 0, a set of randomly chosen nodes (2 nodes in this work) are set to be infected. However, as the time progresses, any of the nodes in the network and at any point of time can be infected. A centralized epidemic malware predictive controller is embedded into the network to modify the network topology during runtime in order to confine the malware propagation in the network. The malware information from each node is directly communicated to the controller. This controller is fed with the node-level malware infection probabilities to perform a stochastic control-based network optimization and preserve the network connectivity and confine the malware epidemics. The process of malware confinement is illustrated in the right zoom-out box of Figure 1. As the deployed network size is limited, a centralized controller is employed. However, for a large network, a hierarchy of controllers can be deployed, or network can be split using techniques such as graph slicing [50]. For real-time confinement in large scale networks, this work proposes MA-DGT.

B. MODELING MALWARE

We present the used notations, malware propagation model, and the effects of malware propagation on the network here.

Notations. We use $\mathbb{R}_{\geq 0}$ to denote the set of non-negative real numbers, and $\mathbb{Z}_{\geq 0}$ for the set of non-negative integers. The expectation of a random variable X is denoted by $\mathbb{E}[X]$. Note that when the measure of expectation is clear from the context, we omit it. However, when necessary, we explicitly include it as a subscript of the operator, i.e. $\mathbb{E}_{\mu}[X]$ indicates the expectation of X w.r.t. measure μ. Similarly, when clear from the context, we omit the initial condition $X(0)$ of a stochastic process. When necessary, we explicitly include it as a part of the expectation's conditioning, i.e. $\mathbb{E}[X(t)|X(0)]$, or interchangeably by a bracketed superscript i.e. $\mathbb{E}^{[X(0)]}[X_t]$.

Malware Spreading Model. The standard model used in the study of computer malware epidemics is the SIS model [51], [52]. In SIS model, the node is assumed to be either in infected (I) or susceptible (S) states. The primary assumptions in this work for the modeling are: the embedded nodes quarantine the malware once detected and the infection happens immediately. The considered model also reflects the real-world malware spreading such as worms and viruses, where the malware infects the node (node transitions to infected (I) state) and gets quarantined by the anti-virus techniques (i.e. node becomes malware free (S state)) [53], [54], [51]. The deployed malware spread model is also analogous to the internet worms and viruses that infects a node and start propagating through the network through downloads or self-replication. Some of the real-world malware that follow this kind of model are 'badBios' [55], 'Yankee Doodle' [56], and 'Magneto' [57]. Thus, based on these real-world examples and the considered node architecture with malware detector, ultimately, we use the stochastic SIS, which is a well-established model for epidemics on large-scale networks [58], [59], [60], [43], [51]. However, the proposed malware confinement only requires the estimate of malware propagation in the network and is applicable to other kinds of malware spreading models. In this work, the network is represented as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ with $|\mathcal{V}| = n$ nodes, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ as the directed edges, and $W \in \mathbb{R}^{n \times n}$ is the weighted adjacency matrix. The edge $(i, j) \in \mathcal{E}$ means that node j is sending data to node i at a rate proportional to w_{ij}. Note that $w_{ij} > 0$ if and only if $(i, j) \in \mathcal{E}$, and $w_{ij} = 0$ otherwise. We denote the set of neighbors of i as $\mathcal{N}_i = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}$. At any given time, the set of nodes are split into two compartments: Susceptible (S) and Infected (I), that represents the infection state of each node. Then, the state of node i at time t is given by the binary random variable $X_i(t) \in \{0, 1\}$, where $X_i(t) = 1$ indicates that the node i is infected with malware at time t, and similarly, $X_i(t) = 0$ indicates that the node i is currently free of malware, but susceptible. The infection state of the entire network is denoted by a vector $X(t) \in \{0, 1\}^n$.

The intuition of malware spreading model is as follows. Any node i that is infected with the malware is capable of passing it to a neighbor $j \in \mathcal{N}_i$ (within radius R) randomly with a Poisson rate $\beta > 0$ proportional to the amount of traffic flow w_{ij}, β termed as infection rate. At the same time each infected node is also able to naturally recover with a Poisson rate $\delta > 0$, δ termed as recovery rate. Thus, the SIS spreading process can be modeled using the Markov process as

$$\begin{aligned} X_i : 0 \to 1 \quad & \text{with rate } \beta \textstyle\sum_{j \in \mathcal{N}_i} w_{ij} X_j, \\ X_i : 1 \to 0 \quad & \text{with rate } \delta. \end{aligned} \tag{1}$$

Effect of Malware on the Network

In addition to the malware spreading model, we also present a model to study how the malware negatively affects the performance of a given network. Depending on the application, we

define a performance function $P : \mathbb{R}^{n\times n} \times \{0,1\}^n \rightarrow [0,1]$, where $P(W,X) \in [0,1]$ is the performance of the network, given network W and state of all the nodes X. If the original network $\mathcal{G}$ is used and all the nodes are healthy, i.e. $X(t) = \mathbf{0}_n$, then $P(W,X(t)) = 1$ meaning the network is running at 100% performance. As nodes are infected and links are removed from W, the performance degrades depending on the application of the network.

More specifically, we will consider a malware threat parameter $\sigma \in [0,1]$ that models the strength of malware for a chosen performance metric P. This malware threat parameter is defined based on the impact that it has on the performance of the network with $\sigma = 0$ being the weakest and $\sigma = 1$ being the strongest, such as DoS malware. The $\sigma = 0$ indicates that the malware has absolutely no effect on the actual performance of the network (benign malware such as data stealing). In this work, the malware threat parameter is determined from the database manually but can be determined as inverse to degradation of the network performance. A specific example of a performance metric P is given later in Section III-B.

C. FIXED-HORIZON STOCHASTIC OPTIMAL CONTROL PROBLEM

Based on the introduced models and notations, we formulate the problem of malware propagation control in the network. Let $W'(t)$ denote the modified graph where the traffic between some nodes may have been reduced due to different control activities such as removal of links. In other words, as a consequence of the control mechanism, traffic (transmission of malicious applications) is regulated between each pair of active links with $w'_{ij} \leq w_{ij}$ to reduce the chance of node j spreading malware to node i [61], [62]. Thus, the problem can be defined as maximizing the objective function

$$J = \frac{1}{T}\int_0^T P(W'(t), X(t)) \tag{2}$$

over some time horizon $T > 0$. Here $P(W'(t), X(t))$ denotes the performance of the network at time-instant t. Note that this objective function explicitly captures the trade-offs between shutting down links to contain the malware at the cost of reducing instantaneous network performance and keeping links active to maintain the network performance at the risk of letting the malware spread.

This problem poses two challenges to address. First, in general, we may not have access to the true infection state $X(t)$, i.e. malware is often meant to be undetected. This indicates that a mechanism to detect the malware on nodes is required to combat the malware propagation in the network. Second, the stochastic network dynamics already make this a non-trivial problem even if access to the true infection state $X(t)$ is available. In addition, there are only very few works that have studied the optimal feedback control problems for stochastic networked epidemics [63], [64], [65], [43], [66], [67].

Optimizing equation (2) is non-trivial even if the infection state $X(t)$ of the network is known at all times. Thus, to solve the problem of malware containment in the network, we partition it into three subproblems: how to best detect the malware on the nodes, confine the malware based on the outputs of inaccurate malware detector and perform malware confinement in large-scale networks.

SUBPROBLEM 1: MALWARE DETECTION AT IOT NODE-LEVEL

As aforementioned, maximizing the objective function J in equation (2) with access to X is a complex problem to solve. However, with a better estimate of true infection state of nodes, the performance maximization can be enhanced by deploying a better malware confinement technique in the IoT network. As such, a fast, and reliable malware detector is required to limit the malware spread in the IoT network. From the IoT device perspective, the malware detector needs to be resource efficient, and low cost to ensure that it fits on to the existing resource on the device.

In order to meet the above-mentioned challenges of malware detection at node-level, we adopt the hardware-assisted malware detection proposed in recent work [36]. In this work, the microarchitectural events are collected through the hardware performance counters and fed to a ML classifier (JRip is chosen in this work based on its superior performance and lower overheads (1 clock-cycle latency and 80× lower area) compared to ML classifiers like neural network) to differentiate benign and malicious applications (shown in left zoom-out of Figure 1). Additional details and evaluations of the adopted malware detection work is presented in Appendix -B.

SUBPROBLEM 2: OPTIMAL NETWORK CONNECTIVITY MAINTENANCE

Regardless of the malware detection method used at the node-level, there exists misclassification (false negatives), especially for the emerging unseen malware. Thus, even with highly accurate malware detection, a well-connected IoT network serves as a vulnerability that can allow the malware to spread very quickly across the network before it can even be detected. Lack of information on node infection estimate leads to a random optimization, which is not efficient. This indicates that in addition to the malware detection, a more proactive and pragmatic defense mechanism to maintain the network connectivity is needed rather than a simple 'detect and quarantine' strategy, as by the time a device identified to contain malware has been quarantined, malware could have already spread to other devices in the network.

As mentioned, to the best of our knowledge there are no prior works that are able to completely solve the problem presented in equation (2). However, if we ignore the performance aspect of the problem, there is a set of research dedicated to containing epidemics. In particular, we consider the solution proposed in [47], [68] where a rolling horizon model predictive controller is designed for containing the stochastic epidemic process as quickly as possible for a given budget constraint. However, the novelty here is to integrate the malware propagation information with node-level

information as well as modeling the malware confinement as rate-constrained problem and deploying rolling horizon model predictive controller. More specifically, the algorithm proposed in [68] is capable of solving the following rate-constrained allocation problem.

Rate-constrained Allocation: Let $g_{ij} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be decreasing function that denotes the cost of setting the weight of an edge. More specifically, $g_{ij}(w'_{ij})$ is the cost required to change the weight of edge $(i,j) \in \mathcal{E}$ from w_{ij} to w'_{ij}. Note that $g_{ij}(w_{ij}) = 0$ for all $(i,j) \in \mathcal{E}$ meaning there is no cost associated with leaving a weight unchanged. Then, given a desired decay rate $r \in (0,1)$, the rate-constrained allocation problem is to determine the optimal network configuration to eradicate the disease with a decay rate of at least r. This can be mathematically formulated as

$$\begin{aligned} \underset{\{w'_{ij}\}_{(i,j)\in\mathcal{E}}}{\text{min.}} \quad & \textstyle\sum_{(i,j)\in\mathcal{E}} g_{ij}(w'_{ij}) \\ \text{s.t.} \quad & \mathbb{E}\left[\textstyle\sum_{i=1}^{n} X_i(t+\Delta t) | \{\Pr(X_i(t)=1)\}_{i\in\mathcal{V}}\right] \\ & \qquad \leq r\left(\textstyle\sum_{i=1}^{n} X_i(t)\right), \\ & w'_{ij} \leq w_{ij} \quad (i,j) \in \mathcal{E} \end{aligned} \tag{3}$$

for all $i \in \{1,\ldots,N\}$. Given the current probability of infection for each node $\hat{X}_i(t) = \Pr(X_i(t))$ and a desired decay rate $r > 0$, the algorithm proposed in [68] finds the minimum-cost allocation to realize this geometric decay rate in expectation i.e. after one time-step, the total number of infected nodes in the network should have decreased in expectation, as given in Section III.

We note that although this problem is not equivalent to optimizing equation (2), it is very closely related through choosing the rate constraint r. More specifically, $r \to 0$ means that the algorithm wants to eradicate the malware as quickly as possible, regardless of how many links it needs to remove. In the case of optimizing equation (2), this is not a practical solution as this would mean completely disconnecting our network to guarantee the infection stops spreading; unfortunately, this would mean that the primary function of the network is completely decimated. On the other hand, $r \to 1$ means that the algorithm is not concerned as much with ensuring that the malware is eradicated. Consequently, this decay rate r, which is an input to the rate-constrained allocation problem, can be used to balance how much we want to cease the spreading of the malware with how important it is to maintain connectivity of network.

SUBPROBLEM 3: SCALABILITY OF MALWARE EPIDEMIC CONTROL

Stochastic control based solutions are slow in nature and often involve computational complexities resulting in large delays. To apply for large-scale IoT networks, we formulate this scalability problem similar to translation of one graph to another graph, where the initial graph is the state of network before solving stochastic control optimization and the output (translated) graph is the graph after solving the stochastic control problem i.e. with new interconnects that limit malware epidemics and maintain network throughput. Thus, the problem is formalized as a multi-attributed graph translation conditioning on two kinds of attributes: the node states and parameters. We define an input graph $\mathcal{G}_X = (\mathcal{V}, \mathcal{E}, W)$ as an undirected weighted graph. The infection state of the entire network is denoted by a vector $X(t) \in \{0,1\}^n$. The is also an undirected weighted target graph $\mathcal{G}_Y = (\mathcal{V}', \mathcal{E}', W')$, which is a result graph after the SIS spreading process. An external parameter vector is given as $P = \{\beta, \delta, r\}$ consisted of infection rate, recovery rate and decay rate. Typically, we focus on learning the translation from one circuit connections $\mathcal{G}_X$ to another $\mathcal{G}_Y$. Translation focuses on learning a translator from an input graph $\mathcal{G}_X$, a random noise U as well as the two kinds of attributes X and P, to a target graph $\mathcal{G}_Y$, the translation mapping is denoted as $T : U, X, P, \mathcal{G}_X \to \mathcal{G}_Y$.

III. MALWARE EPIDEMIC CONTROL

To perform the malware epidemic control, we first obtain the node-level malware information using the adopted HMD [36]. Further to confine the malware, we employ an epidemic model predictive controller to which the output from the node-level malware detector is fed. We first discuss the epidemic model predictive control optimization followed by its deployment to solve the problem of malware confinement in the network.

A. EPIDEMIC MODEL PREDICTIVE CONTROLLER

In order to contain the malware, we apply the rolling horizon controller developed in [47]. Theorem 3.1 shows how the rate-constrained allocation problem in equation (3) can be reformulated as the equivalent convex program below.

Theorem 3.1: [Convex Bayesian SIS Control [47]] Consider the convex optimization program

$$\begin{aligned} \underset{\vec{\delta^c},\vec{\gamma}}{\text{minimize}} \quad & \sum_{i\in V} f_i(\delta_i^c) + \sum_{(i,j)\in E} \tilde{g}_{ij}(\gamma_{ij}) \\ \text{s.t.} \quad & \sum_{i\in\mathcal{O}} \delta_i^c X_i + \psi_i^{\{w\}} X_i^c + \\ & \sum_{i\in\mathcal{O}^c} \delta_i^c \hat{x}_i(t|t) + \psi_i^{\{w\}} \hat{x}_i^c(t|t) \\ & \leq r \sum_{i\in V} \hat{x}_i(t|t), \end{aligned} \tag{4}$$

where we additionally restrict the variables δ_i^c and γ_{ij} to the closed unit interval, and have defined the convex functions

$$\psi_i^{\{w\}} = \begin{cases} 1 - \hat{x}_{j'}(t|t)\gamma_{j'i}^{\frac{1}{w}} \Pi_{j\in\{\mathcal{N}_i\cap\mathcal{X}_i\}} \gamma_{ji}^{\frac{1}{w}} \\ \quad -\hat{x}_{j'}^c(t|t)\Pi_{j\in\{\mathcal{N}_i\cap\mathcal{X}_i\}}\gamma_{ji}^{\frac{1}{w}}, & i \in \mathcal{O} \\ 1 - \Pi_{j\in\{\mathcal{N}_i\cap\mathcal{X}_i\}} \gamma_{ji}^{\frac{1}{w}}, & i \in \mathcal{O}^c \end{cases} \tag{5}$$

where $w > d_{\max}$, the sets $\mathcal{X}_i = \{i \in V \cap \mathcal{O} \,|\, X_i = 1\}$, and the shorthand notation $X_i^c = (1 - X_i)$, $\hat{x}_i^c(t|t) = (1 - \hat{x}_i(t|t))$, and $\delta_i^c = (1 - \delta_i)$ for purposes of compacting notation. Suppose the functions f_i and $\tilde{g}_{ij} = g_{ij}(1 - \gamma_{ij}^{\frac{1}{w}})$ are convex in the variables δ_i^c, and γ_{ij} respectively, then equation (3) and equation (4) are equivalent optimization problems, where the optimal edge weights of equation (3) can be computed as $\gamma'_{ij} = 1 - (\gamma^\star_{ij})^{\frac{1}{w}}$, where $\gamma^\star_{ij}$ is the solution to equation (4).

In Theorem 3.1, the set $\mathcal{O} \subset \mathcal{V}$ denotes the set of nodes for which the binary infection data is known, and $\mathcal{O}^c = \mathcal{V} \setminus \mathcal{O}$ is the set of nodes for which no data is available. Unfortunately, in our practical setup, since the malware detection algorithm will not always be perfect, we actually do not have access to the exact infection data $X_i(t)$ for any nodes $i \in \mathcal{V}$. Instead, the *HMD* provides independent estimates $\hat{X}_i(t) = \Pr(X_i(t) = 1)$. This requires determining a way manipulate Theorem 3.1 to match our requirements and use it to solve the rate-constrained allocation problem equation (3), as discussed below.

Since the true infection state $X_i(t)$ is a binary random variable for each node i, we instead maintain an estimate $\hat{X}_i(t) \in [0, 1]$ at all times. More specifically, we let $\hat{X}_i(t|t)$ be the estimate of infection state $X_i(t)$ about time t available at time t.

Taking expectations of the random binary infection variables, the dynamics according to equation (1) are given by

$$\frac{d\mathbb{E}[X_i](t)}{dt} = -\delta X_i(t) + \beta \sum_{j \in \mathcal{N}_i} w'_{ij} X_j(t) \tag{6}$$

where the modified network W' is used rather than the original network W. In addition, we need to combine this estimate with the updated information provided by the *HMD*. Using this, we can propagate the estimates forward in time.

Given the current infection estimate $\hat{X}_i(t_\ell|t_\ell)$, we propagate this forward according to equation (6), a short time later by

$$\hat{X}_i(t_{\ell+1}|t_\ell) = \hat{X}_i(t_\ell|t_\ell) + \Delta t(-\delta \hat{X}_i(t_\ell|t_\ell) + \tag{7}$$

$$\beta \sum_{j \in \mathcal{N}_i} w'_{ij} \hat{X}_j(t_\ell|t_\ell)). \tag{8}$$

However, these estimates with outputs of the deployed malware detector (*HMD*) has to be combined in order to obtain independent estimates $\hat{X}_i(t|t) = \Pr(X_i(t) = 1)$ each time the malware detection algorithm is executed.

Let $y_i(t_\ell) \in [0, 1]$ be the output of the proposed malware detector (*HMD*) on node i at time t_ℓ. Assuming this output is an independent probability that the node is infected with malware, we update the probability of infection of each node $\hat{X}_i(t_\ell|t_\ell)$ conditioned on the new information available at each sampling time $t \in \{t_\ell\}_{\ell \in \mathbb{Z}_{\geq 0}}$ as

$$\hat{X}_i(t_\ell|t_\ell) = 1 - ((1 - \hat{X}_i(t_\ell|t_{\ell-1}))(1 - y_i(t_\ell))) \tag{9}$$

Thus, given both a way for propagating the estimate $\hat{X}(t'|t)$ at time t and a way to incorporate new measurements $y_i(t_\ell)$, we have a method for estimating $\hat{X}(t)$ of the infection state of all nodes.

More specifically, revisiting the Theorem 3.1 and rather than seeing $\mathcal{O} = \emptyset$ as the set of nodes, we now have the perfect infection information, and we instead use the independent probabilistic estimates of each node's infection state at each time-step $\{t_\ell\}_{\ell \in \mathbb{Z}}$. This is done by combining the propagation equations with the outputs of *HMD*, we can generate artificial observations $\hat{X}_i(t_\ell|t_\ell) = \Pr(X_i(t) = 1)$ for all nodes $i \in \mathcal{V}$, as given in equation (9).

Based on this, by considering the special case of Theorem 3.1 where we now have $\mathcal{O} = \mathcal{V}$ and $\mathcal{O}^c = \emptyset$, and for a fixed recovery rate $\delta > 0$, we simplify it to an equivalent formulation as shown in Theorem 3.2 below.

Theorem 3.2: [Equivalent problem as Theorem 3.1] Consider the convex optimization program

$$\begin{aligned} \underset{\{w'_{ij}\}_{(i,j)\in\mathcal{E}}}{\text{minimize}} \quad & \sum_{(i,j)\in\mathcal{E}} g_{ij}(w'_{ij}) \\ \text{s.t.} \quad & \sum_{i\in\mathcal{O}} (1-\delta)X_i + \psi_i^{\{w\}}(1 - X_i) \leq r \sum_{i\in\mathcal{V}} \hat{X}_i(t|t), \end{aligned} \tag{10}$$

where, we have defined the convex function

$$\psi_i^{\{w\}} = 1 - \hat{X}_{j'}(t|t)\gamma_{j'i}^{\frac{1}{w}} \Pi_{j\in\{\mathcal{N}_i \cap \mathcal{X}_i\}} \gamma_{ji}^{\frac{1}{w}} - \hat{x}_{j'}^{c}(t|t) \Pi_{j\in\{\mathcal{N}_i \cap \mathcal{X}_i\}} \gamma_{ji}^{\frac{1}{w}} \tag{11}$$

where $w > d_{\max}$ and $\mathcal{X}_i = \{i \in V \cap \mathcal{O} \,|\, X_i = 1\}$. Suppose the function g_{ij} is convex in the variable w'_{ij}, then equation (3) and equation (10) are equivalent optimization problems, where the optimal edge weights of equation (3) can be computed as $w'_{ij} = 1 - (w^\star_{ij})^{\frac{1}{w}}$, where $w^\star_{ij}$ is the solution to equation (10).

Note that the convexity of each $\psi_i^{\{w\}}$ can be verified by applying an established result from signomial optimization works such as [69]. However, it is worth noting that product terms are in general non-convex, and so CVX [70] may not solve the problem. Solutions can instead be obtained by coding standard convex optimization algorithms (see [71]). Thus, it is possible to effectively solve the rate-constrained allocation problem with a chosen decay rate.

B. OVERALL SOLUTION

Finally, to put the solutions of the 2 subproblems together in a way to solve the optimization problem in equation (2), we need to consider a specific form of the performance function P. There exist numerous metrics to evaluate the performance of a network such as throughput, latency, and bandwidth. In this work, to evaluate the network performance in terms of network throughput, as given by [72], [73]

$$P(W'(t), X(t)) = \tau = \int_0^T \frac{\log(1+w)}{h} P_{suc}(1 - X(t)) \tag{12}$$

where P_{suc} is the probability of successful transmission, modeled as the inverse of the node malware infection probability; h is the number of hops; and w is the weight assigned to the communicating nodes (signal-to-noise ratio (SNR)). The SNR is given by

$$\frac{g_{ij} d_{ij}^{-\alpha}}{\sum_{k\in L(t)\setminus j} g_{ij} d_{ik}^{-\alpha}} \tag{13}$$

where d_{ij} represents the distance between nodes i, j with a channel gain g_{ij} in the network $L(t)$ at a given time t; The path-loss exponent is given by α. As such, the traffic between nodes is determined based on the throughput, connectivity, and the malware infection of the nodes. The decay rate r is provided to the epidemic controller based on the malware threat parameter σ obtained from Virustotal.com [74] as a

way to limit how much the network can be disconnected. As $r \rightarrow 0$, the solution to Theorem 3.2 provides much more aggressive containment strategies by disconnecting the entire network. On the other hand, as $r \rightarrow 1$, the solution to Theorem 3.2 allows the malware to spread and not disconnect many links. Based on this, the proposed optimization solution determines the best possible network connectivity in order to maintain as many original connections as possible while satisfying the desired decay rate. The proposed work can also be extended to other network of devices under different resource constraints, though we showcase the benefit for IoT network here.

IV. GRAPH TRANSLATION BASED MALWARE CONFINEMENT

In order to perform the graph translation i.e., predict the graph (network topology) instantaneously, which is similar to the graph predicted by the stochastic optimization solution, we propose to use a graph translation model to handle the malware epidemic confinement problem automatically. However, the existing translation model [48] generates output graphs only conditioning on the input adjacent matrix W without additional attributes, e.g.the node states as well as the malware parameters (infection, recovery and decay rates), which are critical to the malware epidemic confinement process. Thus, graph translation-based malware confinement has two challenges that cannot be solved by the existing translation model: 1) Node state representation as the input conditions, and 2) Fusion of different contextual information in different dimensions.

To solve the above two challenges, we propose a fusion representation of graphs containing both edge and node information, and further propose a multi-attributed discriminator and generator architecture (MA-DGT) by integrating the impacts of input graphs and external attributes on the translation process. An overview of the proposed MA-DGT architecture is shown in Figure 2. The functionality of individual components is described below.

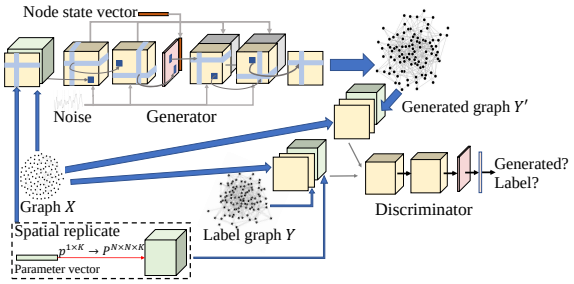

Figure 2: Architecture of proposed MA-DGT for network translation

A. MULTI-ATTRIBUTED GENERATOR

1) Fusion representation

The edges of undirected weighted graph are initially presented as a symmetry matrix. To avoid redundant computations, we restrict the direction of connection from high indexing nodes to low indexing nodes, thus transforming the symmetry matrix to an upper triangular matrix in directed graph. Specifically, denote the $W^{l,m} \in \mathbb{R}^{n\times n}$ as the weighted adjacency matrix for l-th layer in m-th feature map and $W^{l,m}_{i,j}$ is for the edge $e_{i,j}$. Let A denote the weighted adjacency matrix of the input graph. The node state information is stored in the diagonal values of A. To incorporate the parameters vector P, we do a spatial replication to reshape $P \in \mathbb{R}^{1\times 3}$ to $P \in \mathbb{R}^{n\times n\times K}$. Then the attributed input graph tensor $W \in \mathbb{R}^{n\times N\times 4}$ is generated by concatenating adjacent matrix A and parameter matrix P. $\Phi_{l,m} \in \mathbb{R}^{1\times n}$ and $\Psi_{l,m} \in \mathbb{R}^{n\times 1}$ are the incoming and outgoing kernels of l-th layer of k feature map for a node, respectively.

2) Multi-attributed Graph convolution

We define a graph convolution over its in-edge(s) as the weighted sum over all the weights of its incoming edges: $f^{(\text{in})}_{l,m,n,j} = \Phi_{l,m,n} \cdot W^{l,m}_{\cdot,j}$. Similarly, we define the graph convolution over the out-edge(s) as $f^{(\text{out})}_{l,m,n,i} = W^{l,m}_{i,\cdot} \cdot \Psi_{l,m,n}$. And thus, the directed edge-to-edge convolution is defined as follows:

$$Z^{l+1,n}_{i,j} = \sigma(\sum\nolimits_{m=1}^{M_l}(f^{(\text{in})}_{l,m,n,i} + f^{(\text{out})}_{l,m,n,j})) \qquad (14)$$

where $W^{l+1,m}_{i,j}$ refers to the m-th value in position i,j of edge level feature map in the $(l+1)$-th layer. M_l refers to the number of feature maps in the l-th layer. The two components of the formula refer to direction filters as talked above. $\sigma(\cdot)$ refers to activation function that can be set as linear, or ReLu when the edge weights are assumed non-negative. The edge-to-node convolution embeds each edge feature map into a vector which encodes all the incoming and outgoing edges of a node into a value from various combinations:

$$W^{4,n}_{i} = \sigma(\sum\nolimits_{m=1}^{M_3}(f^{(\text{in})}_{3,m,n,i} + f^{(\text{out})}_{3,m,n,i})) \qquad (15)$$

where $W^{4,n}_{i} \in \mathbb{R}^{n\times 1}$ denotes the i-th node (i.e. the 4th layer in graph translator) in Figure 2 under n-th feature map.

After the edge-to-node convolution layer, each node is embedded into a vector $w_i \in \mathbb{R}^{M_4\times 1}$ consisting of M_4 features and the whole graph is embedded as a matrix $W \in \mathbb{R}^{M_4\times n}$. In this node level representation, we input the node states vector $X \in \mathbb{R}^{1\times n}$ to concatenate matrix W and vector X, generating the matrix $W \in \mathbb{R}^{(M_4+1)\times n}$ as input of the deconvolution part.

3) Multi-attributed Graph Deconvolutions

After graph encoder, it requires to deconvolute the node representation back to target graph. This calls for a reverse process of "edge-to-node convolution" as shown in Figure 2. To achieve this, the node representation $W^{l,k} \in \mathbb{R}^{1\times n}$ in l-th layer in k-th feature map will be multiplied by the transpose of incoming and outgoing kernels to obtain weighted adjacency matrix in the $(l+1)$-th layer:

$$Z^{l+1,m}_{i,j} = \sum\nolimits_{k=1}^{M} \sigma([\Phi^{T}_{l,k} \cdot Z^{l,k}_{j}]_i + [Z^{l,k}_{i} \cdot \Psi^{T}_{l,k}]_j) \qquad (16)$$

where $[\cdot]_i$ means the i-th element of a vector. The decoded edges from node representation still encompasses highly-ordered connectivity knowledge, which will be released and

translated back to the neighborhood of incident incoming and outgoing edges by "edge-to-edge deconvolution":

$$W_{i,j}^{l+1,m} = \sigma([\sum_{k=1}^{n} \Phi_{l,m}^{T} \cdot W_{k,j}^{l,m}]_i + [\sum_{k=1}^{n} Z_{i,k}^{l,m} \cdot \Psi_{l,m}^{T}]_j) \quad (17)$$

To ensure the generated graph is undirected, we still constrict the weighted adjacent matrix is upper triangular matrix by multiplying the output matrix W from last layer with a unit upper triangular matrix $O \in \mathbb{R}^{n \times n}$.

B. MULTI-ATTRIBUTED GRAPH DISCRIMINATOR

Discriminator is used to identify whether the generated graphs follow the same distribution of target graphs, which encodes a second-order change between input and target graphs. Different from DGT, we propose a multi-attributed conditioned discriminator, as shown in Figure 2. The input of the discriminator is multi-attributed graph tensor$Z \in \mathbb{R}^{n \times n \times 4}$, same as the input of generator. Specifically, the input graph tensor and target graph are together inputted into discriminator after being concatenated into a $n \times n \times 5$ tensor which can be considered as a 5-channel weighted adjacency matrix of a multi-graph. Next, each of the channels are mapped to its corresponding feature maps and then to the separated edge-to-edge layers. The edge-to-node layer is again applied to obtain node representations, which is then mapped to graph embedding by a fully connected layer. Finally, a softmax layer is implemented to distinguish the generated graphs and target graphs. All the weights of filters in the network are initialized and optimized through ADAM optimization algorithm.

C. TIME AND MEMORY COMPLEXITY

Due to the similar convolution operations, the multi-attributed graph generator and discriminator share same time complexity. For generalization, we assume all the layers (except input and output) have the same number of feature maps as M_0. S is the length of the fully connected layer. K is the number of parameters. Then, the worst-case (i.e. when the weighted adjacency matrix is dense) total complexity is $O((0.5n^2M_0(K+1) + 2n^2M_0^2) + n^2M_0^2 + n^2M_0S)$, where the first, second, and third terms are for the "edge-to-edge convolutions", "edge-to-node convolutions", and fully connected layers in conditional graph discriminator with the indirect input matrix, respectively. Similarly, the total memory complexity is $O((4nM_0 + 4.5n^2M_0) + 3nM_0 + (nM_0S + 2S))$ for all the "edge-to-edge convolutions", "edge-to-node convolutions", and fully connected layers in conditional graph discriminator. Compared to the analytical solutions that needs large search time to determine the (sub-)optimal solution, the MA-DGT requires training with the graph topology and predicted information. However, predicting at runtime is the main objective of this work and hence, we compare the runtime analysis in this work and performing online learning for MA-DGT is out of scope.

V. EXPERIMENTAL RESULTS

We present the evaluation of the malware epidemic control with the aid of proposed stochastic control technique and graph translation.

A. EXPERIMENTAL SETUP

The malware propagation performed in the experiments is similar to that in [75], [76], [77], [78].

Network and Hardware Setup: A small-scale setup of 20 IoT nodes encompassing temperature sensors connected with Intel ATLASEDGE board, Beagle Boards (BeagleBone Blue) having ARM processor, communicating via Bluetooth protocol are deployed as described in Section II-A in an area of 5×5 m^2 (in our lab). The deployed boards host RISC architecture with embedded Linux OS running on them. The devices are statically placed during the experiments. The epidemic model predictive controller is executed on a controller built on Intel Haswell core i5 processor.

Software Framework: On each device, i.e. at the node-level, in order to extract the HPC information, *Perf* tool is employed. It exploits *perf_event_open* function call in the background to measure multiple events simultaneously. We perform feature reduction using correlation extraction and observe a limited number (four) of critical HPCs for malware detection due to limited resource availability. We employ JRip ML classifier for malware detection, similar to that in [34], [36], which achieves an accuracy of 91.08% with 1 clock-cycle latency. This is also explained in Appendix -B. The Python framework is utilized to develop graph translation.

Applications: We executed 3000 benign and malware applications for HPC data collection. Benign applications include MiBench [79] and SPEC2006 [80], Linux system programs, browsers, text editors, and word processor. For malware applications, Linux malware is collected from virustotal.com [74] and classified on virusshare.com [81]. Malware applications include Linux ELFs, python scripts, perl scripts, and bash scripts, which are created to perform malicious activities consisting of four classes of malware including 452 Backdoor, 350 Rootkit, 650 Virus, and 1169 Trojans. The functionality of the deployed malware applications is: Backdoor applications try to provide remote access to the remote user (attacker) and facilitates information leakage; Rootkit applications provide the attackers with privilege even to modify the registers and authorized programs; Trojans perform phishing of information stored in the system, and passwords; self-duplicating Viruses and self-replicating worms that can launch DoS attacks are deployed on the IoT nodes.

B. EVALUATION OF MALWARE EPIDEMIC CONTROL

We evaluate the performance of malware epidemic controller discussed in Section III-B on the network with 20 nodes deployed in our lab within a 5×5m^2 area. 1000 experiments are carried out with each experiment lasting for 40 seconds. Devices in the network are affected by malware randomly at any point of time, with multiple attacks on each of the devices to replicate real-world scenario. At the initial time-instant ($t = 0$), two nodes are deployed with self-propagating malware. We evaluate the network performance in terms of overall throughput, averaged over 1000 experiments.

1) Throughput with Time

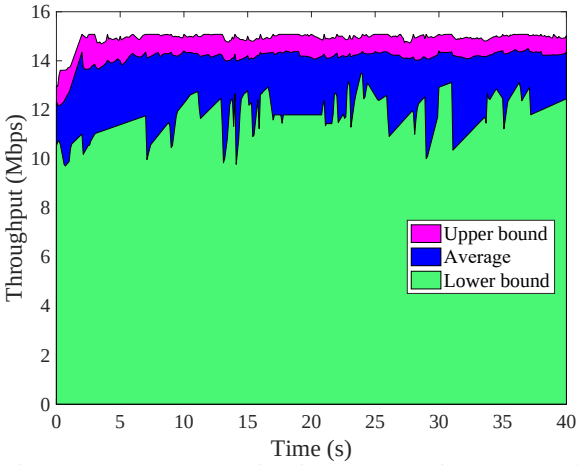

Figure 3: Throughput with time (along with upper and lower bounds)

The variation of throughput with time is presented in Figure 3. The lower bound and the upper bounds represent the best-case and worst-case scenarios achieved by employing the proposed malware epidemic model predictive controller is shown here. One can observe that in most of the cases (i.e. average) the proposed method achieves throughput close to the upper bound. By employing, the proposed epidemic model predictive controller, a throughput $\sim 95\%$ of the upper bound is achieved in most of the experiments. This confirms the effectiveness of the proposed malware confinement and ensures the throughput is not hampered, irrespective of the attack.

2) Throughput under Different Malware Properties

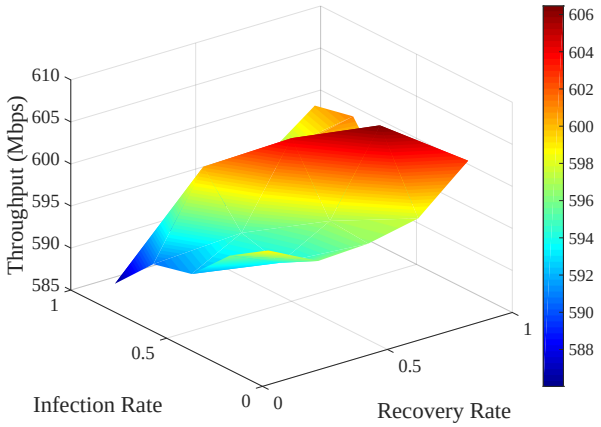

Figure 4: Network performance (throughput) with the infection rate and recovery rate

Figure 4 presents the overall network performance (throughput) under different malware infection and recovery rates. With the increase in recovery rate, the network throughput increases and is highest at lower infection rate and high recovery rate. However, the infection rate increase hampers the overall throughput, and it has higher impact on the throughput compared to recovery rate. It can be seen that the right-most corner in Figure 4 i.e. high infection and high recovery rate has slightly lower throughput compared to the high recovery and low infection rate. This shows that runtime malware detection and quarantining (recovering) is not the panacea to have secure IoT networks, as infection (propagation) has higher impact than recovery (quarantining). Additional analysis with malware parameters is presented in Appendix A.

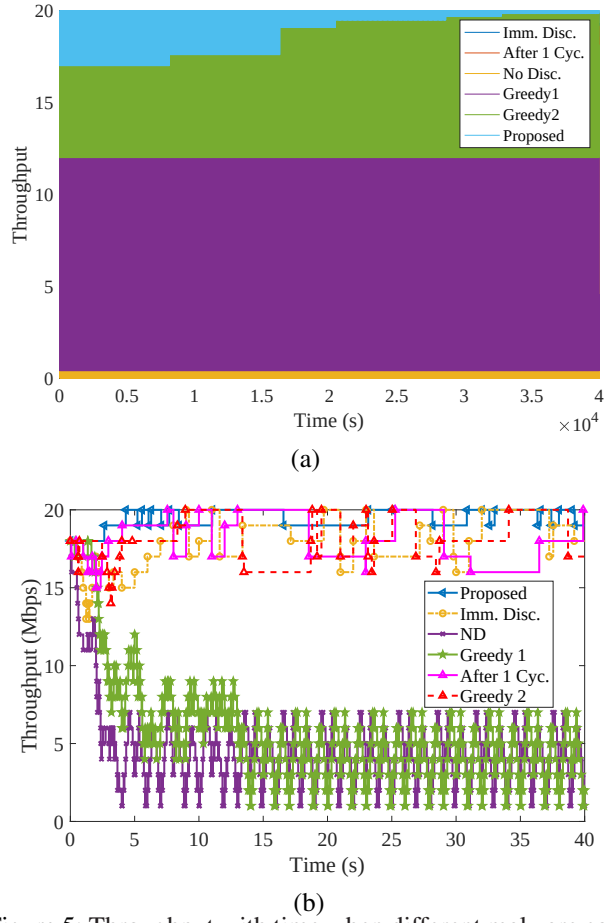

Figure 5: Throughput with time when different malware confinement techniques are employed: (a) experiment executed for nearly 12 hours to observe convergence; (b) zoom-in snapshot.

3) Network Performance Comparison

Figure 5 presents the network performance (throughput) of the proposed epidemic predictive controller based malware confinement provided with real-time malware infection, state of the nodes in the network, and the performance when other heuristic methods are deployed in the IoT network as a defense for malware propagation. The deployed heuristics are: disconnect the node as soon as the malware is detected (denoted as 'Imm. Disc.'); disconnect the node after 1 cycle of malware propagation (denoted as 'After 1 cyc.'); no defense in the network ('ND'); greedy algorithms based on the malware infection probability ('Greedy 1') i.e., disconnect

the node if malware is detected by *HMD* with a probability higher than a threshold (0.75); and based on the degree of infected node ('Greedy 2') i.e., disconnect infected node with highest neighbors. Figure 5(a) shows the throughput w.r.t. time executed for 12 hours. As seen the proposed technique leads to a better convergence compared to other techniques. It needs to be noted that the Figure 5 looks like plain bars, however, it is not the case and has significant overlaps with some other techniques. Figure 5(b) shows a zoom-in snapshot of the throughput. One can observe that with the proposed technique, the throughput remains close to the maximum bound. The immediate disconnect and disconnecting the node after one cycle achieve the next best throughput, respectively. Similarly, the greedy 1 and no disconnect performs worse, as the malware propagates through the network. The greedy 2 i.e. disconnecting the infected node with highest neighbors performs better than Greedy 1 indicating that infection spreads much faster than quarantining the malware and malware propagation has more impact on the overall network throughput. The network throughput obtained for the experiments deployed with various schemes is listed in Table 1 with first row describing the scheme for malware confinement, and the second row providing the overall network throughput (Thr in Mbps). Nearly 200% throughput is achieved compared to network without having any defense for malware propagation defense. Similarly, up to 160% is achieved with proposed malware epidemic control provided with real-time infection data. Additionally, the number of infected nodes with a probability more than 70% are averaged for all the conducted experiments and presented third row ($\#Inf.$) of Table 1. It can be seen that the number of infected nodes with proposed control mechanism is nearly 50% less compared to the immediate disconnect, which has the second best throughput compared to the proposed method.

Table 1: Network throughput and infected nodes (average) under different malware confinement schemes

Technique	**Proposed**	Imm. disc.	After 1 cycle	No disc.	Greedy 1	Greedy 2
Thr. (Mbps)	**596.0528**	460.9631	418.9087	182.143	310.6328	243.2209
# Inf.	**1.27**	2.48	3.32	-	14.79	2.71

C. EVALUATION OF MALWARE STATE TRANSLATION

For the evaluation of MA-DGT, we use the network data (network topology, state of nodes, infection rate, recovery rate, and decay rate, network topology after deploying stochastic controller) and the output of stochastic controller to train the MA-DGT. The number of such data samples used are 343 with 200 of them used for training and the rest used for testing. Furthermore, three datasets with different network sizes (e.g. 40, and 20 termed as I, and II) are explored. The number of feature maps in each of the layers of MA-DGT are $5-10-10-10-5-1$ in generator and $5-10-10-20$ in discriminator. The learning rate is chosen as 0.0001 for both generator and discriminator and in the training process. The mini batch for optimization is 20. All experiments are conducted on a 64-bit machine with Nvidia GPU (GTX 1070,1683 MHz, 8 GB GDDR5). The model is trained by ADAM optimization algorithm.

To evaluate the performance of the MA-DGT, we compare the generated graphs to the label graphs (output of stochastic controller) based on five graph property metrics. First, the Accuracy is utilized to evaluate the ratio of edges that are correctly predicted by computing the percentage of correctly generated edges among all the possible edges. To measure the edge weights which are continuous values, RMSE (root mean squared error), R2 (coefficient of determination score), Pearson and Spearman correlation are computed between weights of generated and real target graphs. To validate the superiority of the MA-DGT over other existing graph generation methods in deep learning domain, we conduct additional two comparison experiments on two models: 1) GraphVAE [82]: a probability-based graph generation method for small graphs and 2) GraphRNN [83]: a recent graph generation method based on sequential generation with LSTM model. The average evaluation results of the whole testing samples are shown in Table 2

Table 2: Evaluation of proposed MA-DGT for malware epidemics for two network sizes

Dataset	Method	Accuracy	RMSE	R2	Pearson	Spearman
II	GraphRNN	70.54%	44.15	0.44	0.29	0.24
	GraphVAE	60.60%	49.09	**0.73**	0.16	0.17
	MA-DGT	**90.33%**	**21.51**	0.13	**0.81**	**0.86**
I	GraphRNN	83.97%	42.13	0.16	0.23	0.19
	GraphVAE	81.19%	45.92	0.39	0.32	0.35
	MA-DGT	**94.53%**	**23.45**	**0.63**	**0.80**	**0.79**

As shown in Table 2, the MA-DGT outperforms the other two methods in almost all the metrics. Specifically, in terms of edge accuracy, the graph generation methods (GraphRNN and GraphVAE) cannot handle the graph translation and got low accuracy of around 60% at dataset II, and 80% at dataset I, while MA-DGT achieves better results with accuracy of 90%. The MA-DGT outperforms the other two methods with an average 76.8% superiority in RMSE, 60.1% in Pearson correlation and 73.2% in Spearman correlation.

To further validate the effectiveness of the MA-DGT, we show two cases for analysis in Figure 6. These two cases have different parameters. Each line is a translation case consisting of input graph, label or target graph and the generated graph. The input graph has many connections and the after de-infected process, some connections are deleted as shown in label graphs. In this malware problem, the performance of translation can be directly viewed from the topology comparison between labeled graphs and the generated graphs. In Figure 6, each of the circles represent a node and the number in the circle refers to node index. The edges 2-11, 2-13 and 7-14 are deleted in the labeled graph and this also happens in the generated graph, while for both label and generated graphs, the 6-15, 9-3, 10-3 edges remain in the translation process. This validates that the translation process learns the way of how to de-infected the malware circuits depending on different parameters and nodes states.

Lastly, we evaluate the time and memory cost of the proposed MA-DGT to determine the scalability of the tech-

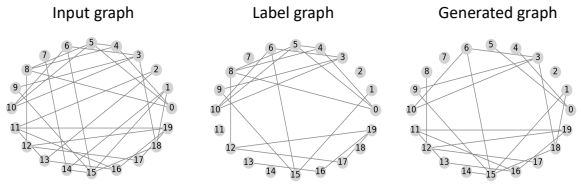

Figure 6: Two cases for malware state translation.

nique. It needs to be noted that we compare the inference time of MA-DGT with the stochastic control technique. Also, we compare the resource consumption with that of the proposed stochastic control technique (Section III). The time and memory consumption are depicted in Figure 7. As the training of MA-DGT is performed offline, it can be ignored in the given work based on the objective, however, it has to be noted that the training of MA-DGT is inherently expensive. Given the objective of this work, improving learning convergence is out of scope of this work. It needs to be noted that the stochastic technique does not require training. Figure 7 presents the comparison of MA-DGT based technique and stochastic optimization solution (represented as ED - Epidemic controller) in terms of time and memory consumption. One can observe that the time consumption and memory requirements for MA-DGT based malware epidemic control is smaller compared to the stochastic technique. However, for smaller networks (such as 20 nodes), the stochastic optimization (EC) solutions outperform the MA-DGT, due to the involved complexity in MA-DGT. Furthermore, one can note that the memory requirements for MA-DGT are nearly constant (irrespective of network size), whereas the memory requirement grows exponentially for the stochastic technique. This proves the ability to perform the malware epidemic control for large scale networks using the proposed MA-DGT effectively with training data obtained from the stochastic controller.

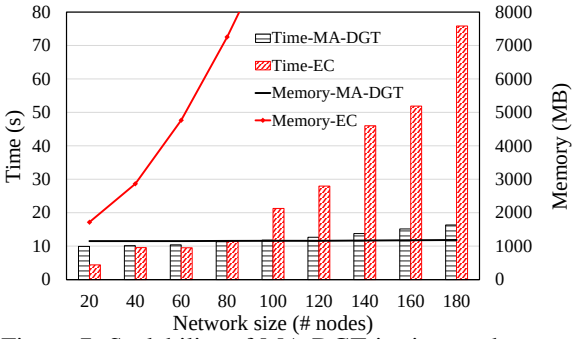

Figure 7: Scalability of MA-DGT in time and memory cost and comparison with stochastic technique (Testing phase)

VI. RELATED WORK

We differentiate the proposed work from state-of-the-art both in terms of malware confinement and graph translation.

A. MALWARE CONFINEMENT IN IOT NETWORK: COMPARISON WITH THE STATE-OF-THE-ART

Controlling epidemics or infection spreading is one of the widely researched areas. A recent survey on controlling epidemics on networks [43] highlights the currently noticeable gap in the literature needed to solve the problems we propose. Specifically, the overwhelming majority of works that study the theoretical containment or eradication of a disease or malware only consider deterministic approximations of the actual stochastic dynamics, and also only consider one-time optimal resource allocation problems rather than real-time feedback control strategies.

These relaxations to the problem arise from the inherent complexity in networked epidemic systems: the stochastic dynamics which describe the fundamental aspects of the process entangle the components of the system's state, making their analysis difficult. This issue is typically addressed by using a mean-field type approximation (see, e.g., [84]), in which the random variables studied in the process are assumed to be uncorrelated with each other across time. This, in effect, assumes that the problems introduced by entanglement have only a weak effect on the evolution of the system, and as such, the resulting approximated probabilities are representations of the statistics of the process itself.

The work in [44] presents a framework in which the spreading infection (Worm) intends to kill the infected node in order to refrain from getting killed and detected by the defender. Despite effective, this work differs in terms of considering both node-level and network-level attributes to control the epidemic. Further, the assumption that malware need to kill an infected node to minimize the spread is not required in this work.

For sufficiently simple epidemic processes (see, e.g., [85]) the mean-field approach yields dynamics which provides an upper-bound for the expectation of the stochastic process. However, in general, this is not the case. Indeed, even for simple models with multiple compartments, simulations have shown standard mean-field approximations to be unreliable proxies for the statistics of the underlying stochastic process (see, e.g., [86]). Ultimately, most works that consider containing or eradicating any type of spreading process on a network finish with an analysis of the deterministic mean-field approximation dynamics, and it remains unclear exactly how this analysis connects to the original stochastic dynamics. Unfortunately, we do not have this luxury as the actual stochastic dynamics are what will ultimately drive the spreading of the infection, meaning we must consider the exact stochastic dynamics rather than an approximation. Instead, in this work we develop a tractable real-time stochastic controller that can operate on the exact stochastic dynamics. For IoT networks, techniques such as HoneyPot [87], deception [88] are proposed. These techniques are effective in detecting the malware or security breaches in the network by luring the attackers to probe the decoys. However, they lack a sophisticated mechanism to defend against attacks [89].

B. GRAPH TRANSLATION: COMPARISON WITH THE STATE-OF-THE-ART

In recent years, there are emerging research works on neural networks for graph learning, which has a wide range of applications, e.g. security, computer networks, bio-information and social network connectivity determination [90], [48], [91], [92], [93], [94], [95]. As one category of graph learning domain, graph generation has attracted a great attention, which is highly tailored to only address the graph generation in specific type of applications such as molecule generation. Generic graph generation can handle general graphs that are not restricted to specific applications. Existing works all proposed in the most recent year, which are based on VAE [82], [96], generative adversarial nets (GAN) [97] and others [98], [83]. Specifically, A graph net proposed by [98] generates nodes and edges sequentially to form a whole graph, which is sensitive to the generation order and time-consuming for the large graphs. Works in [82] and [96] are all new variational autoencoders in parallel for whole graph generation, though they typically only handle very small graphs (e.g., with ≤ 50 nodes) and cannot scale well in both memory and runtime for large graphs. Different from the above methods, GraphRNN represents graphs as sequences using different node ordering, and then builds an autoregressive generative model on these sequences with LSTM model.

A new graph learning problem "graph translation" is initially proposed in [48]. In real-world applications, rare events such as catastrophic events and cyber-attacks only occurred in few locations (can be treated as different local networks), but all the other locations without historical rare event occurrence also need the capability of early detection and reaction. Therefore, it is highly desirable to learn the shared pattern of rare events in those locations with historical rare events, and then proactively synthesize the event occurrence situations exclusive to all the other locations. Graph translation problem aims to learn and transfer the shared complex patterns across different networks with different structures and sizes. However, in the GT-GAN (Graph Translation Generative Adversarial Network) proposed by [48], the translation process only depends on the topology of the input graph and does not consider the node attributes of the input graphs. Whereas, in our problem, the translation process depends not only on the graph topology, but also depends on the node attributes and controlling parameters(e.g. inference rate, reconstruction rate and decay rate). To deal with this problem, we propose the MA-DGT (Multi-Attributed Deep Graph Translation), which is a novel deep graph translation structure considering the node attribute and translation parameters.

VII. CONCLUSION

A single compromised node in an IoT network can infect other nodes in the network, as a consequence of malware spread. The existing works on malware confinement are either too theoretical or does not have any malware detection strategy nor considers true infection state of the nodes. In contrast, in this work we propose a novel practical solution for securing IoT networks against malware epidemics. To this aim, a lightweight runtime malware detector is deployed on IoT nodes for detecting malware with high accuracy. Unfortunately, since the malware detection algorithms are not perfect, their outputs cannot be immediately used in theoretical optimal control problems. Instead, we use the outputs of malware detector to generate probabilistic outputs about the infection state rather than binary deterministic ones that can be used (with small modifications) in a rolling horizon optimal control algorithm. The epidemic model predictive controller considers the network connectivity, estimated infection state of the nodes, performance requirements of the network and performs a stochastic optimization to minimize the infection spread in the network while limiting the losses to the network performance. The deployed malware detector achieves a malware detection accuracy of $\sim$92% on average. The proposed malware epidemic control method achieves a throughput of up to 160% compared to heuristic based approaches. Furthermore, for the purpose of scalability, a Multi-attributed graph translation technique is proposed.

References

[1] A. K. Sikder et al., "A survey on sensor-based threats to internet-of-things (IoT) devices and applications," CoRR, vol. abs/1802.02041, 2018.
[2] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," IEEE Transactions on Emerging Topics in Computing, vol. 5, no. 4, pp. 586–602, Oct 2017.
[3] T. Abera et al., "Things, trouble, trust: On building trust in iot systems," in ACM/EDAC/IEEE Design Automation Conference (DAC), 2016.
[4] J. Wurm et al., "Security analysis on consumer and industrial iot devices," in Asia and South Pacific Design Automation Conference (ASP-DAC), Jan 2016.
[5] S. Koley and P. Ghosal, "Addressing hardware security challenges in internet of things: Recent trends and possible solutions," in IEEE International Conference on Ubiquitous Intelligence and Computing, 2015.
[6] K. Yang et al., "Protecting endpoint devices in iot supply chain," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov 2015, pp. 351–356.
[7] T. Xu et al., "Security of IoT systems: Design challenges and opportunities," in IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD), Nov 2014.
[8] M. Labs, "Infographic: Mcafee labs threats report," Nov 2018.
[9] S. Cheng et al., "Traffic-aware patching for cyber security in mobile IoT," IEEE Communications Magazine, vol. 55, no. 7, pp. 29–35, July 2017.
[10] M. B. Barcena and C. Wueest, "Insecurity in the internet of things," Whitepaper, Apr 2015.
[11] S. Shukla et al., "Stealthy malware detection using rnn-based automated localized feature extraction and classifier," in IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2019.
[12] S. Shukla et al., "Rnn-based classifier to detect stealthy malware using localized features and complex symbolic sequence," in IEEE International Conference on Machine Learning and Applications (ICMLA), 2019.
[13] J. Granjal et al., "Security for the internet of things: A survey of existing protocols and open research issues," IEEE Communications Surveys Tutorials, vol. 17, no. 3, pp. 1294–1312, Jan 2015.
[14] P. Chen et al., "Decapitation via digital epidemics: a bio-inspired transmissive attack," IEEE Communications Magazine, vol. 54, no. 6, pp. 75–81, June 2016.
[15] Kaspersky, "Attacks with exploits: From everyday threats to targeted campaigns," White Paper, pp. 1–12, 2017. [Online]. Available: {https://media.kaspersky.com/en/business-security/enterprise/KL_Report_Exploits_in_2016_final.pdf}
[16] T. Wang et al., "A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing," IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4831–4843, 2018.
[17] C. Miao et al., "Privacy-preserving truth discovery in crowd sensing systems," ACM Trans. Sen. Netw., vol. 15, no. 1, Jan 2019.

[18] M. Z. A. Bhuiyan et al., "Maintaining the balance between privacy and data integrity in internet of things," in International Conference on Management Engineering, Software Engineering and Service Sciences, 2017.

[19] T. Wang et al., "Preserving balance between privacy and data integrity in edge-assisted internet of things," IEEE Internet of Things Journal, vol. PP, pp. 1–1, 2019.

[20] M. H. R. Khouzani et al., "Optimal Control of Epidemic Evolution," in IEEE INFOCOM, no. i, Shanghai, China, 2011.

[21] A. Dinaburg et al., "Ether: Malware analysis via hardware virtualization extensions," in ACM Conference on Computer and Communications Security, 2008.

[22] G. Gu et al., "BotHunter: Detecting malware infection through ids-driven dialog correlation," in USENIX Security Symposium, 2007.

[23] Y. Fan et al., "Gotcha - sly malware!: Scorpion a metagraph2vec based malware detection system," in ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018.

[24] M. Ozsoy et al., "SIFT: A low-overhead dynamic information flow tracking architecture for SMT processors," in ACM Int. Conference on Computing Frontiers, 2011.

[25] H. Yin et al., "Panorama: Capturing system-wide information flow for malware detection and analysis," in ACM Conference on Computer and Communications Security, 2007.

[26] E. J. Schwartz et al., "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in IEEE Symposium on Security and Privacy, 2010.

[27] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in Network and Distributed Systems Security Symposium, 2003.

[28] M. Ozsoy et al., "Malware-aware processors: A framework for efficient online malware detection," in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2015.

[29] M. Christodorescu et al., "Mining specifications of malicious behavior," in ACM SIGSOFT Symposium on The Foundations of Software Engineering, 2007.

[30] R. Sekar et al., "A fast automaton-based method for detecting anomalous program behaviors," in IEEE Symposium on Security and Privacy, 2001.

[31] M. Kayaalp et al., "SCRAP: Architecture for signature-based protection from code reuse attacks," in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2013.

[32] K. Shen et al., "Hardware counter driven on-the-fly request signatures," in International Conference on Architectural Support for Programming Languages and Operating Systems, 2008.

[33] A. A. Elhadi et al., "Malware detection based on hybrid signature behaviour application programming interface call graph," American Journal of Applied Sciences, vol. 9, no. 3, p. 283, 2012.

[34] N. Patel et al., "Analyzing hardware based malware detectors," in ACM/IEEE Design Automation Conf., 2017.

[35] J. Demme et al., "On the feasibility of online malware detection with performance counters," SIGARCH Comput. Archit. News, vol. 41, no. 3, pp. 559–570, June 2013.

[36] H. Sayadi et al., "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), June 2018, pp. 1–6.

[37] P. D. S. Manoj et al., "Adversarial attack on microarchitectural events based malware detectors," in Design Automation Conference (DAC), 2019.

[38] E. Ronen et al., "IoT goes nuclear: Creating a zigbee chain reaction," in IEEE Symposium on Security and Privacy (SP), 2017.

[39] B. Singh et al., "On the detection of kernel-level rootkits using hardware performance counters," in ACM on Asia Conference on Computer and Communications Security, 2017.

[40] A. Vashist et al., "Indoor wireless localization using consumer-grade 60 ghz equipment with machine learning for intelligent material handling," in International Conference on Consumer Electronics (ICCE), 2020.

[41] Y. LeCun et al., "Deep learning," nature, vol. 521, no. 7553, p. 436, 2015.

[42] P. Isola et al., "Image-to-image translation with conditional adversarial networks," arXiv preprint, 2017.

[43] C. Nowzari et al., "Analysis and control of epidemics: A survey of spreading processes on complex networks," IEEE Control Systems, vol. 36, no. 1, pp. 26–46, Feb 2016.

[44] V. Karyotis and M. Khouzani, "Optimal control based techniques," in Malware Diffusion Models for Wireless Complex Networks. Morgan Kaufmann, 2016, pp. 139 – 154.

[45] K. Drakopoulos et al., "An efficient curing policy for epidemics on graphs," in IEEE Conference on Decision and Control, 2014.

[46] K. Drakopoulos et al., "An Efficient Curing Policy for Epidemics on Graphs," IEEE Transactions on Network Science and Engineering, vol. 1, no. 2, pp. 67–75, 2014.

[47] N. J. Watkins et al., "Inference, Prediction, and Control of Networked Epidemics," in IEEE American Control Conference, 2017.

[48] X. Guo et al., "Deep graph translation," arXiv preprint arXiv:1805.09980, 2018.

[49] K. Khasawneh et al., "Ensemble learning for low-level hardware-supported malware detection," in Research in Attacks, Intrusions, and Defenses, 2015.

[50] P. Rost et al., "Network slicing to enable scalability and flexibility in 5g mobile networks," IEEE Communications Magazine, vol. 55, no. 5, pp. 72–79, May 2017.

[51] R. Pastor-Satorras and A. Vespignani, "Epidemic spreading in scale-free networks," Phys. Rev. Lett., vol. 86, pp. 3200–3203, Apr 2001.

[52] E. Valdano et al., "Epidemic Threshold in Continuous-Time Evolving Networks," Physical Review Letters, vol. 120, no. 6, p. 068302, Feb 2018.

[53] D. Chakrabarti et al., "Epidemic thresholds in real networks," ACM Trans. Inf. Syst. Secur., vol. 10, no. 4, pp. 1:1–1:26, Jan 2008.

[54] Y. Wang et al., "Epidemic spreading in real networks: an eigenvalue viewpoint," in International Symposium on Reliable Distributed Systems, 2003.

[55] (2013) badbios. Last accessed: 07-Aug-2018. [Online]. Available: https://arstechnica.com/information-technology/2013/10/meet-badbios-the-mysterious-mac-and-pc-malware-that-jumps-airgaps/

[56] (2007) Yankee doodle. Last accessed: 07-Aug-2018. [Online]. Available: https://www.symantec.com/security-center/writeup/2000-121914-2303-99

[57] (2017) Magneto. Last accessed: 07-Aug-2018. [Online]. Available: https://magento.com/security/tag/malware

[58] G. H. Weiss and M. Dishon, "On the asymptotic behavior of the stochastic and deterministic models of an epidemic," Mathematical Biosciences, vol. 11, no. 3, pp. 261–265, 1971.

[59] R. H. Norden, "On the distribution of the time to extinction in the stochastic logistic population model," Advances in Applied Probability, vol. 14, no. 4, pp. 687–708, 1982.

[60] R. J. Kryscio and C. Lefévre, "On the extinction of the SIS stochastic logistic epidemic," Journal of Applied Probability, pp. 685–694, 1989.

[61] D. Acarali et al., "Modelling the spread of botnet malware in iot-based wireless sensor networks," Security and Communication Networks, pp. 1–13, Feb 2019.

[62] A. Kumar and T. J. Lim, "Edima: Early detection of iot malware network activity using machine learning techniques," in IEEE World Forum on Internet of Things (WF-IoT), 2019.

[63] M. Bloem et al., "Optimal and robust epidemic response for multiple networks," Control Engineering Practice, vol. 17, pp. 525–533, 2009.

[64] A. Khanafer and T. Basar, "An optimal control problem over infected networks," in Proceedings of the International Conference of Control, Dynamic Systems, and Robotics, Ottawa, Ontario, Canada, 2014.

[65] S. Eshghi et al., "Optimal patching in clustered epidemics of malware," IEEE Transactions on Networking, 2015, to appear.

[66] W. K. Chai, "Modelling spreading process induced by agent mobility in complex networks," IEEE Transactions on Network Science and Engineering, vol. PP, pp. 1–1, 2017.

[67] L.-X. Yang et al., "The impact of patch forwarding on the prevalence of computer virus: A theoretical assessment approach," Applied Mathematical Modelling, vol. 43, pp. 110 – 125, 2017.

[68] N. J. Watkins et al., "Robust economic model predictive control of continuous-time epidemic processes," ArXiv e-prints, July 2017.

[69] A. Lundell, Transformation techniques for signomial functions in global optimization. Åbo Akademi University, 2009.

[70] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar 2014.

[71] J. Nocedal and S. Wright, Numerical optimization. Springer Science & Business Media, 2006.

[72] P. H. J. Nardelli et al., "Efficiency of wireless networks under different hopping strategies," IEEE Transactions on Wireless Communications, vol. 11, no. 1, pp. 15–20, Jan 2012.

[73] H. J. N. Pedro et al., "Throughput maximization in multi-hop wireless networks under a secrecy constraint," Computer Networks, vol. 109, no. 1, pp. 13 – 20, Nov 2016.
[74] (2018) Virustotal intelligence service. Last accessed: 07-Aug-2018. [Online]. Available: www.virustotal.com/intelligence
[75] L. Feng et al., "Dynamical analysis and control strategies on malware propagation model," Elsevier Journal of Applied Mathematical Modelling, vol. 37, no. 16-17, pp. 8225–8236, 2013.
[76] Z. Chen and C. Ji, "Spatial-temporal modeling of malware propagation in networks," IEEE Transactions on Neural Networks, vol. 16, no. 5, pp. 1291–1303, 2005.
[77] C. Fleizach et al., "Can you infect me now?: malware propagation in mobile phone networks," in ACM workshop on Recurring Malcode, 2007.
[78] S. Hosseini et al., "Malware propagation modeling considering software diversity and immunization," Elsevier Journal of computational science, vol. 13, pp. 49–67, Mar 2016.
[79] M. R. Guthaus et al., "MiBench: A free, commercially representative embedded benchmark suite," in IEEE International Workshop on Workload Characterization, 2001.
[80] J. L. Henning, "SPEC CPU2006 benchmark descriptions," SIGARCH Comput. Archit. News, vol. 34, no. 4, pp. 1–17, Sep 2006.
[81] (2018) Virusshare team. Last accessed: 07-Aug-2018. [Online]. Available: www.virusshare.com
[82] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," arXiv preprint arXiv:1802.03480, 2018.
[83] J. You et al., "Graphrnn: Generating realistic graphs with deep auto-regressive models," in International Conference on Machine Learning, 2018, pp. 5694–5703.
[84] P. Van Mieghem et al., "Virus Spread in Networks," IEEE/ACM Transactions on Networking, vol. 17, no. 1, pp. 1–14, Feb 2009.
[85] P. Simon and I. Z. Kiss, "On bounding exact models of epidemic spread on networks," arXiv preprint, pp. 1–18, Apr 2017.
[86] N. J. Watkins et al., "Optimal resource allocation for competitive spreading processes on bilayer networks," IEEE Transactions on Control of Network Systems, vol. 5, no. 1, pp. 298–307, 2018.
[87] D. Fraunholz et al., "An adaptive honeypot configuration, deployment and maintenance strategy," in Int. Conf. on Advanced Communication Technology, 2017.
[88] V. E. Urias et al., "Gathering threat intelligence through computer network deception," in IEEE Symposium on Technologies for Homeland Security (HST), 2016.
[89] Smokescreen, "7 deadly sins âĂŞ how to fail at implementing deception technology," 2018, https://www.smokescreen.io/7-deadly-sins-how-to-fail-at-implementing-deception-technology/.
[90] L. Holder et al., "Graph-based relational learning with application to security," Fundamenta Informaticae, vol. 66, no. 1-2, pp. 83–101, 2005.
[91] M. J. Kusner et al., "Grammar variational autoencoder," arXiv preprint arXiv:1703.01925, 2017.
[92] R. Gómez-Bombarelli et al., "Automatic chemical design using a data-driven continuous representation of molecules," ACS central science, vol. 4, no. 2, pp. 268–276, 2018.
[93] H. Dai et al., "Syntax-directed variational autoencoder for molecule generation," in International Conference on Machine Learning, 2018.
[94] W. Jin et al., "Junction tree variational autoencoder for molecular graph generation," arXiv preprint arXiv:1802.04364, 2018.
[95] S. Cavallari et al., "Learning community embedding with community detection and node embedding on graphs," in ACM Conference on Information and Knowledge Management, 2017, pp. 377–386.
[96] B. Samanta et al., "Designing random graph models using variational autoencoders with applications to chemical design," arXiv preprint arXiv:1802.05283, 2018.
[97] A. Bojchevski et al., "Netgan: Generating graphs via random walks," arXiv preprint arXiv:1803.00816, 2018.
[98] Y. Li et al., "Learning deep generative models of graphs," arXiv preprint arXiv:1803.03324, 2018.
[99] A. Azmoodeh et al., "Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning," IEEE Transactions on Sustainable Computing, vol. 4, no. 1, pp. 88–95, Jan 2019.
[100] E. Ronen and A. Shamir, "Extended functionality attacks on IoT devices: The case of smart lights," in IEEE European Symposium on Security and Privacy, 2016.
[101] T. Yu et al., "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in ACM Workshop on Hot Topics in Networks, 2015.
[102] Z. K. Zhang et al., "IoT security: Ongoing challenges and research opportunities," in IEEE International Conference on Service-Oriented Computing and Applications, 2014.
[103] A. Garcia-Serrano, "Anomaly detection for malware identification using hardware performance counters," CoRR, vol. abs/1508.07482, 2015.
[104] A. Tang et al., "Unsupervised anomaly-based malware detection using hardware features," in Research in Attacks, Intrusions and Defenses, 2014.
[105] M. B. Bahador et al., "HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in International Conference on Computer and Knowledge Engineering (ICCKE), 2014.
[106] X. Wang et al., "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," ACM Trans. Archit. Code Optim., vol. 13, no. 1, pp. 3:1–3:23, Mar 2016.

A. APPENDIX

MOTIVATIONAL CASE STUDIES

We perform case study to evaluate the need for malware confinement at the network-level and deploying multi-attributed deep graph translation to highlight the security risk in IoT networks.

A. IMPACT OF MALWARE CONFINEMENT ON THROUGHPUT IN IOT NETWORK

We investigate the impact of malware propagation on the network performance if no malware confinement strategy is deployed. To evaluate the impact of malware propagation on network performance (throughput), we consider a small-scale IoT network of 20 nodes with no restriction on malware propagation enforced. The throughput of each node is set to be 1 Mbps. We assume the malware infection model for IoT devices as Susceptible-Infected-Susceptible (SIS). More details on infection model are presented in Section II-B.

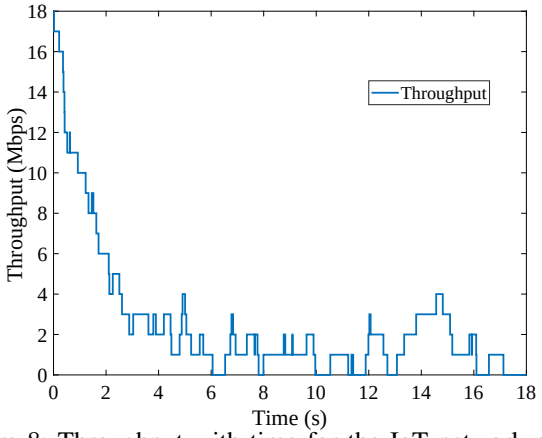

Figure 8: Throughput with time for the IoT network of 20 nodes, but without any malware confinement strategy

In a SIS model, an IoT node is susceptible to infection (malware attack), and can be recovered from infection after certain time i.e. malware can be quarantined and the node is again susceptible for future infections. The impact of malware propagation on the network throughput is depicted in Figure 8. As seen, if no defense mechanism for malware containment is deployed, all the nodes in the network eventually gets infected, reducing the network throughput (to zero at some time instants). It needs to be noted that the observed increase in throughput is due to the considered SIS model i.e. a node recovers from infection after quarantining. This shows that an effective malware epidemic control has to be deployed to maintain the network performance. For this case study, the infection rate is 0.3 with a recovery rate of 0.1. These parameters are chosen based on the observed infection and recovery rates for the employed virus samples. However, it is observed that the trend of network throughput is same for all the different experimented malware. More details on experimental setup are in Section V-A.

B. NEEDS OF CONVOLUTION OPERATIONS ON DYNAMIC AND ARBITRARY GRAPH STRUCTURES

Existing deep learning models for the translation of data [42] (e.g. image data, audio and text) assumes the data structure is fixed before and after the translation. For instance, in case of the image translation, only the intensity of pixels vary, whereas the pixel properties such as the number of neighboring pixels are retained, as shown in Figure 9(a). However, in the problem of malware propagation in IoT networks, there are substantial differences: 1) the graph topology and connectivity are highly flexible, as shown in Figure 9(b). And hence, the deep learning models designed based on convolution operations on grids (such as images) are not directly applicable to IoT network problem. 2) Further, in graph translation problem, both the nodes' statuses and the connectivity structure among the nodes change, as shown in Figure 9(b), while in image translation, the node connectivity structure is always fixed, as shown in Figure 9(a). Hence, new methodologies are imperative for graph translation that predicts both node status as well as node connectivity is needed.

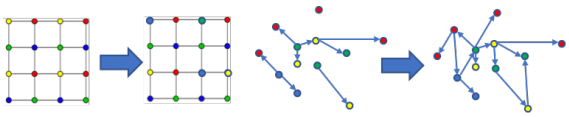

Figure 9: Translation on graph-structured data requires different convolution operations. (a) In image translation, the connectivity among nodes is always grid-structure and will not change. (b) In multi-attributed graph translation problem, the connectivity among nodes can be any generic graph and can change before and after the translation.

HARDWARE-ASSISTED RUNTIME MALWARE DETECTION

To perform effective node-level malware detection in IoT network, we adopt a lightweight, hardware-assisted malware detection technique (HMD) recently proposed in [34], [36]. We briefly describe the adopted technique below.

C. OVERVIEW OF HARDWARE-ASSISTED MALWARE DETECTION

The general overview of deployed malware detector (HMD), depicted in Figure 10, adopted from [34], [36]. It comprises of feature selection, and runtime malware detection stages. Feature selection is performed offline, and malware detection is performed online.

1) Feature Selection

For runtime malware detection, we employ HPC traces in this work. To alleviate hardware overhead and facilitate runtime malware detection irrelevant data (unneccessary HPC features) is identified and removed using a feature reduction algorithm and as such only a subset of HPCs that represent the most critical features required for malware detection are

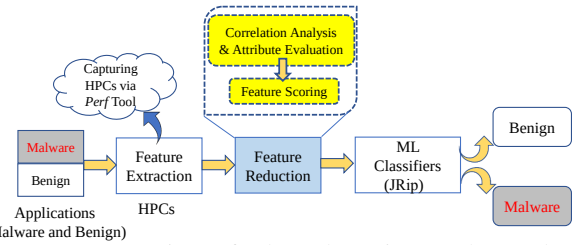

Figure 10: Overview of adopted runtime malware detector *HMD*

selected. This feature selection process is performed offline to determine the most critical microarchitectural events to be captured by the limited available HPCs.

We collected 44 possible diverse microarchitectural events using the available HPCs for the employed IoT devices by repeating the experiments multiple times. As the maximum number of HPCs that can be collected in one iteration is limited to number of available on-chip HPCs (4 in the employed hardware), we ran experiments multiple iterations to obtain all the 44 HPCs. Further, we apply a feature reduction technique to determine the critical HPC events. For feature reduction, we apply "Correlation Attribute Evaluation" to rank the most critical HPC events. Correlation evaluation algorithm calculates the Pearson correlation coefficient between each attribute and class, as given below.

$$\rho(i) = \frac{cov(Z_i, C)}{\sqrt{var(Z_i)\, var(C)}} \quad i = 1, ..., 44 \qquad (18)$$

where ρ is the Pearson correlation coefficient. Z_i is the input dataset of event i ($i = 1, \ldots, 44$). C is the output dataset containing labels, i.e. "Malware" or "Benign" in our case. The $cov(Z_i, C)$ measures the covariance between input data and output data. The $var(Z_i)$ and $var(C)$ measure variance of both input and output datasets, respectively. Based on the ranking of ρ, top 8 features are selected for analysis, given in Table 3. The features are ranked based on their importance and relevance to the target variable through the feature scoring process. The reduced set of prominent features include HPCs representing pipeline front-end, pipeline back-end, cache subsystem, and main memory behaviors which are influential in the performance of standard applications.

Table 3: Microarchitectural (HPC) events of high priority for runtime malware detection

Rank	Event name	Rank	Event name
1	Branch Instructions	5	dTLB_store_misses
2	Branch Loads	6	LLC_prefetch_misses
3	iTLB_load_misses	7	L1_dache_stores
4	dTLB_load_misses	8	cache_misses

Depending on the available number of HPCs, these selected features or events are monitored online on the IoT devices and are provided to the deployed ML classifier to classify and detect malware from benign applications, as described below.

2) Malware Detection by ML Classifiers

Once the key features are selected, they are used to train the ML classifiers in the *HMD*. For evaluation, we experimented various ML classifiers and compare them in terms of malware detection accuracy, hardware overhead, power consumption, and the time required to detect malware (latency). The running application is profiled every 1ms i.e. non-trivial HPCs are collected continuously at 1ms interval and fed to the ML classifier. A k-fold (k=10) validation is employed in this work for evaluating and comparing the malware detection accuracy of different classifiers. The ML classifiers are trained with the non-trivial HPCs and is performed offline.

For the inference and runtime malware detection, the critical HPCs listed in Table 3 are captured during application execution and provided as input to the ML classifiers. Based on the derived model in the training phase, the ML classifiers provide information regarding the existence of malware. As the malware detection is performed on individual nodes, it is independent of the network topology. The deployed epidemic controller for malware confinement requires an estimate of infection rather than deterministic infection state of the node, the *HMD*'s output will be fed to the epidemic controller.

Based on these experimental evaluations (presented in -D), *HMD* employing JRip ML classifier that has relatively smaller area (80× lower than neural network (MLP)), lower latency (1 clock-cycle) for malware detection is deployed on the IoT devices in this work. We would like to emphasize that HMD is not a contribution of this work nor the proposed malware confinement is limited by HMD. As the devices we experimented in this work host HPCs, we adopted this methodology. However, depending on the configuration of IoT devices, one can adopt other techniques such as [99]. The critical part is to employ a node-level malware detector whose information (malware estimates) can be fed to the proposed malware confinement solution.

D. EVALUATION OF ADOPTED MALWARE DETECTION

We evaluate the deployed *HMD* with different ML classifiers in terms of malware detection accuracy, resource consumption, and processing overheads to verify the suitability for runtime malware detection on IoT devices. A 10-fold cross-validation is used to verify the performance of malware detection with reduced features. For the detection accuracy, we calculate the percentage value of samples that are correctly classified. The experimental setup is same as that described in the earlier section.

Table 4 presents the 10-fold validation of the *HMD*'s performance and the silicon overhead incurred by the malware detector that employs 4 HPCs. As the software implementation of ML classifiers for malware detection is slow, in the range of tens of milliseconds which is an order of magnitude higher than the latency needed to capture malware at runtime [34], hardware implementation is performed in this work. The deployed HMD's hardware footprint is evaluated on a FPGA for a fair comparison, as the experimented het-

Table 4: Evaluation of different ML classifiers when deployed in HMD using 4 HPCs

Classifier	Accuracy (%)	Area (%)	Power (mW)	Latency (@10ns)	F1-score
MLP	93.03	41.5	0.78	93	0.93
JRip	**91.08**	**0.2**	**0.28**	**1**	**0.92**
Logistic Reg.	92.21	19.9	0.55	58	0.92
SVM	81.55	4.1	0.42	13	0.82
J48	92.62	0.9	0.26	3	0.93
SGD	92.21	4.1	0.39	13	0.92

erogeneous IoT devices have different hardware resources available. We use Vivado HLS compiler to develop the HDL implementation of the classifiers (*HMD*) and deploy on Xilinx Virtex 7 FPGA. FPGA is a target in our study, as few modern microprocessors have on-chip FPGAs available for programmable logic implementation. Such arrangement makes it feasible to implement reprogrammable low-level malware detection logic (ML model) which can detect malware by reading the CPU HPCs through the shared memory bus. Latency unit is represented in terms of the number of clock cycles (cycles @10 ns) required to classify input vector. In order to compare the area overhead of the implemented hardware-based ML classifiers, we consider the OpenSPARC implementation as a reference and calculate the area overhead relative the core size. The area is a function of total number of utilized LUTs, FFs, and DSP units inside Virtex 7 FPGA.

One can observe from Table 4, that among all the experimented classifiers, complex techniques such as multi-layer perceptron (MLP) i.e. neural network delivers the highest performance of 93.03%, but incurs a large area overhead and delays. To be able to accommodate on IoT devices, we perform malware detection with small footprint and overhead, yet obtaining a good accuracy. we chose and deploy JRip based HMD on each of the nodes for malware detection. It needs to be noted that the underlying system architecture (instruction set and pipeline) is not modified, rather a separate unit that utilizes existing architecture is designed.

E. MALWARE DETECTION AT NODE-LEVEL: COMPARISON WITH THE STATE-OF-THE-ART

Detection of malware with software based approaches (including off-the-shelf anti-virus) technique have set backs of are large runtime, inefficient detection based on signature, and complexity which makes it an unattractive solution for IoT networks [100], [101], [102]. In response, hardware-based malware detection is proposed, which will be reviewed here. The work in [35] was the first study that proposed to utilize the HPC data for malware detection and demonstrated the effectiveness of offline machine learning algorithms in malware classification. They showed high detection accuracy results for Android malware by applying complex ML algorithms, namely Artificial Neural Network (ANN) and K-Nearest Neighbor (KNN). This work lacks runtime malware detection, and mostly applicable for larger systems due to the resource consumption of the employed classifier and the number of HPCs used.

The researchers in [103] and [104] discussed the feasibility of employing unsupervised learning method on low-level features to detect Return-oriented programming (ROP) and buffer overflow attacks by finding an anomaly in the hardware performance counters' information. Although unsupervised algorithms are more effective in detecting new malware and attacker evolution, they are complex in nature demanding more sophisticated analysis, computational overheads. In a different study [28], Ozsoy and et al., used sub-semantic features to detect the malware. Additionally, they suggested changes in microprocessor pipeline to detect malware in truly real-time nature. The discussed computational and processing overheads are too high to be adopted for IoT devices. Additionally, changing microprocessor pipeline for IoT devices is not a cost-effective solution. In contrast, adopted *HMD* does not require any change in processor pipeline and lightweight in nature. The work in [105] collected hardware performance counters to construct support vector machine (SVM) detectors to identify malicious programs in real-time. The SVMs are heavy-weight classifiers and incurs heavy computational overheads, making them not a good option to be deployed on IoT devices. In contrast, we employ a simple ML classifier with less number of HPCs for malware detection.

The work in [49] uses logistic regression to classify malware into multiple classes and trained a specialized classifier for detecting malware class. They further used specialized ensemble learning to improve the accuracy of logistic regression. Despite good accuracy, the computational overheads posed are high, and employs large number of HPCs for classification. One of the recent works [106] uses "sample-locally-analyze-remotely" technique, where the HPCs are collected locally, but analyzed on a server. Compressed sensing is utilized to minimize the communication bandwidth. This technique though mitigates the on-chip processing overheads, the communication costs are still high for IoT devices and is not effective for IoT networks, as the malware propagation in network is faster than the time to communicate with the sever. In contrast to the existing works, the deployed malware detector in this work employs one single lightweight ML classifier ('JRip'), employing limited number of HPCs, with low computational overheads, and suitable for runtime malware detection. Most importantly, the utilized malware detector [34] is devised in order to suit the needs of IoT devices.

A. MALWARE PROPAGATION EVALUATION

Throughput with Malware Propagation

Figure 4 presents the overall network performance (throughput) under different malware infection and recovery rates. With the increase in recovery rate, the network throughput increases and is highest at lower infection rate and high recovery rate. However, the infection rate increase hampers the overall throughput, and it has higher impact on the throughput compared to recovery rate. It can be seen that

the right-most corner in Figure 4 i.e. high infection and high recovery rate has slightly lower throughput compared to the high recovery and low infection rate. This shows that runtime malware detection and quarantining (recovering) is not the panacea to have secure IoT networks, as infection (propagation) has higher impact than recovery (quarantining).

Throughput with Malware Threat Level

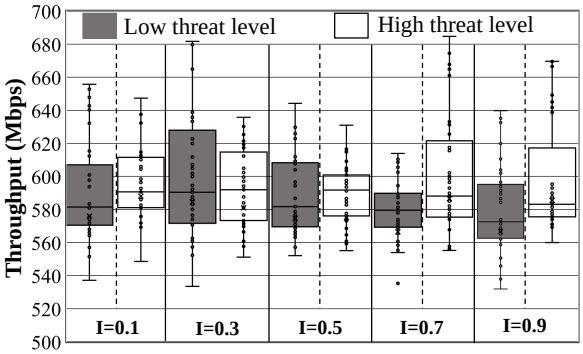

Figure 11: Overall network throughput when nodes are deployed with malware of different threat levels and has different infection rates

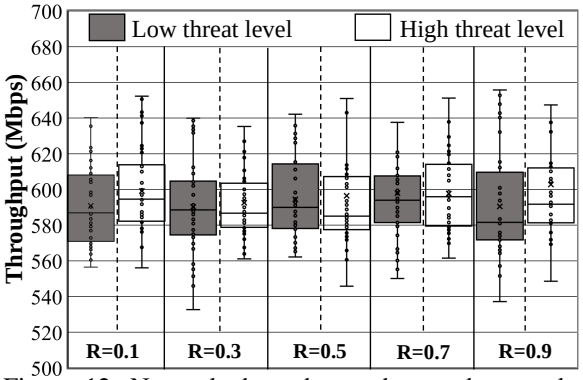

Figure 12: Network throughput when nodes are deployed with malware of different threat levels and has different recovery rates

We evaluate the throughput of the network when proposed solution is employed under different malware threat levels (σ). Figure 11 shows the impact of network throughput with the infection rate for malware threat levels (σ) at a constant recovery rate of 0.5. Increase in the infection rate leads to a reduced throughput, irrespective of malware threat level. With the proposed solution, in the presence of malware with smaller threat level, the network lets the disease to propagate to a certain extent, as long as the decay rate r is satisfied, leading to a lower throughput compared to the case with high malware threat level. In case of malware with higher threat level, the proposed solution shuts the links or reduces the traffic aggressively to control the epidemics and reduce the impact of malware, leading to a higher average throughput. This can be observed from the circles in the box-plot (Figure 11), where each circle represents the throughput achieved in different experiments. Also, in case of higher infection rate, one can see that more experiments lead to throughput higher than (and closer to) the average indicating that the proposed solution maintains the network throughput.

Similarly, we also evaluate the network throughput under different recovery rates and malware threat levels at a constant infection rate of 0.5. Figure 12 shows the impact of network throughput under different recovery rates for malware threat levels (σ). It has been observed that for malware with low threat levels, the average throughput remains constant. However, the maximum throughput achieved increases due to higher recovery rate. However, for malware having higher threat levels, the malware confinement equipped with faster recovery lead to an increased average throughput obtained across the experiments conducted.