ADRIoT: An Edge-Assisted Anomaly Detection Framework Against IoT-Based Network Attacks

Ruoyu Li, Qing Li, *Member, IEEE*, Jianer Zhou, and Yong Jiang, *Member, IEEE*

***Abstract*—Internet of Things (IoT) has entered a stage of rapid development and increasing deployment. Meanwhile, these low-power devices typically cannot support complex security mechanisms and, thus, are highly susceptible to malware. This article proposes ADRIoT, an anomaly detection framework for IoT networks, which leverages edge computing to uncover potential threats. An edge is empowered with an anomaly detection module, which consists of a traffic capturer, a traffic preprocessor, and a collection of anomaly detectors dedicated to each type of device. Each detector is constructed by an LSTM autoencoder in an unsupervised manner that requires no labeled attack data and is able to handle emerging zero-day attacks. When a device connects to the edge, the edge will fetch the corresponding detector from the cloud and execute it locally. Another problem is the resource constraint of a single edge device like a home router hinders the deployment of such a detection module. To mitigate this problem, we design a multiedge collaborative mechanism that integrates the resource of multiple edges in a local network to increase the overall load capacity. The evaluation demonstrates that ADRIoT can detect various IoT-based attacks effectively and efficiently, showing that ADRIoT can feasibly help build a more secure IoT environment.**

***Index Terms*—Anomaly detection, edge computing, Internet of Things (IoT) service, machine learning (ML).**

I. Introduction

THE PROLIFERATION of the Internet of Things (IoT) is expected to reach 29 billion connected devices by 2022 [1]. Meanwhile, the rapidly growing number of insecure connected things also provides an accessible attack surface for network attackers. On the one hand, a large proportion of IoT products are typically low power and low functional [2], which hinders the utilization of complicated security mechanisms; on the other hand, though the IoT market size is anticipated to reach U.S. $54 billion by 2022 [3], the issues of outdated and vulnerable firmware and software will still exist for a long time [4]. Furthermore, ordinary consumers usually lack expert knowledge or awareness of IoT security, such as leaving default credentials unchanged whereas attackers can easily crack by a dictionary attack [5]. These threats offer adversaries an open playground for malware spreading, especially the IoT botnet propagation. In 2016, an IoT botnet Mirai controlled over 600 000 connected things by exploiting default Telnet passwords and launched an overwhelming 620 Gb/s Distributed Denial-of-Service (DDoS) attack [6], [7]. Specifically, after the source code of Mirai was released, the arise of its variations is making the environment even worse.

To mitigate the impact of IoT-based network attacks, the network intrusion detection system (NIDS) is commonly used, including signature-based and recent machine learning (ML)-based approaches [8]. However, due to the heterogeneousness of IoT devices, it is nearly impossible to profile a universal model for all types of devices. Moreover, these detection systems are often incapable of capturing zero-day attacks or unseen malicious behaviors that are not included in the signature database or training data set.

The location for the deployment of the detection/monitoring system for IoT networks is another undetermined issue. One thought is that such systems should be as close to the source of traffic data as possible to trigger early-stage alarms before huge damages are caused. In an IoT botnet, devices serve as attack sources, receiving commands from the botmaster and undermining other entities. Given this observation, *edge computing* is a promising method to bring detection capability to the source of potential IoT botnets. Several studies have proposed their intrusion detection frameworks for smart home networks leveraging consumer's home routers as edges [9], [10]. However, the limited scalability of a single router's resources might restrict their use in practice, especially considering the increasingly common use of ML-based systems with high resource consumption. Besides, even if the underlying malicious activities can be detected, these solutions are unreliable in practice due to consumer's lack of security awareness and expert knowledge for further measures.

To solve the challenges above, this article proposes a framework, called ADRIoT, for the detection of IoT-based attacks. The framework serves as Detection-as-a-Service with a cloud–edge architecture to strengthen the security of IoT systems. The cloud is responsible for the generation, storage, and

Manuscript received May 18, 2021; revised September 10, 2021; accepted October 16, 2021. Date of publication October 22, 2021; date of current version June 23, 2022. This work was supported in part by the Guangdong Province Key Area R&D Program under Grant 2018B010113001; in part by the National Key Research and Development Program of China under Grant 2020YFB1804704; in part by the National Natural Science Foundation of China under Grant 61972189; in part by the Shenzhen Key Lab of Software Defined Networking under Grant ZDSYS20140509172959989; and in part by The Social Science Fund of Shenzhen under Grant SZ2020D009. *(Corresponding author: Qing Li.)*

Ruoyu Li is with the Tsinghua–Berkeley Shenzhen Institute, Tsinghua University, Shenzhen 518055, China, and also with Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: liry19@mails.tsinghua.edu.cn).

Qing Li is with Peng Cheng Laboratory, Shenzhen 518066, China, and also with the Southern University of Science and Technology, Shenzhen 518055, China (e-mail: liq@pcl.ac.cn).

Jianer Zhou is with the Institute of Future Networks, Southern University of Science and Technology, Shenzhen 518055, and also with Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: zhouje@sustech.edu.cn).

Yong Jiang is with the Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China, and also with Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: jiangy@sz.tsinghua.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3122148

distribution of anomaly detectors; edge requests, installs, and executes the anomaly detectors received from the cloud. An anomaly detector is an unsupervised model that profiles the normal activities of a device, which eliminates the need for various attack data sets and is able to detect zero-day attacks by capturing the traffic pattern deviations from normality. Since different IoT devices exhibit various behaviors, the cloud will generate one dedicated detector for each type of device. We also attempt to minimize the reliance on feature engineering by using raw traffic data as input to test the generality of our detector. To present its practicality on edge, this article proposes a multiedge collaborative mechanism implemented on the top of Simple Service Discovery Protocol in a specific networking paradigm to optimize the resource utilization. With the help of this mechanism, an overloading edge can discover and request an idle edge for task offloading which significantly releases the resource burden of a single edge.

In general, the contributions of this article are as follows.

1) An edge-assisted framework that discovers potential malicious activities originating from the compromised IoT devices.
2) A novel unsupervised anomaly detection module that requires no labeled attack data set and detects a wide range of IoT-based network attacks.
3) A multiedge collaborative mechanism for task offloading that saves the resource of a single edge.

The remainder of this article is organized as follows. Section II reviews the related work on network anomaly detection, IoT security, and edge computing for IoT security. Section III discusses the threat model. Section IV presents the system design, including the overall architecture, workflow, and the multiedge collaborative mechanism. Section V elaborates the components and workflow of the anomaly detection module. Section VI exhibits the empirical experiments and results. Section VII presents the conclusion.

II. Related Work

This section will present a review of some related work. In this article, network anomaly detection mainly refers to the security-related anomaly rather than performance-related anomaly [11]. Hence, those studies on the detection of network failure and congestion are not deeply discussed.

A. Network Anomaly Detection

Network anomaly detection is a class of network intrusion detection. The primary goal of anomaly detection is to identify the abnormal activity inconsistent with the expected behavior. Compared to other intrusion detection techniques, the anomaly detection system is usually considered better at discerning unknown attacks. Ahmed *et al.* [12] and Bhuyan *et al.* [13] gave a comprehensive survey on network anomaly detection.

At the early stages, statistical methods were widely applied in anomaly detection tasks. Ye and Chen [14] presented a statistical anomaly detection technique using a chi-square statistic to build a profile for normal behavior. Wang and Stolfo [15] presented a statistical technique to detect the worm in the payload by using the n-gram model. The major drawback of these methods is the difficulty to determine the most applicable statistics in various real network environments. As ML becomes popular, ML-based anomaly detection leverages the knowledge of attack signatures and classifies the network traffic into a certain attack category, such as Denial of Service (DoS), probe, User to Root (U2R), Remote to User (R2U), etc. Decision tree [16], SVM [17], [18], K-Means [19], KNN [20], and other algorithms were also applied to the detection of network attacks. In recent years, the state-of-the-art deep learning (DL) technique is gradually showing its capability in anomaly detection. Techniques, such as deep belief nets (DBNs) [21], deep neural networks (DNNs) [22], LSTM [23], and Autoencoder [21] were shown to be effective in network anomaly detection. Some more recent works [24]–[26] exhibited the promise of DL to use raw bytes of packet payload as input for malware detection and traffic classification, which shows the promise of reducing the reliance on handcrafted feature engineering.

Some works discussed the anomaly detection tasks in certain network scenarios. For example, Tang *et al.* [27] proposed a seq2seq model that can be deployed behind an HTTP-based Web application firewall (WAF) to detect network attacks. Mirsky *et al.* [9] proposed a lightweight network anomaly detection framework using ensemble autoencoders for smart home IoT. Rachedi and Benslimane [28] designed a new mechanism based on genetic algorithms to find the optimal solution between security and Quality of Service (QoS) in wireless sensor networks (WSNs). Haddadou *et al.* [29] proposed a novel distributed trust model inspired from the job market signaling model for vehicular ad hoc networks (VANETs) to detect and exclude malicious nodes which spread false and forged data.

B. IoT Security

A technical report [4] revealed that IoT devices are suffering plenty of security issues, including insecure network communications, data leakage, susceptibility to malware infection, etc. The report believes that these threats will still exist for a long time. A few studies [30], [31] measured the network traffic of some mainstream IoT devices, finding that suspicious communications with third-party endpoints are common and botnet malware is a primary threat. An IoT botnet can be utilized to launch DDoS attacks, send spams, mine cryptocurrency, and exploit other weakly configured devices. A notorious example is the Mirai botnet. Antonakakis *et al.* [6] introduced in detail the process of Mirai's propagation to weakly configured IoT devices. Khan *et al.* [32] suggested that the security standards have to be applied at all levels, from technology to consumer/business to legal frameworks, in order to build a more secure IoT ecosystem.

Many previous studies proposed ML and DL-based security frameworks to prevent certain steps of a complete IoT-based attack process [33], such as the detection and prevention of C&C communications [34], botnet communications [35], DDoS launched by IoT [36], [37], etc. Besides, the deployment of such detection mechanisms is also an open issue, such as using a home router as a middlebox for intrusion detection [10].

C. Edge Computing for IoT Security

Edge computing is an emerging network paradigm to fill up the gap between end devices and cloud and it has a natural fit for distributed systems like IoT. Chiang and Zhang [38] summarized the challenges and opportunities of edge computing's applications in IoT. A few studies [39], [40] introduced their edge-based anomaly detection/mitigation strategies against DDoS and other IoT-based attacks. Some papers [41]–[43] promote the detection performance with edge computing's paradigm for data and model aggregation between the edge and cloud, such as using federated learning and transfer learning. Some other studies [39], [44] focus on the optimization of resource utilization for anomaly detection on the low-power edge device. Bendouda *et al.* [45] proposed a Software Defined Network-based (SDN) architecture for IoT, in which a security module can dynamically tune the security services on edges thanks to the SDN paradigm.

III. Threat Model

This article mainly concerns the cyber threats initiated by the compromised IoT devices. Due to the prevalent vulnerability of IoT systems, attackers can get unauthorized access and transform the devices into part of a botnet. According to [6], a typical IoT botnet like Mirai has a set of infrastructure, including a command and control (C&C) server, a loader server, and a report server. It presents a life cycle with seven phases from the perspective of an adversary.

1) Brute force or utilize an exploit to log in a weakly configured device.
2) Gain a shell and forward the device information, such as its hardware architecture and operating system, to the report server.
3) Frequently check new prospective target victims as well as the botnet's current status.
4) Issue an infect command in the loader server containing necessary details of the target device.
5) The loader server logs in the target device and instructs it to download and execute the malware, and the malware actively establishes a connection with the C&C server by resolving a hardcoded domain name in the binary.
6) It instructs all bots to commence an attack against a target server by issuing a command through the C&C server with corresponding parameters such as the attack type, duration, and target IP address.
7) The bot starts attacking the target server.

As Fig. 1 shows, a compromised device can be controlled by the botmaster to launch a DDoS attack, eavesdrop on other devices and leak sensitive data, scan and infect more devices, and conduct other malicious activities. Therefore, this article considers IoT devices as a potential source of attacks.

Our detection target primarily consists of two stages of threats.

1) *Command and Control:* It includes that a device is being compromised by brute force login, malicious payload execution, or being controlled by a connection with the remote C&C server. It corresponds to the first six phases of the IoT malware lifecycle above.

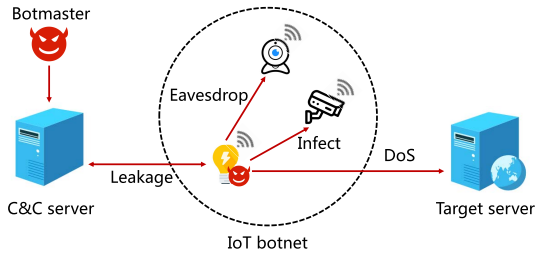


Fig. 1. Threat model: an IoT device is compromised and controlled by a botmaster to conduct malicious activities against other devices and entities.

2) *Launching Attack:* During this stage, the device is being used by the botmaster for various malicious activities, such as DDoS, scanning attacks, and data leakage. It mainly refers to the last phase of the lifecycle.

Additionally, certain assumptions are necessarily made.

1) A newly produced IoT device is initially benign and trusty, and no malware or backdoor is preinstalled.
2) All attacks from IoT bots leave traces in and above the IP layer traffic; accordingly, attacks like BLE spoofing or MAC spoofing are not considered.
3) IoT gateways, such as home routers and wireless access points (WAPs), have basic runtime environment to support custom programs for the detection service. As programmable routers, such as OpenWrt/LEDE [46] and Raspberry Pi Access Point (RaspAP) [47] are becoming popular, this assumption is realistic as long as the issue of the resource constraint can be mitigated.

IV. System Design

Our system architecture is an edge-assisted mechanism for the detection and mitigation of IoT-based network anomalies, especially the cyber threats originating from IoT botnets. The intuition behind this architecture is that we expect the perception of malicious activities as close to these susceptible end devices (IoT devices) as possible so that prompt actions can be taken before these devices are compromised and utilized to launch large-scale attacks. The framework can be executed on an edge device, such as an IoT gateway. By cooperating with a cloud server that stores the pretrained detectors, the framework provides a form of Detection-as-a-Service for the IoT devices connected to the edge. A primary difference between our system and other cloud–edge systems for IoT is that to increase the practicality and scalability of the framework on resource-constraint edges, instead of compressing models and vectors [9], [48] or using a collaborative mechanism that requires a new protocol for edge communications [37], [49], we utilize a widely used protocol by home routers and edge devices (SSDP) to implement a multiedge collaborative mechanism for the offloading of computing resources, which presents better scalability on existing edge devices.

A. IoT Networking

We first introduce a common networking paradigm for IoT connection—multigateway networking (Fig. 2). As a large proportion of current IoT devices require access to the Internet

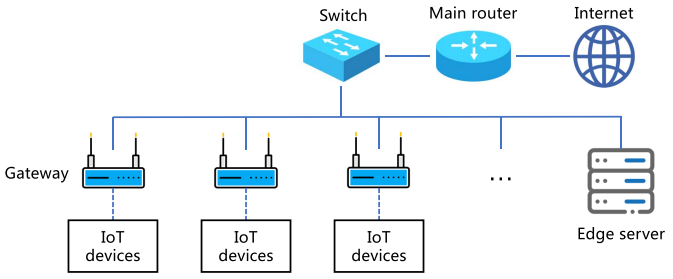


Fig. 2. Multigateway networking paradigm for IoT device connection.

for the connection with remote servers, a gateway is used as an intermediate node between end devices and the wide-area network. Specifically, this gateway can be a wireless router, a WAP, an integrated modem, or a switch in different scenarios. These gateways are usually connected to a switch to form a local area network and, therefore, are able to communicate with each other directly. Besides, some edge servers are also likely to exist in the network to provide certain services, such as data aggregation and analytics. To better illustrate this, we describe two typical scenarios that adopt this networking paradigm.

1) Home Networking: In a residence community, the real estate developer implements such a networking paradigm by providing each household with a modem integrated with a wireless router function. The real estate developer can also provide a complete set of smart home service to boost its scales and, thus, each integrated modem performs as a gateway for smart devices.

2) Enterprise Networking: Multiple WAP networking is commonly used in enterprises. A device connects to an appropriate WAP based on its position and the WAP's condition. In this scenario, a WAP plays a role of the gateway for IoT connections.

B. Multiedge Collaborative Design

Since the gateway serves as an intermediate node for all incoming and outgoing IoT traffic, it is an ideal deployment location for a network anomaly detection framework. However, the main challenge of this motive is the resource shortage of a gateway, especially for a sophisticated detection algorithm such as an ML-based detection algorithm. When the number of connected devices continuously increases, the gateway's resources will be gradually running out, which further leads to processing delay, impact on its basic forwarding function or even getting overwhelmed. There are other studies that use different strategies to mitigate this problem, such as incremental inference [9], quantization [48], pruning [50], distributed training [49], and distributed inference [51], which all achieve a great performance, but some of these techniques may not be suitable for our edge device, which is a home router or a WAP since a router (even an open-source router like OpenWrt) is manufactured with limited functions and not easy to extend for a consumer.

To improve the scalability, this article proposes a multiedge collaborative architecture for anomaly detection based on the multigateway networking paradigm (Fig. 3). The ADRIoT

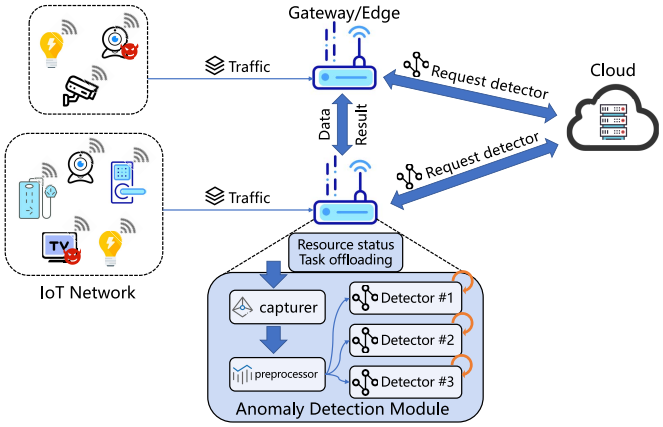


Fig. 3. Illustration of a multiedge collaborative design for ADRIoT.

framework is essentially an anomaly detection module on each gateway/edge and a detector container on the cloud. The cloud is responsible for the generation and storage of anomaly detectors for each type of IoT devices. When a new device connects to the gateway, the anomaly detection module identifies this device, requests the corresponding detector from the cloud, and executes it on the gateway. As the number of connected devices grows, a single gateway's burden also increases. For the quantitative evaluation, we design an algorithm to determine a gateway's resource status by its hardware usage. When this status surpasses a certain threshold, which means the gateway no longer has resources for more tasks, it will search other gateways/edges that also provide this anomaly detection service for task offloading. With a procedure of data exchange, the original gateway selects a proper co-worker and establishes a channel for further data transmission. We implement this procedure by utilizing SSDP, which is a widely supported service in almost every router and WAP. Since the raw traffic is preprocessed on the original gateway, and other peer gateways/edges are located in the same local network, the user's privacy will not be violated or leaked to any remote servers.

1) Resource Determination: A gateway determines its resource status that further decides whether it needs to offload the detection task or is able to take over the detection task from other gateways. Two parameters are taken into consideration: 1) CPU usage of the process denoted by u and 2) RAM usage of the process denoted by r where $0 \leq u < 1$ and $0 \leq r < 1$. It is easy to obtain these parameters by Linux commands like ps aux or top. For the sake of generalization, we consider gateways and edge servers providing no GPU resources. An upper bound threshold is set for these two resources to prevent overloading, denoted by u_0 and r_0. Inspired by DL methods, we use a ReLU activation function to reflect the signal that a resource is running out. A ReLU function is formulated as

$$g(x) = \max(0, x). \tag{1}$$

The resource status s is a binary indicator, which normally is equal to 0. When either of CPU usage and RAM usage surpasses the threshold, s is set to 1 that indicates the demand for task offloading. Hence, an indicator function with respect to the parameter u and r is used to determine if the resource

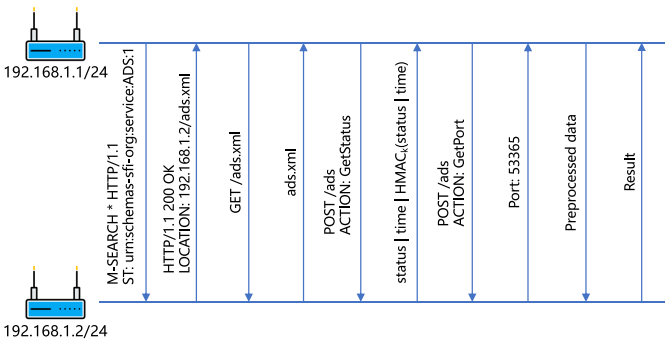


Fig. 4. Workflow of SSDP-based task offloading.

consumption is over the threshold (2). The resource status is calculated at regular intervals. If s is equal to 0, it means the gateway still has extra resources for more detection tasks; otherwise, the gateway will initiate the task offloading

$$s = \mathbf{1}\{u, r | g(u - u_0) + g(r - r_0) > 0\}. \tag{2}$$

2) SSDP-Based Task Offloading: We design and implement a simple and efficient task offloading mechanism in multi-gateway networking based on SSDP. It is practical and well compatible since most of current routers and access points support Universal Plug and Play (UPnP) protocol stack and SSDP is an important component in UPnP. As mentioned above, each gateway in such a networking paradigm is assumed able to communicate with each other and, thus, a multicasting SSDP message can be received by each gateway for service announcement and discovery. Fig. 4 illustrates the workflow of this process. Specifically, it consists of the following steps.

1) When a gateway G_1 finds its resource status $s_1 == 1$, it issues an M-SEARCH message searching for other devices that support the anomaly detection service.
2) Another gateway G_2 receives the message and responds by an HTTP message with a URL for the description file of the detection service.
3) G_1 gets the XML description file using the given URL.
4) The description file provides two remote call interfaces: a) GetStatus and b) GetPort. G_1 uses GetStatus to ask if G_2 has extra resources for task offloading.
5) G_2 responds by its resource status s_2, a timestamp, and a message authentication code.
6) If $s_2 == 0$, G_1 uses GetPort to request a new connection for data transmission; otherwise, G_1 requests another gateway's resource status according to the order of the responses in step 2.
7) G_2 randomly opens a new port and binds it with a new socket, and notifies G_1 of the port number.
8) G_1 initiates a new connection with G_2 for data transmission and returning the result.

In step 2, if multiple gateways supporting the service exist in the network, all of them will notify G_1 of their description file location. Then, G_1 will simply request the gateway according to the arrival order of the responses, for the fast reply generally suggests low delay and congestion between two gateways. In step 5, the reason for a message authentication code (typically by an HMAC function with a shared key) is to prevent the service hijacking, considering that an adversary may pretend to be a "gateway" by fabricating the same SSDP and HTTP response. Besides, a timestamp is added to protect the service from replay attacks. Moreover, a random new port opened for data transmission is to mitigate the risk of being scanned by attackers. As the task offloading is finished, the port will be closed immediately. Since SSDP itself offers limited security features, all these steps are designed for a more secure channel for the task offloading service.

3) Multiedge Collaboration: We utilize multiple edges' extra resources to release the burden of a single edge. As Fig. 2 illustrates, an edge can be a gateway or an edge server in such a network. It follows the resource determination and task offloading process as mentioned above. To further reduce the resource consumption on the gateway, the detector is compressed to a lightweight version on the cloud by quantization techniques which will be elaborated in the next section. Compared to a gateway, an edge server typically has sufficient computing resources to support more complicated processes. To maximize the utilization of the resources, the detector on these edges is retrained with very limited traffic data from local network, which fine tunes the model for better adaption to the local environment and user preference. It helps the reduction of false alarms considering that the normal behaviors from the local network can be different from the training data set on the cloud. It also explains why we do not aggregate the model on cloud: we want each detector to be *specific* to the local rather than to be *general* but unrefined.

To summarize, the workflow of the multiedge collaboration is as follows.

1) An edge requests the corresponding detectors from the cloud and execute them.
2) If the edge is a server, it retrains the detectors using limited local traffic data.
3) The edge checks its resource status at intervals by (2). If overloading, it starts the SSDP-based task offloading process.
4) If any anomalies are detected, the edge will generate an alarm and an anomaly report for further analysis.

V. Anomaly Detection Module

This section presents the anomaly detection module deployed on the edge to inspect the traffic of an IoT network. This module takes network traffic from IoT devices as input and outputs anomaly alarms if malicious activities are detected. The module mainly consists of three key components: 1) capturer; 2) preprocessor; and 3) detector.

A. Capturer

The capturer is the module entrance that captures traffic from the IoT network. Since the focus of this article is not device identification, we simply assume that the device type can be identified by its MAC address [52]. As the capturer mainly relies on an external tool Scapy for network packet sniffing and dissection, this article concentrates on the introduction of the other two components.

B. Preprocessor

The task of the preprocessor is to vectorize the network traffic so that an ML algorithm can digest. According to [34], botnet traffic usually exhibits distinctive sequential characteristics. For example, consecutive small packets might reveal a covert connection with a C&C server waiting for attack commands. Given this, we transform a set of consecutive network packets into a sequential data sample to unearth their sequential relationship. We also refer to some recent studies [24]–[26] for malware detection and traffic classification to use raw packet bytes as features. It borrows the idea from computer vision to automatically extract features from raw data by DL layers. As the neural network itself is capable of telling which fields and features are useful or useless for the classification, the reliance on handcrafted feature engineering from expert knowledge can be largely reduced.

The workflow of the preprocessor follows three steps.

1) Manipulation: During this step, some fields in packets related to local network configuration will be removed or modified to avoid overfitting. Specifically, the data link layer is removed and IP addresses are masked by zero. Furthermore, since an ML model requires a fixed length of the input, each packet will be zero-padded to the size of the Maximum Transmission Unit (MTU, typically, 1500 bytes). For example, an HTTP packet is manipulated to the following form:

$$\underbrace{\text{IP}(\text{src} = 0.0.0.0, \text{dst} = 0.0.0.0)|\text{TCP}|\text{HTTP}|0000\cdots}_{\text{MTU}}$$

2) Normalization: To obtain a better convergence, each byte of a packet is normalized by being divided by 255, the maximum value of a byte, turning each byte into a value between 0 and 1. A packet is then transformed into a 1500-D feature vector

$$\boldsymbol{p} = [b_1, b_2, b_3, \ldots, b_d]^T, b \in [0, 1], d = 1500. \tag{3}$$

Such an approach to packet normalization is shown by other work [24], [25] to be simple yet effective for the reduction of feature engineering.

3) Aggregation: Considering that different connections carry out different activities, we use a time-series sliding window to cache the packets from the same connection to form a sequential data sample. The determination of sliding window length m is done according to the average length of five-tuple flows from a device. Given device types, IoT devices are categorized into *streaming* device and *controlling* device: a *streaming* device, such as a TV, a camera and a speaker that continuously transfer streaming data usually has a longer average flow length; a *controlling* device, such as a Wi-Fi bulb and plug often uses short flows for command reception and response. For a sequential learning model, an empirical rule is that the longer sequence length, the more information it contains, whereas the more difficult the training convergence will be. To balance the information acquisition and training difficulty, we empirically determine m as follows:

$$m_{\text{streaming}} = \frac{1}{3}\frac{\sum_{i=1}^{k} \text{len}_{f_i}}{k}, m_{\text{controlling}} = \frac{2}{3}\frac{\sum_{i=1}^{k} \text{len}_{f_i}}{k} \tag{4}$$

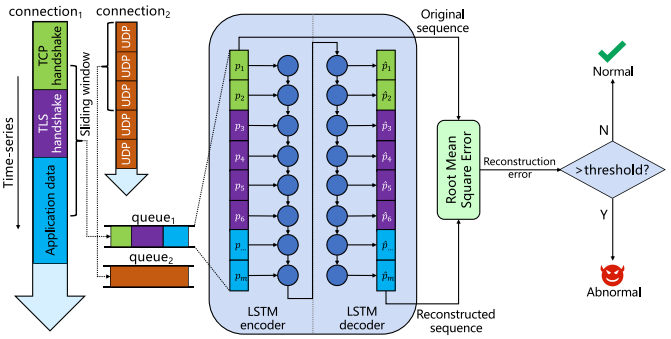


Fig. 5. Overview of an LSTM AE detector; for example, packets from a TCP connection and a UDP connection are cached in two queues, respectively, to form sequential input samples for the detector using (5).

where f_i is a sampled five-tuple flow, k is the number of flows collected from a device. Therefore, each type of device will have a neither too long nor too short sequence length which still contains sufficient information of their activities. Then, for each connection c_i, an m-length first-in-first-out queue Q_{c_i} caches the consecutive packets and forms a sequential data sample. If Q_{c_i} is filled up, it constructs a time-series sequential data sample X as (5) describes. When a new packet from c_i arrives, Q_{c_i} pops out the first-in packet and pushes in the new packet to compose the next sequential data sample.

$$Q_{c_i} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m] = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ b_{d1} & b_{d2} & \cdots & b_{dm} \end{bmatrix}$$
$$X \leftarrow Q_{c_i}. \tag{5}$$

C. Detector

In the IoT attack detection scenario, attack traffic is very rare compared to normal device traffic, and zero-day attacks are always emerging so that it is almost impossible to maintain a complete data set for all attacks. Therefore, an unsupervised learning detector is more suitable for our goal. To the best of our knowledge, it is also the first attempt to use the raw packet bytes as features in a manner of unsupervised learning for network attack detection.

The detector is essentially a profile model for each type of device as Fig. 5 illustrates. It detects anomalies by the perception of deviation from the normal behavior. The detectors are pretrained on the cloud. When a device connects to the edge, the corresponding detector will be retrieved from the cloud and deployed on the edge.

1) LSTM Autoencoder: The LSTM autoencoder (LSTM AE) is a sequence-to-sequence unsupervised model. It is composed of three layers—1) an LSTM encoder; 2) an LSTM decoder; and 3) a fully connected layer. The encoder first compresses the input sequence into a lower dimensional hidden representation. For an input sequence $X = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m]$, each $\boldsymbol{p}_t$ follows the transition below to calculate the intermediate parameters:

$$\begin{cases} f_t = \sigma\left(W_f \cdot [h_{t-1}, \boldsymbol{p}_t] + b_f\right) & \text{(6a)} \\ i_t = \sigma\left(W_i \cdot [h_{t-1}, \boldsymbol{p}_t] + b_i\right) & \text{(6b)} \\ l_t = \tanh\left(W_l \cdot [h_{t-1}, \boldsymbol{p}_t] + b_l\right) & \text{(6c)} \\ o_t = \sigma\left(W_o \cdot [h_{t-1}, \boldsymbol{p}_t] + b_o\right) & \text{(6d)} \end{cases}$$

where W and b are the trainable parameters (weight and bias) in an LSTM, and then the hidden representation at timestamp t can be calculated using the intermediate parameters

$$h_t = o_t \cdot \tanh(f_t \cdot c_{t-1} + i_t \cdot l_t). \quad (7)$$

Next, the decoder uses the same number of neurons to decompress the hidden representation back to a higher-dimensional representation. Finally, a fully connected layer maps the representation to a sequence with the same length and dimension as the input sequence X, denoted by $\hat{X} = [\hat{\boldsymbol{p}_1}, \hat{\boldsymbol{p}_2}, \ldots, \hat{\boldsymbol{p}_m}]$. The training target of an LSTM AE is to minimize the difference between X and $\hat{X}$. It is measured by the root mean square error (RMSE) defined as follows:

$$\text{RMSE}(X, \hat{X}) = \sqrt{\frac{\sum_{i=1}^{m} (\boldsymbol{p}_i - \hat{\boldsymbol{p}}_i)^2}{m}}, p_i \in X, \hat{p}_i \in \hat{X}. \quad (8)$$

The RMSE can be interpreted as the reconstruction error between two sequences. During the inference phase, the reconstruction error obtained from an abnormal sequence will not match the value range produced by normal sequences, therefore anomalies can be detected. We use the maximum reconstruction error during the training multiplied by a cutoff coefficient ϵ as the threshold. Any data sample that generates a reconstruction error over this threshold will be considered anomalous.

2) Lightweight Detector: When the detector is requested by a gateway, considering the limited resources, the cloud will generate a lightweight version of the requested detector. It uses the post-training quantization technique to convert the 32-bit float weights in the LSTM AE to 8-bit integers, which reduces the size of the LSTM AE by 75%. The post-training quantization technique can also accelerate the inference speed to twice the previous. Although this process might cause a degree of accuracy loss, our evaluation shows that our detector basically exhibits very little decline in anomaly detection accuracy.

VI. Evaluation

This section presents the empirical evaluation of our framework ARDIoT. Overall, our evaluation covers the following aspects.

1) *Reconstruction:* We demonstrate the reconstruction of packet sequences and compare the separability between normal and attack traffic using LSTM AE and three baseline unsupervised methods.
2) *Detection:* We show the detection performance on various attacks based on the reconstruction error and the threshold determined during the training process.
3) *Resource:* We show the resource consumption under the multiedge collaborative mechanism and present the advantage.
4) *Discussion:* We discuss a possible attack against ADRIoT and mitigation policy derived from ADRIoT.

TABLE I
Description of the IoT Traffic Data Set

No.	Device	Size (US)	Size (UK)	Description
1	AppleTV	44.8MB	66.9MB	Smart TV
2	Nest thermostat	16.2MB	33.3MB	Thermostat
3	T-WeMo plug	55.7MB	47.8MB	Wi-Fi plug
4	TP-Link bulb	36.4MB	12.6MB	Wi-Fi bulb
5	TP-Link plug	15.1MB	7.4MB	Wi-Fi plug
6	Zmodo doorbell	147.9MB	N/A	Smart camera
7	Echo Plus	56.5MB	35.6MB	Speaker
8	Roku TV	42.4MB	26.6MB	Smart TV
9	Sengled Hub	3.2MB	1.6MB	IoT hub
10	Echo Spot	45.6MB	87.6MB	Speaker

TABLE II
Description of the Attack Data Set

Category	Name	Size	Description
C&C	Mirai [9]	62.9MB	IoT botnet by Telnet
	XBash [53]	89.5MB	coin miner malware
	Bashlite [54]	53.1MB	IoT botnet for DDoS
DDoS	TCP SYN flood [55]	225.8MB	transport layer
	UDP flood [55]	212.5MB	transport layer
	HTTP flood [55]	243.7MB	application layer
Scan	Service scan [55]	243.4MB	Nmap, hping3
	OS scan [55]	243MB	Nmap, Xprobe2
Leakage	Data exfiltration [55]	246.3MB	brute force SSH
	Keylogging [55]	245.1MB	brute force SSH

A. Data Set

Our IoT traffic data set is from the research team of Northeastern University and Imperial College London [30].[1] The data set consists of network traffic collected from 26 popular smart home IoT devices. This data set is very suitable for our evaluation because: 1) it keeps the raw PCAP files with complete payload and 2) same types of IoT devices were conducted in two laboratories, one located in the U.S. and the other located in the U.K. It is helpful for the evaluation of whether a pretrained *general* detector in one location can be fine tuned into a *specific* detector in another location with a different network environment and user preference. We utilize the traffic data from ten different types of devices for our experiment, including smart camera, speaker, home automation, smart TV, sensor, etc. The description of the data set is shown in Table I. For the rest of the paper, we will use the **No.** in Table I to refer to each device.

Table II summarizes the attack data sets. They are collected from published studies or public access on the Internet. We categorize them into four types of common IoT-based attacks: 1) C&C; 2) DDoS; 3) scanning; and 4) data leakage. For each experiment, the data set from each attack category is balanced by oversampling to make sure the detection result is not biased to certain attack categories. Afterward, the legitimate data set and attack data set are merged by a ratio of 1 : 1 by oversampling again to balance the positive samples and negative samples for evaluation.

[1]This data set can be accessed by following the data sharing procedure on https://moniotrlab.ccis.neu.edu/imc19/.

B. Configuration

A laptop is utilized to simulate an IoT network by replaying the above traffic data set using Tcpreplay to the edge. We use multiple Raspberry Pi boards (ARM Cortex-A72, 1.50 GHz) to simulate a multiedge network with multiple low-power IoT gateways and a CPU server (Intel Xeon CPU E5-2630 v2, 2.60 GHz) to simulate an edge server. Another server (Intel Xeon CPU E5-2630 v2, 2.60 GHz) simulates the cloud for the generation and distribution of detectors to the edge.

C. Metrics

For an anomaly detection system, the common metrics include true positive (TP), false negative (FN), true negative (TN) and false positive (FP). We can further calculate true positive rate TPR $=$ (TP/TP $+$ FN) and false positive rate FPR $=$ (FP/FP $+$ TN).

Moreover, to measure the separability of the reconstruction error between normal and attack traffic, we use the receiver operating characteristics curve (ROC) along with area under curve (AUC) and equal error rate (EER). The ROC curve plots TPR against FPR by setting each reconstruction error from normal and attack data samples as threshold. AUC is the area under the ROC curve and x-axis. The closer the AUC is to 1, the better capability of binary classification. EER is the x-coordinate of the point on the ROC curve that indicates an equal misclassification probability of a positive sample and a negative sample. It can be determined by drawing a line between (0, 1) and (1, 0) (i.e., $y = 1 - x$) that intersects with the ROC curve. A lower EER suggests that the algorithm has a lower probability of making mistakes. An advantage of these metrics is their immunity to the imbalance of positive and negative data. Though we do balance our data set in the experiment, the attack (positive) traffic data in the real world are much more rare compared to normal (negative) traffic data, which highlights the choice of these metrics for our evaluation.

We also use *precision*, *recall*, and *F1* score to measure the detection performance. These three metrics can be calculated as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{10}$$

D. Reconstruction

1) Sequence Length Determination: Fig. 6 shows the descent of the detector's training loss with three different sequence lengths. It can be observed that all curves gradually drop and nearly converge to the same level, which means the model with different sequence length is successfully learning from raw traffic. The difference is that longer sequence converges relatively slower. We count their average length of five-tuple upstream flows as Fig. 7 shows. Given their functions, we categorize smart TVs, speakers, and cameras to *streaming* devices, and plugs, bulbs, thermostats, and IoT hubs to *controlling* devices. Apparently, *streaming* devices usually have longer flow length than *controlling* devices. The sliding window length for each device is then determined by

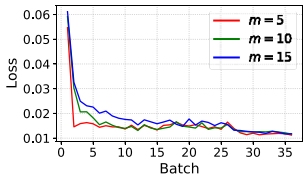


Fig. 6. Training loss.

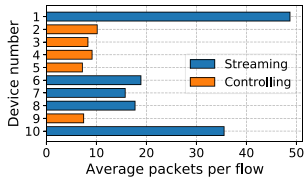
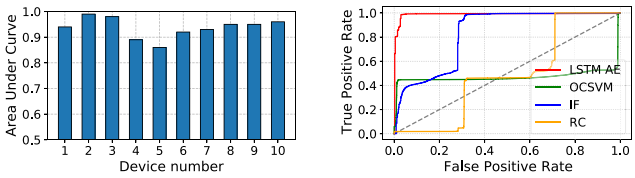


Fig. 7. Upstream packets per flow.



(a) (b)

Fig. 8. AUC from ten devices using LSTM AE and ROC curves plotted by LSTM AE and three baseline algorithms. (a) AUC. (b) ROC.

(4), which balances the information acquisition and training difficulty.

2) Distinguishability by LSTM Autoencoder: We first illustrate the AUC of reconstruction errors from normal traffic of each device and all attack traffic as Fig. 8(a) depicts. We can find that the AUC is over 0.9 for most of devices. To compare the performance with other methods, we select three unsupervised outlier detection algorithms as baselines: 1) one-class SVM (OCSVM); 2) isolation forest (IF); and 3) robust covariance (RC). They are implemented by scikit-learn, a Python ML module. We use exactly the same training and testing data set and input representation (raw packet bytes) for the LSTM AE and baseline methods. The only difference is that these methods use single packets as the detection unit rather than packet sequences.

Fig. 8(b) shows the ROC curve comparison. The LSTM AE can achieve nearly 100% true positive rate with a false positive rate below 3%, which significantly outperforms other baseline methods. We ascribe this encouraging result to two reasons: 1) the autoencoder architecture uncovers highly useful hidden representation in the middle layer and 2) LSTM layers capture the sequential relationship between time-series consecutive packets.

We are also curious about the comparison across different attack categories. Fig. 9(a) and (b) shows the AUC and EER metrics among four categories of attacks. The LSTM AE outperforms other methods by both AUC and EER metrics in all categories. Among the baseline methods, IF exhibits the best performance while RC shows extremely poor capability to distinguish normal and attack traffic. Among these attacks, the LSTM AE is best at distinguishing C&C and DDoS attacks,

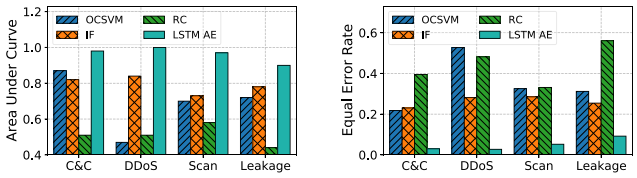


Fig. 9. Metrics comparison between LSTM AE, OCSVM, IF, and RC. (a) AUC comparison. (b) EER comparison.

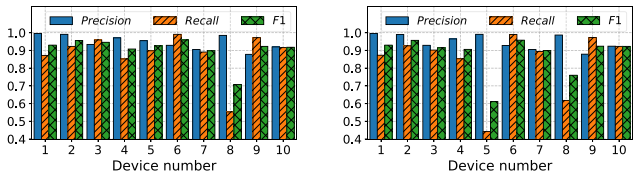


Fig. 10. Detection performance on ten different IoT devices. (a) Original detectors. (b) Lightweight detectors.

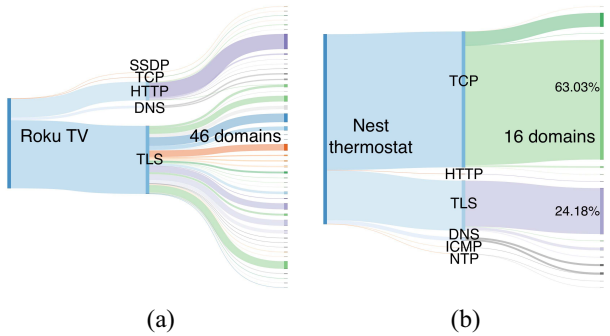


Fig. 11. Sankey diagrams of device traffic, from left to right representing device name, protocol, and connected domains. (a) Roku TV. (b) Nest thermostat.

and slightly weak for data leakage. It is probably because the traffic of data leakage more or less resembles the normal traffic, especially for a *streaming* device of which the main function is to transmit streaming data.

E. Detection

1) Detection Capability: The threshold of reconstruction error is determined by the maximum reconstruction error during the training multiplied by a cutoff coefficient ϵ. We empirically set $\epsilon = 0.9$. Fig. 10(a) shows the detection performance with respect to precision, recall and F1 score on ten devices. Our detector presents satisfactory detection performance with respect to all three metrics on most of the devices. One exception is the Roku TV (No.8, Recall = 0.553). The poor recall suggests that its detector has a high false negative rate. We explain this exception by the comparison of behavior complexity between Roku TV and Nest thermostat. As Fig. 11 shows, the traffic of Roku TV is far more diverse than the traffic of Nest thermostat. Although the domain names/IP addresses are masked during the preprocessing, the traffic from different domains exhibits highly diverse patterns as they are dedicated to different services and purposes. For example, the connection with "appboot.geo.netflix.com" is used for positioning when Netflix app is used, which uses the plain-text HTTP POST messages with 1500 bytes per packet; another connection with "api.rokutime.com" is used for time synchronization when the TV is open, which in contrast follows a standard TLS handshake procedure with approximately 250-byte cipher-text messages. It can be seen that their traffic are far from conforming a uniform pattern. In fact, the Roku TV has the most connections among the ten devices and, thus, presents the most complex activities. As a result, its detector tends to profile the "normality" to a relatively large range so that substantial attack data samples are also encircled in the boundary and, thus, the false negative rate is increased.

TABLE III
COMPARISON OF DETECTION RATE ON C&C BETWEEN SUPERVISED AND UNSUPERVISED PROPOSALS

Proposal	Label?	Zero-day?	Known	Unknown
[24]	Yes	No	99.9% (99.8%)	67.1% (0.55%)
ADRIoT	No	Yes	90.3%	70.0%

To highlight the advantage of unsupervised methods, we compare ADRIoT with [24], which uses similar raw traffic data representation as ours in a supervised manner. We train this model using the same normal data and four categories of attack data. The only difference is that we only include Mirai and Bashlite during the training of C&C category and leave XBash as an zero-day attack for testing. As Table III shows, the supervised model's detection rate on known C&C attacks reaches an extremely high level (99.9% correct classification into any attack category, 99.8% correct classification into C&C category). However, its detection rate on the unknown C&C by XBash is lower than our detector (67.1% correct classification into any attack category, 0.55% correct classification into C&C category). In fact, since XBash uses HTTP, which is a commonly used protocol in IoT traffic, for C&C communications rather than Telnet as Mirai and Bashlite, the supervised model identifies most of its traffic as normal traffic or data theft category. For this reason, the detection rate of our detector on XBash is also lower than the detection rate of Mirai and Bashlite, but it still shows its effectiveness for the alarm of unknown attacks.

2) Accuracy Loss of Lightweight Detector: We evaluate the lightweight detectors compressed by the post-quantization technique with the same data set in Fig. 10(a) and illustrate Fig. 10(b). The decline of precision, recall and F1 score are minor and, thus, the performance degradation of the lightweight detectors can be considered insignificant. Still, the sharp decline of recall on the TP-Link plug (No.5, from 0.898 to 0.442) suggests the possible risk of a tradeoff between lightweight deployment and the detection performance.

3) Benefit of Fine-Tuning: We use the data set from the Echo Spot to evaluate its benefits. Echo Spot is a multiple control device. Users can either directly interact with it by talking to it, or control it remotely by an Android app. Apparently, given different user preferences, different control methods will

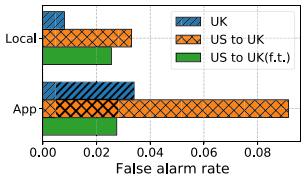


Fig. 12. False alarm rate.

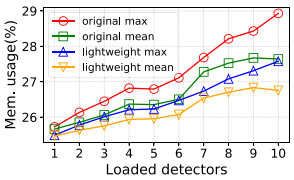


Fig. 13. Memory usage.

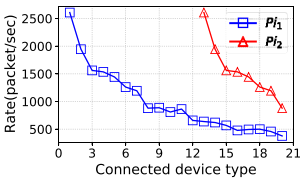


Fig. 14. Processing speed.

generate different network traffic. We compare the false alarm rate of the Echo Spot's detector under three circumstances: 1) a *general* detector trained on the U.K. data set and evaluated on the U.K. data set; 2) a *general* detector trained on the U.S. data set and evaluated on the U.K. data set; and 3) a *specific* detector trained on the U.S. data set, fine tuned on only 10% of the U.K. data set with one of the control methods and evaluated on the U.K. data set. The result is shown in Fig. 12. When the *general* detector is transferred from the U.S. environment to the U.K. environment, the false alarm rate is significantly increased either by local control or by remote app control. Meanwhile, with fine-tuning by limited local traffic data, the false alarm rate of the *specific* detector is substantially reduced to almost the same level as the first circumstance.

F. Resource

1) Resource Saving by Lightweight Detector: The lightweight detectors are deployed on the Raspberry Pi—the size of the original detector is 2.0 MB and it is compressed to 512 KB after the post-training quantization. Fig. 13 presents the memory usage of the Raspberry Pi with the increase of loaded detectors. Compared to the original detector, the lightweight detector consumes less memory; among the lightweight detectors, the one that consumes the maximum memory still outperforms the average memory consumption of the original detectors.

2) Resource Saving by Multiedge Collaboration: We first use two Raspberry Pi boards to simulate two gateways. They serve as WAPs, and their Ethernet interfaces are connected to a main router. We increasingly send different device's traffic to one Pi (P_1) and keep the other Pi idle (P_2). As Fig. 14 shows, when the connected device types reach 13, the processing speed of P_1 drops by over 70%. Then, it starts to offload

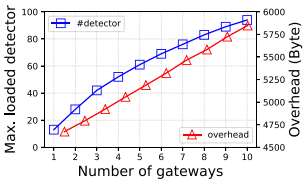


Fig. 15. Maximum loaded detectors and bandwidth overhead.

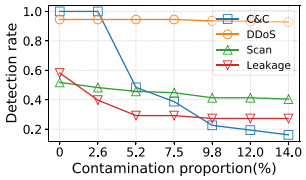


Fig. 16. Poisoning attack.

the task to P_2 and, thus, the descent of processing speed on P_1 is apparently decelerated. As P_1 only needs to handle the preprocessing procedure for new connected devices, its processing speed only drops by about 5.6% after the task offloading. With the multiedge collaborative mechanism, this processing speed maintains an acceptable level, especially considering that ADRIoT is designed to be an asynchronous detection system rather than a real-time system.

Then, we add more gateways (Raspberry Pi) to the network. All traffic is still sent only to one gateway, and others are used for task offloading. The maximum number of loaded detectors for a network is counted when every gateway's resource status is equal to 1, which means no gateway is able to take more detection tasks. As the blue line in Fig. 15 illustrates, more gateways engaging in the collaboration give the network better resource utilization to deploy more detectors. The reason that the line is not perfectly linear is mainly because of the increasing overhead caused by the multicasting and peer communications in the SSDP service (Fig. 4). We measure this bandwidth overhead on the main router where all the traffic between gateways goes through and draw the red line in Fig. 15. It suggests that the overhead caused by the multiedge collaborative mechanism is trivial compared to the overall throughput of the router.

G. Discussion

1) Adversarial Attack on ADRIoT: The adversarial attack aims to deteriorate the performance of detectors by making it misclassify certain tasks. It can be classified into *poisoning* attacks that contaminate the training data set or labels during the training phase, and *evasion* attacks that manipulate the input data during the executing phase to deceive the detector. Although we make the assumption in the threat model that the manufacturer and device is initially benign, however, some malware in reality compromises the manufacturer's software update server to spread it to the end device [56]. Therefore, poisoning attacks on ADRIoT are possible. To assess its impact, we contaminate the training data set by four categories of attack traffic and demonstrate the detection rate in Fig. 16. As the contamination proportion in the training

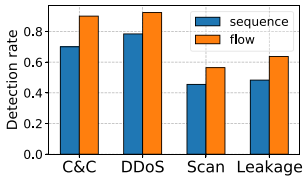


Fig. 17. Sequence versus flow.

data set increases, the detection rate of DDoS attacks is almost unaffected; the impact on the detection of scanning attacks and data leakage exists but is not very significant. However, the detection rate of C&C attacks sharply drops by 90% when the contamination proportion reaches 14%. It shows the possibility of certain attacks to avoid detection by poisoning the detector. This article does not include the evaluation on evasion attacks, which will be our future work.

2) Mitigation Policy Generation: Although this article mainly focuses on the anomaly detection, we also propose a simple mitigation policy based on the detection result. For a certain flow, it is labeled as *suspicious* when packet sequences in the flow are detected as anomalies. The network administrator can configure two parameters: 1) a suspicion threshold t and 2) a time window w. When the number of labeled *suspicious* flows from one device exceeds t during the time window w, the administrator can infer that this device is compromised and needs to be quarantined. In Fig. 17, we set $t = 1$ and $w = 1$ s, and it shows the detection rate of the suspicious flows is even higher than the detection rate based on sequences. It is reasonable since not all packet sequences in an attack flow carry malicious activities whereas all labeled as "attack" in the data set.

VII. Conclusion

This article proposed a novel anomaly detection framework called ADRIoT. We adopted an edge-assisted architecture in which the anomaly detection module is running on the edge so that IoT-based attacks can be detected in a close position to the data source promptly. To resolve the challenge of resource limitation on edge, we designed a multiedge collaborative mechanism that integrates resources in a local network to support the service. The anomaly detection module was constructed in an unsupervised manner to prevent increasing zero-day attacks on IoT networks. The evaluation showed that ADRIoT can detect a wide range of IoT-based attacks effectively and efficiently.

Acknowledgment

The authors thank the Mon(IoT)r Research Group at Northeastern University for kindly sharing their complete IoT data set in the format of raw PCAP files.

References

[1] "Ericsson Mobility Report: Internet of Things Forecast." 2018. [Online]. Available: https://www.ericsson.com/en/mobility-report/reports

[2] S. Heble, A. Kumar, K. V. D. Prasad, S. Samirana, P. Rajalakshmi, and U. B. Desai, "A low power IoT network for smart agriculture," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, 2018, pp. 609–614.

[3] A. Bujari, M. Furini, F. Mandreoli, R. Martoglia, M. Montangero, and D. Ronzani, "Standards, security and business models: Key challenges for the IoT scenario," *Mobile Netw. Appl.*, vol. 23, no. 1, pp. 147–154, 2018.

[4] "Internet of Things Security and Privacy Recommendations." BITAG. 2016. [Online]. Available: https://www.bitag.org/documents/BITAG_Report__Internet_of_Things_(IoT)_Security_and_Privacy_Recommendations.pdf

[5] "IoT Default Password Lookup." [Online]. Available: https://www.defpass.com/ (Accessed: Oct. 2020).

[6] M. Antonakakis *et al.*, "Understanding the Mirai botnet," in *Proc. 26th USENIX Security Symp. (USENIX Security)*, 2017, pp. 1093–1110.

[7] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, Jul. 2017.

[8] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Netw.*, vol. 8, no. 3, pp. 26–41, May/Jun. 1994.

[9] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. 25th Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2018, pp. 1–15.

[10] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," in *Proc. IEEE Security Privacy Workshops (SPW)*, 2018, pp. 29–35.

[11] M. Thottan and C. Ji, "Anomaly detection in IP networks," *IEEE Trans. Signal Process.*, vol. 51, no. 8, pp. 2191–2204, Aug. 2003.

[12] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *J. Netw. Comput. Appl.*, vol. 60, pp. 19–31, Jan. 2016.

[13] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 303–336, 1st Quart., 2014.

[14] N. Ye and Q. Chen, "An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems," *Qual. Rel. Eng. Int.*, vol. 17, no. 2, pp. 105–112, 2001.

[15] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proc. 7th Int. Symp. Recent Adv. Intrusion Detect. (RAID)*, 2004, pp. 203–222.

[16] T. Abbes, A. Bouhoula, and M. Rusinowitch, "Efficient decision tree for protocol analysis in intrusion detection," *Int. J. Security Netw.*, vol. 5, no. 4, pp. 220–235, 2010.

[17] L. Khan, M. Awad, and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering," *VLDB J.*, vol. 16, no. 4, pp. 507–521, 2006.

[18] C. Wagner, J. François, R. State, and T. Engel, "Machine learning approach for IP-flow record anomaly detection," in *Proc. NETWORKING*, 2011, pp. 28–39.

[19] S. Vahid and M. Ahmadzadeh, "KCMC: A hybrid learning approach for network intrusion detection using K-means clustering and multiple classifiers," *Int. J. Comput. Appl.*, vol. 124, no. 9, pp. 18–23, 2015.

[20] L. Kuang, "DNIDS: A dependable network intrusion detection system using the CSI-KNN algorithm," M.S. thesis, School Comput., Queen's Univ., Kingston, ON, Canada, 2007.

[21] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," *Int. J. Security Appl.*, vol. 9, no. 5, pp. 205–216, 2015.

[22] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, 2016, pp. 258–263.

[23] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *Proc. Int. Conf. Platform Technol. Service (PlatCon)*, 2016, pp. 1–5.

[24] G. Marín, P. Casas, and G. Capdehourat, "Deep in the dark-deep learning-based malware traffic detection without expert knowledge," in *Proc. IEEE Security Privacy Workshops (SPW)*, 2019, pp. 36–42.

[25] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Comput.*, vol. 24, no. 3, pp. 1999–2012, 2020.

[26] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, "Byte segment neural network for network traffic classification," in *Proc. IEEE/ACM 26th Int. Symp. Qual. Service (IWQoS)*, 2018, pp. 1–10.

[27] R. Tang *et al.*, "ZeroWall: Detecting zero-day Web attacks through encoder-decoder recurrent neural networks," in *Proc. 39th IEEE Conf. Comput. Commun. (INFOCOM)*, 2020, pp. 2479–2488.

[28] A. Rachedi and A. Benslimane, "Multi-objective optimization for security and qos adaptation in wireless sensor networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2016, pp. 1–7.

[29] N. Haddadou, A. Rachedi, and Y. Ghamri-Doudane, "A job market signaling scheme for incentive and trust management in vehicular ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 64, no. 8, pp. 3657–3674, Aug. 2015.
[30] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. Internet Meas. Conf.*, 2019, pp. 267–279.
[31] O. Alrawi, C. Lever, M. Antonakakisbi, and F. Monrose, "SoK: Security evaluation of home-based IoT deployments," in *Proc. IEEE Symp. Security Privacy (SP)*, 2019, pp. 1362–1380.
[32] A. A. Khan, M. H. Rehmani, and A. Rachedi, "Cognitive-radio-based Internet of Things: Applications, architectures, spectrum related functionalities, and future research directions," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 17–25, Jun. 2017.
[33] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for Internet of Things (IoT) security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1646–1685, 3rd Quart., 2020.
[34] G. Kirubavathi and R. Anitha, "Botnet detection via mining of traffic flow characteristics," *Comput. Electr. Eng.*, vol. 50, pp. 91–101, Feb. 2016.
[35] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the Internet of Things using deep learning approaches," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2018, pp. 1–8.
[36] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS attack via deep learning," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, 2017, pp. 1–8.
[37] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for Internet of Things," *Future Gener. Comput. Syst.*, vol. 82, pp. 761–768, May 2018.
[38] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.
[39] K. Bhardwaj, J. C. Miranda, and A. Gavrilovska, "Towards IoT-DDoS prevention using edge computing," in *Proc. USENIX Workshop Hot Top. Edge Comput. (HotEdge)*, 2018, pp. 1–7.
[40] I. M. Ozcelik, N. Chalabianloo, and G. Gür, "Software-defined edge defense against IoT-based DDoS," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, 2017, pp. 308–313.
[41] J. Schneible and A. Lu, "Anomaly detection on the edge," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, 2017, pp. 678–682.
[42] B. Cetin, "Wireless network intrusion detection and analysis using federated learning," M.S. thesis, Dept. Comput. Sci. Inf. Syst., Youngstown State Univ., Youngstown, OH, USA, 2020.
[43] Y. Zhao, J. Chen, Q. Guo, J. Teng, and D. Wu, "Network anomaly detection using federated learning and transfer learning," in *Security and Privacy in Digital Economy*, S. Yu, P. Mueller, and J. Qian, Eds. Quzhou, China: Springer, 2020, pp. 219–231.
[44] O. M. Ezeme, Q. H. Mahmoud, and A. Azim, "A deep learning approach to distributed anomaly detection for edge computing," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. ICMLA*, 2019, pp. 992–999.
[45] D. Bendouda, A. Rachedi, and H. Haffaf, "Programmable architecture based on software defined network for Internet of Things: Connected dominated sets approach," *Future Gener. Comput. Syst.*, vol. 80, pp. 188–197, Mar. 2018.
[46] "OpenWrt." [Online]. Available: https://openwrt.org (Accessed: May 2021).
[47] "RaspAP." [Online]. Available: https://raspap.com (Accessed: May 2021).
[48] M. Salman, D. Husna, S. G. Apriliani, and J. G. Pinem, "Anomaly based detection analysis for intrusion detection system using big data technique with learning vector quantization (LVQ) and principal component analysis (PCA)," in *Proc. Int. Conf. Artif. Intell. Virtual Real.*, 2018, pp. 20–23.
[49] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy, "DEFT: A distributed IoT fingerprinting technique," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 940–952, Feb. 2019.
[50] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 1398–1406.
[51] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "DeepSlicing: Collaborative and adaptive CNN inference with low latency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2175–2187, Sep. 2021.
[52] A. Singh *et al.*, "HANZO: Collaborative network defense for connected things," in *Proc. Principles Syst. Appl. IP Telecommun. (IPTComm)*, 2018, pp. 1–8.
[53] "Stratosphere Laboratory Datasets." [Online]. Available: https://www.stratosphereips.org/datasets-overview (Accessed: Oct. 2020).
[54] V. Bezerra, V. G. T. da Costa, R. A. Martins, S. Barbon, R. S. Miani, and B. B. Zarpelão, "Providing IoT host-based datasets for intrusion detection research," in *Proc. Simpósio Brasileiro Segurança Informação Sistemas Computacionais (SBSeg)*, 2018, pp. 1–14.
[55] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Gener. Comput. Syst.*, vol. 100, pp. 779–796, Nov. 2019.
[56] "Petya." [Online]. Available: https://en.wikipedia.org/wiki/Petya_(malware) (Accessed: Oct. 2020).

Ruoyu Li received the B.S. degree in cybersecurity from the Huazhong University of Science and Technology, Wuhan, China, in 2017, and the M.S. degree in computer science from Columbia University, New York, NY, USA, in 2019. He is currently pursuing the Ph.D. degree with the Tsinghua–Berkeley Shenzhen Institute, Tsinghua University, Beijing, China.

He is a Research Intern with Peng Cheng Laboratory, Shenzhen, China. His research interests mainly include Internet of Things security, intrusion/anomaly detection system, network security, and AI for computer network.

Qing Li (Member, IEEE) received the B.S. degree in computer science and technology from Dalian University of Technology, Dalian, China, in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013.

He is currently a Research Associate Professor with Peng Cheng Laboratory, Shenzhen, China. His research interests include reliable and scalable routing of the Internet, in-network caching/computing, intelligent self-running network, and edge computing.

Jianer Zhou received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2016.

He is currently a Research Assistant Professor with the Southern University of Science and Technology, Shenzhen, China, and also with Peng Cheng Laboratory, Shenzhen. His research interests lie in network performance, future Internet architecture, and Internet measurement.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively.

He is currently a Full Professor with Tsinghua Shenzhen International Graduate School, and also with Peng Cheng Laboratory, Shenzhen, China. He mainly focuses on the future Internet architecture, software defined networking, network function virtualization, information-centric networks, network security, cloud computing, edge computing, content delivery, and AI for networks.