






# IoTEnsemble: Detection of Botnet Attacks on Internet of Things

Ruoyu Li<sup>1,2</sup> , Qing Li<sup>2</sup> , Yucheng Huang<sup>1,2</sup>, Wenbin Zhang<sup>3</sup>,  
Peican Zhu<sup>4</sup> , and Yong Jiang<sup>1,2</sup>

<sup>1</sup> Shenzhen International Graduate School, Tsinghua University, Shenzhen, China  
{liry19, huangyc20}@mails.tsinghua.edu.cn  
jiangy@sz.tsinghua.edu.cn

<sup>2</sup> Peng Cheng Laboratory, Shenzhen, China  
liq@pcl.ac.cn

<sup>3</sup> Xidian University, Xi'an, China  
wbzhang\_2@stu.xidian.edu.cn

<sup>4</sup> Northwestern Polytechnical University, Xi'an, China  
ericcan@nwpu.edu.cn

**Abstract.** As the Internet of Things (IoT) plays an increasingly important role in real life, the concern about IoT malware and botnet attacks is considerably growing. Meanwhile, with new techniques such as edge computing and artificial intelligence applied to IoT networks, these devices nowadays become more functional than ever before, which challenges many existing network anomaly detection systems due to the lack of generalization ability to profile diverse activities.

To address it, this paper proposes IoTEnsemble, an ensemble network anomaly detection framework. We propose a tree-based activity clustering method that aggregates network flows dedicated to the same activity so that their traffic patterns remain identical. Based on the clustering result, we implement an ensemble model in which each submodel only needs to profile a specific activity, which highly reduces the burden of a single model's generalization ability. For evaluation, we build a 57.1 GB IoT dataset collected in 9 months composed of comprehensive normal and malicious traffic. Our evaluation proves that IoTEnsemble possesses a state-of-the-art detection performance on various IoT botnet malware and attack traffic, exhibiting a significantly better result than other baselines in a more intelligent and functional IoT network.

**Keywords:** Internet of Things · Network anomaly detection · Malware detection · Botnet

---

This work is supported by the National Key Research and Development Project of China under grant No. 2020AAA0107704, National Natural Science Foundation of China under grant No. 61972189 and 62073263, Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989, and Research Center for Computer Network (Shenzhen) Ministry of Education.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022  
V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13555, pp. 569–588, 2022.  
[https://doi.org/10.1007/978-3-031-17146-8\\_28](https://doi.org/10.1007/978-3-031-17146-8_28)

## 1 Introduction

The number of the Internet of Things (IoT) connections worldwide will reach 5.8 billion by 2029 [5]. Nevertheless, IoT security issues remain severe, such as insecure communications, lack of timely firmware updates, and weak configurations by consumers, which highly increase the risk of being attacked [1].

The network anomaly detection system (NADS) is a promising solution, which learns the pattern of normal traffic and can detect known and unknown attack traffic by deviation from the normality. Since IoT devices are typically low-functional, their traffic is relatively easy to model, and many studies have proposed effective NADS solutions for IoT networks [12, 26, 28, 37].

However, due to emerging technologies applied to IoT networks like edge/fog computing and Artificial Intelligence of Things (AIoT), IoT devices are becoming more functional than ever before [4], and thus the scope of “normal” network activities becomes harder to define. Technically, different activities are far from conforming to an independent identical distribution (i.i.d.), which challenges the fundamental assumption of many algorithms [12, 26, 31, 37]. Besides, the amount and frequency of activities greatly depend on the user’s habits, such as a smart camera barely connecting to a command server unless a user opens its app. As a result, a NADS might produce inaccurate results on infrequent functions. These facts are challenging the practicality of existing NADS.

To address them, a conceivable method is to “divide and conquer”: instead of building a single comprehensive model, we cluster the traffic for each activity and build multiple fine-grained *submodels* into an *ensemble model*, which reduces the difficulty for one model to learn with a more specific pattern of normality. However, to achieve it, the following challenges should be resolved:

- 1) **Activity clustering.** Many IoT devices use non-standard domain resolution, dynamic ports or no certificates, possibly making rule-based clustering methods [10, 13] unreliable. Besides, statistical clustering like K-Means usually needs a priori like the number of clusters. Too coarse-grained clustering has little significance, while too fine-grained clustering leads to excessive loads.
- 2) **Heterogeneous IoT traffic.** A variety of network protocols is used for IoT communications, such as HTTP, MQTT, XMPP, CoAP and even many proprietary protocols. This fact invalidates many proposed traffic representation methods since they only focus on specific protocols [10, 29, 34] or rely on deep packet inspection [38] which is incapable of handling encrypted traffic.
- 3) **Nontrivial overhead.** Running an ensemble of detection submodels simultaneously might be unrealistic for some deployment positions, such as a home router or a gateway [28, 37]. Also, since a NADS aims to treat all incoming traffic suspicious and inspect each traffic flow, the mapping process from a 5-tuple flow to an activity cluster must be efficient to avoid extra processing delay.

Toward this end, this paper proposes IoTEnsemble, which resolves the challenges above by the following design: 1) We propose a tree-based activity clustering algorithm that combines traffic rules and traffic statistics to produce a reliable clustering result. 2) We design a preprocessing pipeline that transforms

any raw IP traffic into a data representation that only requires the first few packets of a flow. 3) The clustering tree generates a set of rules that fast maps a 5-tuple flow to an activity and provides a *trigger-action* mechanism for only limited number of submodels being awakened during the execution.

We configure a real-world IoT testbed composed of 28 IoT devices running for 9 months. The experiment shows that IoTEnsemble outperforms the state-of-the-art NADS in the detection of botnet-related attacks. In particular, while other baselines are greatly affected by the increasing activities, IoTEnsemble shows extremely little reduction in effectiveness, which demonstrates its feasibility to secure a more diverse IoT ecosystem in the future.

We summarize our contributions as follows: 1) A reliable and efficient clustering algorithm for device activities; 2) A state-of-the-art NADS that uses an ensemble of autoencoders to profile diverse activities of increasingly functional IoT networks; 3) A real-world IoT testbed with a diversity of devices and activities, contributing a 57.1 GB dataset as a benchmark for IoT networks.<sup>1</sup>

## 2 Related Work

### 2.1 IoT Security, Malware and Botnet

A report in 2016 gives a detailed summary of IoT threats, including vulnerabilities, insecure communications, data leaks, malware, service disruption, persistence of these problems and “disposable” devices [1]. Among these threats, malware is a persistently annoying issue, mainly because IoT devices have relatively weak protections and improper configurations. Mirai, one of the most notorious IoT malware that compromised over 600,000 devices and launched a 620 Gbps DDoS attack in 2016, has been studied by many researchers [6, 17]. Its infrastructure includes a report server, a loader server and a command and control (C&C) server. As its source code was somehow released [11], Mirai becomes a paradigm for many variations [3, 24, 27]. Infected bots can be used for DDoS attacks, scanning, spamming, data leaking or even cryptomining [2].

### 2.2 Network Anomaly Detection System

The network anomaly detection system (NADS) is commonly used against botnet and other network attacks. Bhuyan et al. summarize over 200 related works and categorize NADS into seven classes [9]. Among these classes, many machine learning-based (ML) methods are used, such as KNN, SVM, decision tree and neural network. Recently, the community turns to deep learning (DL) for its great generalization ability [20, 25]. Compared to traditional ML, an advantage of DL is to automate the feature extraction from raw traffic to reduce the reliance on feature engineering. Tang et al. propose a seq2seq model to detect zero-day attacks after a web application firewall [33]. However, it can only parse HTTP packets, which is not suitable for heterogeneous IoT traffic. Marin et al. propose

<sup>1</sup> Our datasets are made public: <https://github.com/HeliosHuang/ESORICS>.

a DL-based malware traffic detection method relying on no handcrafted feature engineering but using raw bytes as data representation [23]. Despite achieving high accuracy, it lacks interpretability on encrypted traffic and may cause privacy violations in IoT scenarios. Mirsky et al. propose Kitsune, an unsupervised NADS using an ensemble architecture similar to ours and achieving a state-of-art performance [28], but it does not separate a device's entire activity into multiple specific activities. We will use Kitsune as a baseline in the evaluation to highlight the advantage of IoTEnsemble.

### 2.3 Activity Clustering

Activity clustering is a technique to group the traffic dedicated to the same activity. It can be primarily divided into rule-based methods and statistical methods. Rule-based methods identify the flows with the same or a certain range of domain names [10], destination ports [15] or TLS/SSL certificates [13] as the same activity, which has great interpretability. However, simply matching the rules with IoT traffic may cause unreliable clustering results since many IoT devices are not manufactured to use fixed domains and ports. On the other hand, statistical methods typically use statistical features of traffic along with ML algorithms for clustering, such as packet size and flow rate [14, 29]. The disadvantage of these methods includes relatively weak interpretability and unstable validity due to high sensitivity to the fluctuation of the statistical features. Other applications like device identification [22, 34] and app fingerprinting [10, 13] also use activity clustering techniques, but their design goal is different from ours as they can ignore the commonly used activities between devices or apps and only identifies a small proportion of unique activities for fingerprinting, whereas a NADS needs to handle all traffic efficiently because it treats every flow as a suspicious target.

## 3 Threat Model

In this paper, adversaries are the people who infect and compromise IoT devices as part of a botnet. They are assumed to have the following capabilities: 1) they can infect a device by known or unknown exploits, which means zero-day attacks are possible; 2) they can either be located in the same local area network (LAN) as IoT devices or outside the LAN; 3) after infecting a device, they can command the victim device to take malicious actions, such as data leaking, cryptomining, spamming or attacking other devices and websites.

Our system is positioned inside the same LAN as the IoT devices to be protected. The deployment could be on a home gateway or a computer that is able to sniff all the traffic from and to the IoT devices, including the traffic across the devices inside the LAN, which can be realized by mirroring the traffic through the LAN interface on the home gateway to our system. Besides, we assume that a newly connected device can be simply identified by its DHCP messages and MAC addresses so that the traffic can be separated.

To define a clear scope of this paper, we also make the following assumptions: 1) IoT vendors are not adversaries; in other words, IoT devices are not

manufactured initially to be malicious and no backdoors are pre-installed; 2) the malicious activities launched by adversaries leave a trace on L3 network layer. This paper does not consider the attacks on L1 and L2 network layers, such as spoofing attacks on Bluetooth Low Energy (BLE).

## 4 Flow Clustering by Activity

### 4.1 IoT Network

Recent years have witnessed IoT’s astonishing promotion in functions, such as AI-based functions like movement detection and face recognition in IP cameras. In addition, many IoT vendors like Samsung SmartThings are building an ecosystem for easy communications between devices. Accordingly, these functions are reflected on the network by a variety of connections.

Although IoT devices are becoming more intelligent, they are still far from general-purpose devices as most of their network communications are pre-set by their vendors. This paper uses the expression *network activity* or simply *activity* referring to a purpose that the network communications are dedicated to.

### 4.2 Design Goal

In this paper, the ultimate purpose of clustering is to group the traffic flows for the same activity so that a submodel can easily learn each activity’s normal pattern. Given this purpose and the condition of IoT networks, we list three design goals for the clustering algorithm: 1) It has good interpretability for network administrators to understand; 2) It needs as little prior knowledge as possible, such as what protocols to use or the number of clusters; 3) It has an appropriate granularity with as few numbers of clusters as possible and meanwhile still achieves a reliable clustering validity.

To achieve these goals is not easy. For one thing, most ML-based clustering methods disobey them. For another, rule-based methods usually cannot satisfy the requirement of reliability due to the unstable IoT traffic rules deriving from the use of non-standard domain resolution, dynamic ports, no TLS/SSL authentication and proprietary protocols [15]. To propose a proper method, we firstly give a motivating case about our observations on IoT traffic and activities.

### 4.3 A Motivating Case: TP-Link Camera

We manually traverse the functions of a TP-Link camera and monitor its network activities by Wireshark. By associating the traffic with the triggered functions, we notice seven activities (Table 1) including device status reporting (dev), API call (api), business (biz, related to cloud service), stream relaying (relay), UPnP broadcasting (UPnP), streaming to local apps (local) and request for public IP (STUN) with their traffic characteristics, where  $\mathbb{S}_{ps}$  is the set of packet sizes (TLS handshakes in bold) and  $\Delta t$  is the mean of packet inter-arrival time in two directions. We present our three observations on these activities:

**Table 1.** Traffic characteristics of seven activities

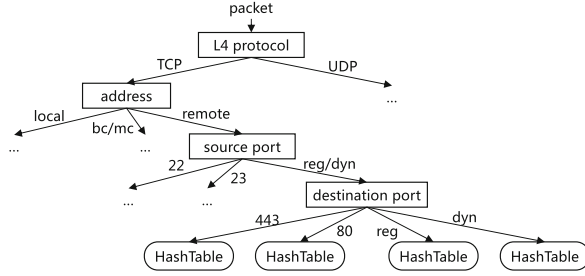
Activity	Domain/IP	Protocol (port)	Period	TLS/SSL?	$\mathbb{S}_{ps}$	$\Delta t^{out}$	$\Delta t^{in}$
dev	n-device-api.; dynamic IP	TCP (50443)	30 s	✓	153, 37, 52	0.25	0.042
api	n-devs-ipc	TCP (443)	3600 s	✓	<b>134, 873, 319,</b> 143, 457, 329, 121	12.59	0.12
biz	biz-ipc	TCP (443)	–	✓	<b>122, 849, 307,</b> 131, 333, 141, 301, 237, 109	0.021	0.039
relay	n-txc-relay-ipc-nj	TCP (443)	900 s	✓	<b>122, 857, 307,</b> 131, 237, 381, 109	0.024	0.052
UPnP	255.255.255.255; local IP	UDP (5001)	20 s	×	278	20.0	–
local	local IP	UDP (dynamic)	–	×	48, 69, 92, 1476	0.15	0.15
STUN	dynamic IP	UDP (dynamic)	–	×	56, 96	0.097	0.024

**Observation 1:** *A device could contain multiple activities that show greatly different traffic patterns.* For example, some of the activities in Table 1 use encryption protocols, which clearly present distinct traffic patterns from the other plaintext traffic. Also, these highly asynchronous activities due to diverse periods result in different patterns of the device traffic in different time. These facts challenge the generalization ability of training a single model to profile the complex characteristics of the mixed traffic by diverse activities.

**Observation 2:** *Rules like domains, protocols and ports are still effective for preliminary clustering but do not always work.* For example, the “dev” activity contacts dynamic IP addresses not in any of its DNS answers (we did power off, reset and reboot the device before the experiment to clear the cached domain resolutions). Some ports like 443 and 5001 are stably used by certain activities, but activities like local streaming and STUN always use random dynamic ports.

**Observation 3:** *The set of packet sizes and inter-arrival time are surprisingly helpful for distinguishing the activities.* In fact, Trimananda et al. have found a similar observation on the packet size of a request-reply packet sequence [34], but it only works for TCP connections. We extend this observation by finding that the request-reply sequence is not necessary – instead, a limited set of packet sizes has identical efficacy, which also works with UDP activities. To the best of our knowledge, it is the first time to find such an easy but effective observation for activity clustering. We explain this observation by two points: 1) TLS provides a stable and differential certificate exchange process (packet size in bold); 2) IoT communications comply with strict data formatting.

We also analyze the traffic from other types of devices such as plugs and speakers and find consistent observations. We attribute these observations to the typically purpose-driven design of IoT devices – though a device may contain a diverse set of functions, each of them is pre-set by the firmware for a specific purpose and hardly changes its traffic patterns. This preliminary analysis suggests a possibility to combine traffic rules with spatial and temporal statistics for reliable activity clustering.



**Fig. 1.** An illustration of the tree-based activity clustering; bc/mc: broadcasting/multicasting; sys: system port (0-1023); reg: registration port (1024-32767 in Linux); dyn: dynamic port (32768-65535 in Linux).

#### 4.4 Clustering Method

We present a tree-based clustering method for IoT traffic activity as Fig. 1 illustrates. Given Observation 2, a packet belonging to a bidirectional flow  $f := (\text{device-IP}, \text{dst-domain/IP}, \text{src-port}, \text{dst-port}, \text{protocol})$  goes through the four-level hierarchical rules for a preliminary clustering: 1) L4 protocol: protocol is TCP or UDP; 2) address: dst-domain/IP is a specific domain, a remote IP address, a local IP address or a broadcasting/multicasting address; 3) src-port is a specific system port, or in registration/dynamic port range; 4) destination port: dst-port is a specific system port, in registration port range or in dynamic port range.

Note that the use of each system port has an individual tree path as it typically indicates a specific service. Since IoT devices are at the client end, the source port is usually useless for clustering (unless it is a system port like 22 for SSH, 23 for Telnet, etc.) while the destination port in registration range reveals some commonly used IoT services, such as 1900 for SSDP, 3478 for STUN, etc.

At the end of the tree, each leaf node is a hash table in which the key is  $f$  and the value is an incremental statistical structure. This structure is defined by a five tuple  $IS := (N^{in}, N^{out}, T^{in}, T^{out}, \mathbb{S})$ , where  $N$  is the number of packets,  $T$  is the sum of packet inter-arrival time,  $in$  and  $out$  indicate the direction,  $\mathbb{S}$  is the set of packet sizes. The advantage of  $IS$  is its constant storage complexity no matter how many packets a flow has. When a packet with direction  $d$ , packet size  $s$  and inter-arrival time  $\Delta t$  arrives at a leaf node,  $IS$  is updated by:

$$\begin{aligned}
 N^{in} &\leftarrow N^{in} + 1 \text{ if } d = in \text{ else } N^{out} \leftarrow N^{out} + 1 \\
 T^{in} &\leftarrow T^{in} + \Delta t \text{ if } d = in \text{ else } T^{out} \leftarrow T^{out} + \Delta t \\
 \mathbb{S} &.add(s)
 \end{aligned} \tag{1}$$

For each leaf node, a more abstract flow rule can be obtained by merging the flows on the leaf node to reduce the final number of clusters. Each two flows  $f_1$ ,  $f_2$  with their incremental statistics  $IS_1$ ,  $IS_2$  are compared by two aspects:

---

**Algorithm 1.** ClusterMerging

---

**Input:** An empty set  $K$ , hash table  $H$ , 5-tuple set  $L$

```

1  $f \leftarrow L.Pop()$ ;
2  $IS := (N^{in}, N^{out}, T^{in}, T^{out}, \mathbb{S}) \leftarrow H[f]$ ;
3 for  $f_i$  in  $L$  do
4   if  $f_i \subseteq f$  then
5      $L.remove(f_i)$ ; continue;
6      $IS_i := (N_i^{in}, N_i^{out}, T_i^{in}, T_i^{out}, \mathbb{S}_i) \leftarrow H[f_i]$ ;
7     if  $J(\mathbb{S}, \mathbb{S}_i) > h_s$  and  $b(\lambda, \lambda_i) < h_t$  then
8        $L.remove(f_i)$ ;  $f \leftarrow merge(f, f_i)$ ;
9   end for
10  $K.add(f)$ ;
11 ClusterMerging( $K, H, L$ );
```

---

**Spatial Correlation.**  $\mathbb{S}_1, \mathbb{S}_2$  are compared by the Jaccard index:  $J(\mathbb{S}_1, \mathbb{S}_2) = \frac{|\mathbb{S}_1 \cap \mathbb{S}_2|}{|\mathbb{S}_1 \cup \mathbb{S}_2|}$ . If the result surpasses a threshold  $h_s$ ,  $f_1$  and  $f_2$  are believed to be spatially correlated.

**Temporal Correlation.** Packet inter-arrival time is typically modelled by a Poisson process [7], which conforms to an exponential distribution  $f(t) = \lambda e^{-\lambda t}$ . By a set of observations  $t_1, t_2, \dots, t_N$ , parameter  $\lambda$  can be calculated by maximum likelihood estimation:  $\lambda = \frac{N}{T}$ . The theoretical derivation of the estimation is in Appendix A. Accordingly,  $\lambda_1$  and  $\lambda_2$  can be derived from  $IS_1$  and  $IS_2$  to determine their distributions. We separately calculate the difference between two parameters for two directions:  $b(\lambda_1, \lambda_2) = \frac{|\lambda_1 - \lambda_2|}{max(\lambda_1, \lambda_2)}$ . If any of the results is below a threshold  $h_t$ ,  $f_1$  and  $f_2$  are believed to be temporally correlated.

For each hash table  $H$  at a leaf node and the set of its 5-tuple keys  $L$ , we use a greedy strategy to compare and merge the flow rules as Algorithm 1 describes. In line 8, if two flows are correlated, they are merged into a new five tuple where the new address and port will be their path name in the clustering tree (e.g., “local”, “reg/dyn”, “dyn”). If the merged destination domains have a common secondary-level domain, the new address will use a wildcard (e.g., \*.tplink.com). The symbol of subset in line 4 means that  $f$  has a more abstract expression that contains  $f_i$  (e.g., src-port is 29983 in  $f_i$  and “reg/dyn” in  $f$ ). This algorithm is recursive that has an average complexity of  $O(n \log n)$ . By using the Depth-First-Search (DFS) on the entire tree, the tuple sets on each leaf node are finally merged into a single set  $K$ . We call the tuples in this set *activity keys* as each of them indicates a rule for one activity. The next section presents the use of activity keys for the construction of our framework.



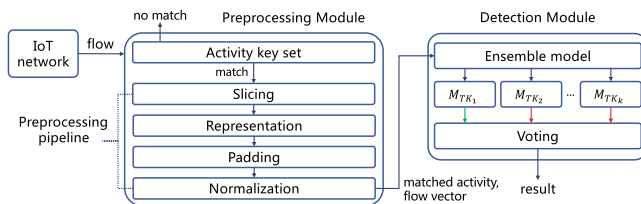


Fig. 2. An overview of IoTEnsemble’s architecture.

## 5 IoTEnsemble Design

### 5.1 Overview

IoTEnsemble is an anomaly detection framework designed for current increasingly intelligent and functional IoT networks. Generally, it is a combination of two detection stages:

**Rule matching** by activity keys that efficiently filters out the extremely anomalous traffic flows with unknown domain, address or port range.

**Ensemble model** that learns each activity and detects anomalies by the deviation from the normal traffic pattern.

Figure 2 illustrates IoTEnsemble’s overview. The preprocessing module firstly handles the IoT network traffic and matches 5-tuple flows to activity keys. Subsequently, it processes the first  $r$  packets of each flow into a numerical data representation, where  $r$  is the fixed length of a packet window. The detection module is composed of multiple submodels that profile one activity respectively. It follows a *trigger-action* mechanism which only awakens the submodels with similar activities to the flow for the inspection, and the final detection result is voted only by the awakened submodels. Compared to the basic ensemble architecture, this mechanism reduces the overhead of multiple submodels’ simultaneous execution. Since IoTEnsemble is neither privacy-intrusive nor protocol-specific, it is trustworthy and practical for wide usage scenarios. For example, the preprocessing module can be deployed on a home router and the detection module can be executed on a router, a personal computer or an edge server.

IoTEnsemble has two phases during the operation. The *learning phase* starts with a new device joining the network and assumes that it will not be compromised in an initial time window (e.g., one or two hours). During this period, IoTEnsemble generates the activity key set and trains an ensemble model using the observed network traffic. Then the framework enters the *detection phase* in which the internal parameters of IoTEnsemble are fixed for inference. We discuss how each component works in greater detail in the following subsections.

### 5.2 Preprocessing Module

**Activity Key Set.** As assumed in Sect. 3, all traffic is considered benign during an initial observation window. Hence, we generate an activity key set and use this set as a profile of normal activities for rule matching.

**Construction in Observation Window:** As a device goes through its functions to present all its activities, the activity key set is constructed by the following steps: 1) Initialize a clustering tree as Fig. 1; 2) Each tuple enters the tree and follows the path condition to reach a leaf node and is stored in the hash table; 3) At the end of the observation window, conduct the DFS on the tree and run Algorithm 1 on each leaf node; 4) Merge the activity key set from each leaf node and output the complete set  $K$ .

**Utilization in Detection Phase:** An incoming 5-tuple flow is mapped to activity keys, which is simply a matching process with at most  $O(n)$  complexity and is practical for online use. Given the abstract address and port range indicators in an activity key (e.g., wildcard domains, “local”, “reg/dyn”, etc.), it is a relaxed matching process and may result in multiple matches, which gives a chance of further inspection to every possible activity key to reduce false positives. However, if a flow cannot match any of the activity keys, it will be directly labeled as malicious traffic.

**Preprocessing Pipeline.** A traffic flow that matches activity keys will enter the second detection stage by the ensemble model. To make a network flow processible by an ML model, a preprocessing pipeline is built to convert a flow of packets into a numerical vector. Formally, given a flow  $f$  and its packets by an ordered list  $\mathbf{p}_f = [p_1, p_2, \dots, p_l]$ , it follows a pipeline of functions  $\{\mathcal{S}(\textit{Slicing}), \mathcal{R}(\textit{Representation}), \mathcal{P}(\textit{Padding}), \mathcal{N}(\textit{Normalization})\}$  to generate a representation that describes the underlying characteristics of the flow.

**Slicing:** Some IoT flows last extremely long or even never end, such as keepalive traffic with the server. To deal with this, we use a time window to slice a flow into multiple small flows with equal intervals. We set two time windows separately for TCP and UDP denoted by  $t_T$  and  $t_U$ .

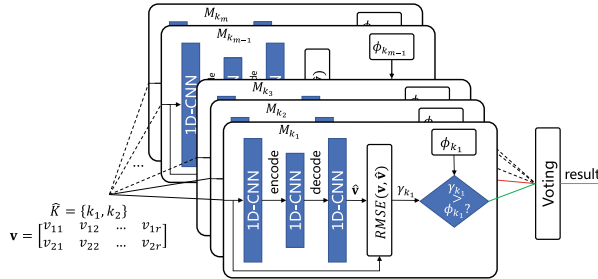
**Representation:** The data representation derives from the sequential relationship of IP packet size and inter-arrival time between packets in a flow. It is motivated by the characteristics of botnets that generate sequential small packets when searching for susceptible hosts and follow a uniform pattern throughout C&C life-cycle to reduce their observability on the network [36]. Packet size and inter-arrival time form two ordered lists that only preserve the first  $r$  packets. Based on our preliminary experiment, the first 100 packets have sufficed an effective representation for the entire flow.

**Padding:** A list with less than  $r$  items is zero-padded to satisfy a fixed-length input for an ML algorithm. Another use of padding is to reveal the length of a flow – a long flow has more values in a list compared to short flows.

**Normalization:** We adopt the L2 norm to make each value in a list between 0 and 1 and stack two lists into a two-dimensional sequential data sample.

### 5.3 Anomaly Detection Module

**Ensemble Architecture.** Figure 3 illustrates the overall architecture, which is a combination of multiple unsupervised learning submodels that learn normal activity’s traffic pattern. The advantages of using an unsupervised learning algorithm include 1) no need for malicious traffic data for training; 2) the capability of discovering zero-day attacks. We believe that such an ensemble architecture is a good solution to an increasingly functional IoT network given a submodel only learns a specific activity which better conforms to an i.i.d.



**Fig. 3.** An illustration of our ensemble model; only the submodels with matched activity keys engage in the execution.

A submodel is constructed by a one-dimensional Convolutional Neural Network autoencoder (1D-CNN AE) [18]. It is an unsupervised model that brings CNN’s invariant feature extraction to sequential data. It learns the latent distribution of the training data and adjusts its parameters to minimize the reconstruction error between the input and output measured by root mean squared error (RMSE). In the detection phase, a data sample inconsistent with the learned distribution produces a higher reconstruction error and thus can be detected.

**Learning Phase.** The training of an ensemble model starts by the end of the construction of activity key set  $K$ . A device’s network flows processed by the pipeline  $\{\mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{N}\}$  compose the training dataset denoted by  $\mathbf{X} = \{\mathbf{X}_{k_1}, \mathbf{X}_{k_2}, \dots, \mathbf{X}_{k_m}\}$ , where  $\mathbf{X}_{k_i}$  represents the traffic data belonging to the activity with activity key  $k_i$ . Each dataset is then split into a training subset for training a submodel’s parameter  $\theta$  and a validation subset for the determination of a hyperparameter  $\phi$ , i.e., the threshold of RMSE. It is determined by a quantile of order  $q$ , which can also be interpreted as the setting of true negative rate (TNR) on the validation subset. Finally, an ensemble model  $\mathbf{M}$  can be described as a set of submodels:  $\mathbf{M} = \{M_{k_1}, M_{k_2}, \dots, M_{k_m}\}$  where  $M_{k_i} = \{\theta_{k_i}, \phi_{k_i}\}$ ,  $m = |K|$ .

**Detection Phase.** When a flow passes the rule matching, the matched activity key set  $\hat{K}$  awakens the corresponding submodels in  $\mathbf{M}$  and each of them parallelly inspects the feature vector  $\mathbf{v}$  by comparing the reconstruction error to its threshold, which is called the *trigger-action* mechanism. A flow is judged as normal if any of the awakened submodels claims it is normal. The process of detection phase is described by Algorithm 2.

---

**Algorithm 2:** Ensemble Model Detection

---

**Input:** Ensemble model  $\mathbf{M}$ , feature vector  $\mathbf{v}$ , matched activity key set  $\hat{K}$   
**Output:** Detection result and failed activity key set  $\mathcal{Y}$

- 1 Initialize an empty set  $\mathcal{Y}$ ;
- 2 **for**  $k$  **in**  $\hat{K}$  **do**
- 3     Awaken submodel  $M_k = \{\theta, \phi\}$  from  $\mathbf{M}$ ;
- 4      $\hat{\mathbf{v}} \leftarrow h_\theta(\mathbf{v})$ ;
- 5      $\gamma \leftarrow RMSE(\mathbf{v}, \hat{\mathbf{v}})$ ;
- 6     **if**  $\gamma > \phi$  **then**
- 7         Append  $k$  into  $\mathcal{Y}$ ;
- 8 **end for**
- 9 **if**  $|\mathcal{Y}| < |\hat{K}|$  **then return** (Negative,  $\mathcal{Y}$ );
- 10 **else return** (Positive,  $\mathcal{Y}$ );

---

## 6 Evaluation

### 6.1 Testbed and Dataset

To demonstrate a realistic and functional IoT network, we set up a real-world testbed consisting of 28 popular IoT devices that cover most mainstream IoT vendors in China and diverse device types. We basically categorize them into *camera*, *sound box*, *gateway* and *appliance*. Figure 4 illustrates the network topology of our testbed. To present each device’s activity, we consider both manual and automated interaction as follows: 1) we explore each device’s functions by instructions and app UI and make a list of functions for our researchers to trigger at least once a day, such as pressing physical buttons, walking before a camera or talking to a sound box; besides, the testbed is located in a public location for free use; 2) we configure a laptop with Android Debug Bridge (ADB) and an emulator installed with 19 apps for all devices, and run a Python script to trigger their functions by app and voice commands (to four sound boxes) at regular intervals; for example, a smart camera is requested for streaming or a sound box plays a song at 2pm every day. This testbed has been run for 9 months and contributed over 57 GB data. Given that IoTEnsemble does not require a large amount of data for training, we use the data of the first week for training and the rest for testing. We also use two benchmark datasets: one from UNSW collected in a lab environment [32] and one from NEU collected in an idle status of IoT devices [30]. As synthesized without heterogeneous activities, they are used to compare the performance between functional and low-functional IoT networks.

We make two Raspberry Pi boards infected by IoT malware (Mirai, BASH-LITE [11]) and a C&C server (in WAN) for malware traffic collection. We modify the packet header information like IP address to make it consistent with the IoT devices. We also replay some public IoT botnet attack datasets as supplementary [2, 8, 19]. All traffic is collected by a computer via port mirroring by the router, making it a comprehensive IoT traffic dataset for evaluation.

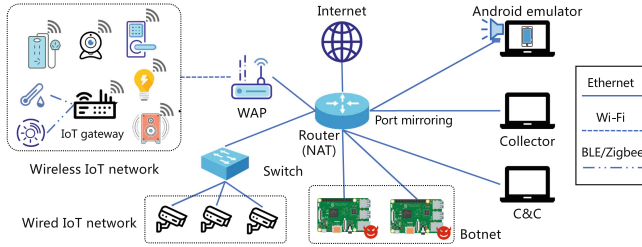


Fig. 4. Network topology of our testbed

## 6.2 Metrics

The detection performance of an IDS is typically evaluated by true positives ( $TP$ ), true negatives ( $TN$ ), false positives ( $FP$ ), false negatives ( $FN$ ) and their tradeoff. In our evaluations, we fix the true negative rate ( $TNR = \frac{TN}{TN+FP}$ ) by adjusting the threshold  $\phi$  and measure the true positive rate ( $TPR = \frac{TP}{TP+FN}$ ). For our framework, each sample in the dataset is considered benign only if it passes the inspection of both the detection stages.

The validity of activity clustering, however, is harder to evaluate as the target of clustering is more subjective and therefore it lacks a general metric [16]. For a fair and comprehensive comparison with both previous rule-based methods and statistical methods, we propose a network-specific metric called *cohesion index*, which is the mean of the standard deviation of the distance between every two 5-tuple flows within each cluster. It is further divided into *rule cohesion* ( $RC$ ) and *statistical cohesion* ( $SC$ ). In  $RC$ , the distance is measured by the header information: 1) add 1 if two destination IP addresses are not in the same range (i.e., local, external, multicasting); 2) add 1 if two ports are not in the same range (i.e., system port, register port, dynamic port); 3) add 1 if two L4 protocols are different (i.e., TCP, UDP, ICMP). In  $SC$ , the distance is measured by the vector distance composed of five commonly used statistical features for traffic profiling (mean and variance of packet size, mean and variance of inter-arrival time, inbound/outbound ratio). We also include the number of clusters ( $N_c$ ) in the metrics. To present the tradeoff between the cluster number and the validity, which is one of our design goals, we multiple the two cohesion indexes (after normalization) and the cluster number to form  $TO_R$  and  $TO_S$ . A method with smaller cohesion and tradeoff indexes is better for our design goal.

## 6.3 Validity of Activity Clustering

Firstly, our method clearly satisfies the first two goals because: 1) it generates a set of rules by the tree paths with good interpretability; 2) it does not assume any prior knowledge about the traffic. As for the third goal, we compare our method with four baselines, including two rule-based methods: 1) FlowPrint [13] that uses destination and port numbers along with TLS certificates for clustering;

**Table 2.** Comparison between activity clustering methods.

Type	Method	$RC$	$SC$	$N_c$	$TO_R$	$TO_S$
Rule-based	FlowPrint	0.0029	0.4096	220.33	0.6407	90.24
	MUDgee	0.2791	0.8775	29.22	8.156	25.64
Statistical	Botminer	0.7185	0.5142	68.67	49.34	35.31
	MCluster	0.4531	0.4682	110.56	50.09	51.76
Hybrid	Ours	0.0682	0.6706	12.33	0.8414	8.270

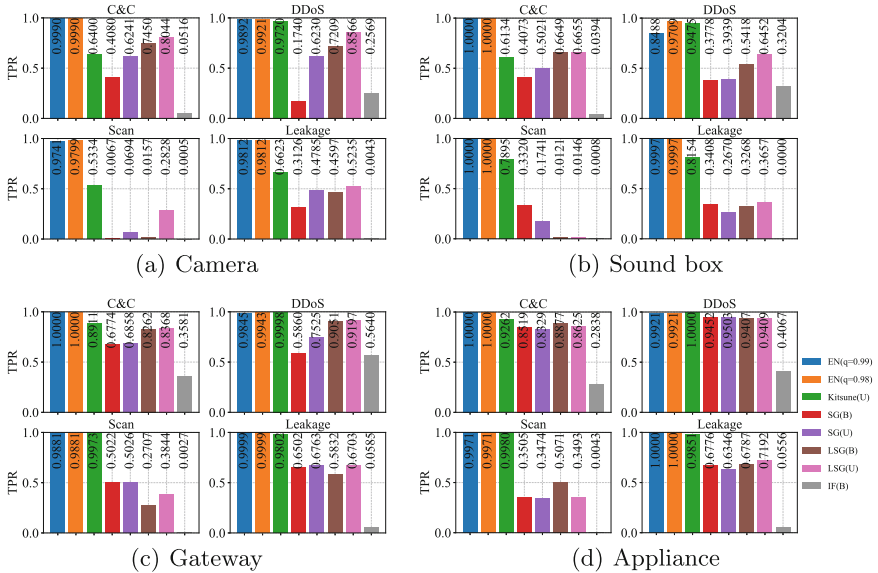
2) MUDgee [15] that uses the rules upon the YANG model (RFC 6020) to separate network activities; and two statistical methods: 3) Botminer [14], a two-stage clustering framework using X-Means; 4) MCluster [29], a network-level behavioral clustering system for malware detection.

We use a one-week dataset of normal device traffic for the evaluation. Table 2 shows the result. It can be seen that the rule-based methods have lower  $RC$  and higher  $SC$ , and vice versa for the statistical methods, which is in line with their design. Besides, we notice that the methods with the lowest  $RC$  (FlowPrint) and the lowest  $SC$  (MCluster) result in a great number of clusters. It is reasonable since more fine-grained clustering generally owns better cohesion inside each cluster, while it does not accord with our design goal to reduce the overhead of cluster numbers. Note that a high  $RC$  suggests the possibility of obviously wrong clustering, such as treating a TCP flow and a UDP flow as the same activity just because they are coincidentally similar in statistics. In contrast, our method combines both of the advantages, using heuristic rules for preliminary clustering and statistics for cluster merging, which demonstrates a better tradeoff between the validity and the number of clusters. It proves that our method outperforms previous works in the case of our design goal.

We also measure the observation window for a device to present all its activities. Before the experiment, we power off the devices and reset them to make sure the cached configurations are cleared. The manual interaction and automated interaction methods mentioned above are separately adopted to traverse the device’s activities. It shows that the average time of exhibiting all activities by manual use is 20 days, whereas the average time of exhibiting all activities by our automated script is only 100 min. It implies that the activity key set can be reliably generated only within one or two hours after the connection, which is practical for real use. The full result can be found in Appendix B.

## 6.4 Overall Performance

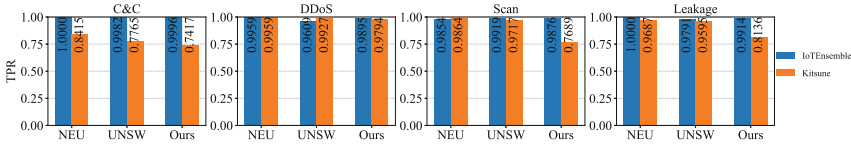
We train IoTEnsemble and evaluate its detection performance on four categories of attacks: C&C, DDoS, Scan and Leakage. We empirically set  $h_s$  and  $h_t$  to 0.4 and 0.5, resulting in a stable number of clusters. The time window for both UDP and TCP flows is set to 30 s. The sequence length  $r$  is 100. For each submodel, the total amount of trainable parameters is 10530. We set two



**Fig. 5.** Detection performance of IoTEnsemble ( $q = 0.99$  and  $q = 0.98$ ) and five baselines on four categories of attacks.

thresholds:  $q = 0.99$  and  $q = 0.98$ , which means the FPR is set to be 0.01 and 0.02 on the validation dataset. To demonstrate the advantage of IoTEnsemble over a single model's architecture, six unsupervised baseline NADS are used: 1) Kitsune [28], a state-of-the-art unsupervised NADS using an ensemble model of multiple autoencoders; 2)  $SG(U)$ , a single 1D-CNN AE model identical to the submodel in IoTEnsemble trained without balanced data from different activities (unbalanced training); 3)  $SG(B)$  a single model identical to  $SG(U)$  trained with balanced data from different activities by oversampling (balanced training); 4)  $LSG(U)$ , a large single 1D-CNN AE model with approximately 16 times trainable parameters as  $SG(U)$  by unbalanced training; 5) a large single model identical to  $LSG(U)$  by balanced training; 6)  $IF(B)$  [21], an isolation forest by balanced training. The threshold of the baselines is uniformly set by an FPR of 0.02.

Figure 5 illustrates the experiment result. Detailed results of each device are in Appendix B. It shows that IoTEnsemble outperforms Kitsune and other baselines in all four attacks, suggesting that IoTEnsemble achieves a state-of-the-art detection performance. We attribute the advantage to our reliable activity clustering that makes the pattern of traffic better conform to an i.i.d. Besides,  $LSG$  exhibits insignificant superiority over  $SG$  compared to the large difference between their scales, showing that a single model's generalization ability is challenged by the variety of activities. Another finding is that all baselines including Kitsune exhibit worse detection performance on complex devices like cameras and sound boxes, while IoTEnsemble shows little difference.



**Fig. 6.** Comparison of detection performance on three datasets.

Figure 6 shows the comparison between IoTEnsemble and Kitsune on three datasets, whose diversity of activities is getting enriched from left to right. It again proves that the variety of device functionality apparently affects the performance of Kitsune and meanwhile barely influences IoTEnsemble.

We highlight IoTEnsemble’s runtime advantage by its average 2.1 submodels simultaneously awakened during the execution (camera: 1.7/11.3, sound box: 3.9/12.7, gateway: 1.4/5.2, appliance: 1.4/5.7). Compared to a normal ensemble model like Kitsune that has over 20 submodels for parallel inference, IoTEnsemble is more efficient thanks to the trigger-action mechanism that only awakens the best match submodels for execution. Moreover, we measure the average inference time of the ensemble model on a CPU computer (Intel Xeon(R) Gold 5117 @ 2.00 GHz with one single core used), resulting in about 244.59  $\mu$ s, which is extremely trivial compared to the time window of a sliced flow (30 s).

## 6.5 Discussion

**Possible Attacks.** It is possible that malware bypasses the first detection stage by disguising the C&C channel as a normal service like HTTP. To assess its impact, we modify half of the C&C traffic of four malware by encapsulation with HTTP headers and ports. We choose part of the devices that normally use HTTP traffic as victims so that the disguised C&C traffic can bypass their rules. As Table 3 shows, even if they evade the first detection stage, most malware traffic can still be detected by the second detection stage, which presents the robustness of IoTEnsemble. As for evasive attacks [35], given the trigger-action mechanism, it is difficult to find a perturbation for a dynamic combination of submodels while maintain the malicious function. IoTEnsemble may be susceptible to *poisoning attacks* that spoil the training data. However, it is not easy for an adversary to compromise a device so fast considering our short observation window, and afterwards most of the poisoning traffic can be easily filtered out.

**Limitations.** One limitation of IoTEnsemble is the false positive caused by normal but unseen activities, which is also an inherent drawback of unsupervised learning-based methods. Nevertheless, as we have shown in the experiment, a deliberately designed script can reduce the trigger time of complete device activities from days to minutes, which suggests a trivial probability of missing learning from existing activities. Meanwhile, we believe that this limitation can be more highlighted when new functions are induced by IoT firmware/software updates.



**Table 3.** Detection on modified malware that bypasses the first detection stage.

Malware	Camera	Soundbox	Gateway	Appliance
Aidra	0.647	0.873	0.987	0.873
BASHLITE	0.958	1.00	1.00	1.00
Mirai	1.00	0.985	0.596	0.985
Xbash	1.00	1.00	1.00	1.00

To address it, a solution is to add a feedback function to IoTEnsemble by developing a controller app on the cellphone or the PC. An unseen activity is reported to the app for a manual decision from the administrator, so that a corresponding rule can be added to the first detection stage of IoTEnsemble to avoid the same false positives. We consider the development of this function as part of our future work.

## 7 Conclusion and Future Work

This paper presents IoTEnsemble, an anomaly detection framework against IoT botnet attacks. It is designed to accommodate current increasingly functional IoT devices with a variety of activities. It contains a two-stage detection process of rule matching by a tree-based clustering algorithm and an ensemble model. We set up a real-world testbed to generate a dataset that reveals the diversity of IoT activities. The experiment result shows that, no matter how many activities an IoT network exhibits, IoTEnsemble obtains a state-of-the-art performance on botnet attacks. Our future work is to design and implement a plug-and-play programmable home router that secures an IoT network based on IoTEnsemble. To fulfill this goal, we are trying to transform IoTEnsemble into a more lightweight model that consumes fewer resources while still maintains its effectiveness.

## A Maximum Likelihood Estimation

Suppose  $N$  observations of packet inter-arrival time  $t_1, t_2, \dots, t_N$  that conform to an exponential distribution  $f(t) = \lambda e^{-\lambda t}$  are sampled.  $T$  represents the sum of each observation, i.e.,  $T = \sum_{i=1}^N t_i$ . The maximum likelihood estimation of the parameter  $\lambda$  is derived as follow:

$$\begin{aligned}
 \lambda &= \arg \max_{\lambda} \prod_{i=1}^N \lambda e^{-\lambda t_i} = \arg \max_{\lambda} \sum_{i=1}^N \ln \lambda e^{-\lambda t_i} \\
 &= \arg \max_{\lambda} N \ln \lambda - \sum_{i=1}^N \lambda t_i \\
 &= \frac{N}{\sum_{i=1}^N t_i} = \frac{N}{T}
 \end{aligned}$$

## B Complete Dataset Information and Evaluation Result

See Table 4.

**Table 4.** Complete experimental result of each device in the testbed;  $N_c$  is the number of activity clusters;  $t_0$  and  $t_1$  are required time to observe all activities of a device by manual use and automated script; the last 8 devices are not IP-enabled and they are connected to the corresponding gateways.

Device	Size	$N_c$	$t_0$ (day)	$t_1$ (min)	C&C	DDoS	Scan	Leakage	C&C	DDoS	Scan	Leakage
					$q = 0.99$				$q = 0.98$			
360-camera	4.7 GB	13	6	150	1.0	1.0	0.992	1.0	1.0	1.0	0.992	1.0
360-doorbell	6.5 GB	17	6	100	1.0	0.956	1.0	1.0	1.0	0.976	1.0	1.0
EZVIZ-camera	2.9 GB	14	5	50	1.0	0.998	0.958	0.999	1.0	1.0	0.958	0.999
Hichip-camera	320 MB	7	1	50	1.0	0.973	0.998	0.835	1.0	0.976	1.0	0.835
Mercury-camera	2.5 GB	13	13	100	0.993	1.0	0.992	1.0	0.993	1.0	0.992	1.0
Philips-camera	7.0 GB	6	2	80	1.0	1.0	0.951	1.0	1.0	1.0	0.951	1.0
Skyworth-camera	1.2 GB	7	6	100	0.998	1.0	0.942	0.997	0.998	1.0	0.994	0.997
TPLink-camera	3.8 GB	11	9	100	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Xiaomi-camera	4.7 GB	14	13	120	1.0	0.976	0.933	1.0	1.0	0.976	0.933	1.0
Biu-speaker	3.4 GB	6	5	240	1.0	0.999	1.0	1.0	1.0	1.0	1.0	1.0
Xiaomi-soundbox	9.5 GB	23	7	390	1.0	0.913	1.0	0.999	1.0	0.927	1.0	0.999
Xiaodu-audio	7.7 GB	9	3	280	1.0	0.635	1.0	1.0	1.0	0.986	1.0	1.0
Aqara-gateway	366 MB	5	7	30	1.0	0.962	0.941	1.0	1.0	0.995	0.941	1.0
Gree-gateway	242 MB	5	1	30	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
TCL-gateway	903 MB	4	1	30	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
iHORN-gateway	241 MB	3	1	20	1.0	1.0	1.0	0.999	1.0	1.0	1.0	0.999
Xiaomi-gateway	594 MB	9	3	40	1.0	0.960	1.0	1.0	1.0	0.976	1.0	1.0
HONYAR-strip	211 MB	5	1	40	1.0	1.0	0.991	1.0	1.0	1.0	0.991	1.0
WiZ-LED	221 MB	5	3	20	1.0	0.976	1.0	1.0	1.0	0.976	1.0	1.0
Xiaomi-plug	208 MB	7	5	30	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
iHorn-temperature	\	\	\	\	\	\	\	\	\	\	\	\
iHorn-door sensor	\	\	\	\	\	\	\	\	\	\	\	\
iHorn-body sensor	\	\	\	\	\	\	\	\	\	\	\	\
Xiaomi-light sensor	\	\	\	\	\	\	\	\	\	\	\	\
Xiaomi-temperature	\	\	\	\	\	\	\	\	\	\	\	\
Xiaodu-doorbell	\	\	\	\	\	\	\	\	\	\	\	\
Aqara-water sensor	\	\	\	\	\	\	\	\	\	\	\	\
TCL-body sensor	\	\	\	\	\	\	\	\	\	\	\	\

## References

1. Internet of things security and privacy recommendations (2016). [http://www.bitag.org/documents/BITAG\\_Report\\_-\\_Internet\\_of\\_Things\\_\(IoT\)\\_Security\\_and\\_Privacy\\_Recommendations.pdf](http://www.bitag.org/documents/BITAG_Report_-_Internet_of_Things_(IoT)_Security_and_Privacy_Recommendations.pdf) (2016)
2. IoT malware dataset (2018). <https://www.stratosphereips.org/datasets-iot>
3. The ransomware tsunami (2019). <https://www.pandasecurity.com/en/mediacenter/security/2019-the-ransomware-tsunami/>
4. AIoT (2020). [http://report.iresearch.cn/report\\_pdf.aspx?id=3529](http://report.iresearch.cn/report_pdf.aspx?id=3529)

5. IoT forecast: connections, revenue and technology trends 2020–2029 (2021). <https://www.analysys.com/research/content/regional-forecasts-/iot-worldwide-forecast-rdme0>
6. Antonakakis, M., et al.: Understanding the Mirai botnet. In: 26th USENIX Security Symposium, USENIX Security (2017)
7. Bertsekas, D.P., Gallager, R.G.: Data Networks. Prentice Hall, Hoboken (1992)
8. Bezerra, V., Turrisi da Costa, V., Martins, R., Barbon, S., Miani, R., Bogaz Zarpelo, B.: Providing IoT host-based datasets for intrusion detection research. In: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) (2018)
9. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: methods, systems and tools. *IEEE Commun. Surv. Tutorials* **16**(1), 303–336 (2014)
10. Dai, S., Tongaonkar, A., Wang, X., Nucci, A., Song, D.: NetworkProfiler: towards automatic fingerprinting of Android apps. In: Proceedings of the IEEE INFOCOM (2013)
11. Ding, F.: IoT malware (2017). <https://github.com/ifding/iot-malware>
12. Doshi, R., Apthorpe, N.J., Feamster, N.: Machine learning DDoS detection for consumer Internet of Things devices. In: 2018 IEEE Security and Privacy Workshops, SP Workshops (2018)
13. van Ede, T., et al.: FlowPrint: semi-supervised mobile-app fingerprinting on encrypted network traffic. In: 27th Annual Network and Distributed System Security Symposium, NDSS (2020)
14. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th USENIX Security Symposium (2008)
15. Hamza, A., Ranathunga, D., Gharakheili, H.H., Roughan, M., Sivaraman, V.: Clear as MUD: generating, validating and applying IoT behavioral profiles. In: Proceedings of the 2018 Workshop on IoT Security and Privacy, IoT S&P@SIGCOMM (2018)
16. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall, Hoboken (1988)
17. Kambourakis, G., Koliass, C., Stavrou, A.: The Mirai botnet and the IoT zombie armies. In: 2017 IEEE Military Communications Conference, MILCOM (2017)
18. Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., Inman, D.J.: 1D convolutional neural networks and applications: a survey. *CoRR* (2019)
19. Koroniotis, N., Moustafa, N., Sitnikova, E., Turnbull, B.: Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Future Gener. Comput. Syst.* **100**, 779–796 (2019)
20. Li, R., Li, Q., Zhou, J., Jiang, Y.: ADRIoT: an edge-assisted anomaly detection framework against IoT-based network attacks. *IEEE Internet Things J.* **9**(13), 10576–10587 (2022)
21. Liu, F.T., Ting, K.M., Zhou, Z.: Isolation forest. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM) (2008)
22. Ma, X., Qu, J., Li, J., Lui, J.C.S., Li, Z., Guan, X.: Pinpointing hidden IoT devices via spatial-temporal traffic fingerprinting. In: 39th IEEE Conference on Computer Communications, INFOCOM (2020)
23. Marín, G., Casas, P., Capdehourat, G.: Deep in the dark - deep learning-based malware traffic detection without expert knowledge. In: 2019 IEEE Security and Privacy Workshops, SP Workshops (2019)
24. Marzano, A., et al.: The evolution of Bashlite and Mirai IoT botnets. In: 2018 IEEE Symposium on Computers and Communications, ISCC (2018)

25. McDermott, C.D., Majdani, F., Petrovski, A.: Botnet detection in the internet of things using deep learning approaches. In: 2018 International Joint Conference on Neural Networks, IJCNN (2018)
26. Miettinen, M., Marchal, S., Hafeez, I., Asokan, N., Sadeghi, A., Tarkoma, S.: IoT SENTINEL: automated device-type identification for security enforcement in IoT. In: 37th IEEE International Conference on Distributed Computing Systems, ICDCS (2017)
27. Mimoso, M.: New IoT botnet malware borrows from Mirai (2016). <https://threatpost.com/new-iot-botnet-malware-borrows-from-mirai/121705>
28. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. In: 25th Annual Network and Distributed System Security Symposium, NDSS (2018)
29. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI (2010)
30. Ren, J., Dubois, D.J., Choffnes, D.R., Mandalari, A.M., Kolcun, R., Haddadi, H.: Information exposure from consumer IoT devices: a multidimensional, network-informed measurement approach. In: Proceedings of the Internet Measurement Conference, IMC (2019)
31. Singh, A., et al.: HANZO: collaborative network defense for connected things. In: 2018 Principles, Systems and Applications of IP Telecommunications, IPTComm (2018)
32. Sivanathan, A., et al.: Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Trans. Mob. Comput.* **18**, 1745–1759 (2019)
33. Tang, R., et al.: ZeroWall: detecting zero-day web attacks through encoder-decoder recurrent neural networks. In: 39th IEEE Conference on Computer Communications, INFOCOM (2020)
34. Trimananda, R., Varmarken, J., Markopoulou, A., Demsky, B.: Packet-level signatures for smart home devices. In: 27th Annual Network and Distributed System Security Symposium, NDSS (2020)
35. Usama, M., Asim, M., Latif, S., Qadir, J., Al-Fuqaha, A.I.: Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. In: 15th International Wireless Communications & Mobile Computing Conference, IWCMC (2019)
36. Venkatesh, G.K., Anitha, R.: Botnet detection via mining of traffic flow characteristics. *Comput. Electr. Eng.* **50**, 91–101 (2016)
37. Wan, Y., Xu, K., Xue, G., Wang, F.: IoTArgos: a multi-layer security monitoring system for Internet-of-Things in smart homes. In: 39th IEEE Conference on Computer Communications, INFOCOM (2020)
38. Yao, H., Ranjan, G., Tongaonkar, A., Liao, Y., Mao, Z.M.: SAMPLES: self adaptive mining of persistent lexical snippets for classifying mobile application traffic. In: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom (2015)