IoT-KEEPER

Detecting Malicious IoT Network Activity Using Online Traffic Analysis at the Edge

Hafeez, Ibbad; Antikainen, Markku; Ding, Aaron Yi; Tarkoma, Sasu

DOI
10.1109/TNSM.2020.2966951

Publication date
2020

Document Version
Accepted author manuscript

Published in
IEEE Transactions on Network and Service Management

IoT-KEEPER: Detecting Malicious IoT Network Activity using Online Traffic Analysis at the Edge

Ibbad Hafeez*, Markku Antikainen‡, Aaron Yi Ding†, Sasu Tarkoma*
*University of Helsinki, Finland, ‡Aalto University, Finland, †Delft University of Technology, Netherlands

***Abstract*—IoT devices are notoriously vulnerable even to trivial attacks and can be easily compromised. In addition, resource constraints and heterogeneity of IoT devices make it impractical to secure IoT installations using traditional endpoint and network security solutions. To address this problem, we present IOT-KEEPER, a lightweight system which secures the communication of IoT. IOT-KEEPER uses our proposed anomaly detection technique to perform traffic analysis at edge gateways. It uses a combination of fuzzy C-means clustering and fuzzy interpolation scheme to analyze network traffic and detect malicious network activity. Once malicious activity is detected, IOT-KEEPER automatically enforces network access restrictions against IoT device generating this activity, and prevents it from attacking other devices or services. We have evaluated IOT-KEEPER using a comprehensive dataset, collected from a real-world testbed, containing popular IoT devices. Using this dataset, our proposed technique achieved high accuracy (≈ 0.98) and low false positive rate (≈ 0.02) for detecting malicious network activity. Our evaluation also shows that IOT-KEEPER has low resource footprint, and it can detect and mitigate various network attacks—without requiring explicit attack signatures or sophisticated hardware.**

***Index Terms*—IoT, Network, Security, Privacy, Activity Detection, Anomaly Detection, Traffic Classification**

I. INTRODUCTION

IoT-enabled automation systems have opened homes and industrial environments to countless new threats [1], [2]. There are several reasons for the sad state of IoT device security. IoT development teams often work without sufficient resources and under strict time constraints. These factors make it tempting for development team to cut corners, for example, by re-using unverified code snippets [3], insecure third-party libraries [4], and not following secure software development practices [5]. These, and several other factors, result in production of inherently vulnerable IoT devices for consumer markets.

Due to prevalence of insecure IoT devices, network owners can no longer rely on the assumption that all devices in their network are well-behaving and trustworthy [6]. While this, to some extent, applies to every network, it is a particular concern in small office, home office (SOHO) environments where the network owners do not have the know-how or resources to improve security. This, together with the fact that IoT devices are rarely updated [7], makes it probable that some devices in the network will, eventually, get compromised by an attacker.

The number of device specific exploits is constantly increasing due to growing number of IoT installations. Attackers can also re-use existing exploits, from PC-platforms, against IoT devices running a stripped down Linux [8] or Windows [9] as device firmware. On several occasions, attackers have been able to compromise IoT devices installed deep inside SOHO networks, to launch extremely large scale attacks [10], [11] as these devices have no security in place except for the network address translation (NAT), which is done on the gateway.

To address the sorry state of IoT security, our goal in this paper is to develop a system capable of securing the communication of IoT in edge networks. Such a system should be able to detect and isolate malicious IoT devices, with high sensitivity and minimal false alarms. This system should also be lightweight enough to operate efficiently using limited resources available at network gateways, typically used to set up edge networks.

We propose IOT-KEEPER, an edge system capable of performing online traffic classification at network gateways. IOT-KEEPER uses fuzzy C-Means clustering to differentiate between network traffic generated by IoT devices—in different modes of operation. It then uses fuzzy interpolation scheme to classify whether a given traffic flow belongs to malicious or benign network activity. Once an IoT device is identified as source of malicious activity, IOT-KEEPER uses *adhoc overlay networks* to restrict this device's network access such that it can perform normal operation, for example, communicate with respective cloud service, but can not perform network attacks against other device or services.

Given the challenges of collecting labeled traffic data, IOT-KEEPER uses unlabeled traffic data for model training. It uses a custom feature analysis technique to extract the set of features used for traffic classification. Once the classification model is trained, it can be represented as a set of rules. This representation enables sharing of trained classification models among multiple deployments of IOT-KEEPER, which reduces the time required to train initial classification model at new deployments, and improves scalability of system.

We made a prototype implementation of IOT-KEEPER to demonstrate how a simple yet efficient classification algorithm, combined with sophisticated feature analysis, enables us to perform real-time traffic classification, using only limited resources, available on a typical network gateway. It also shows that our classification technique is robust and achieves good classification performance in both closed-world and open-world scenario. More specifically, *our contributions are*:

- We design and implement a robust traffic classification technique for online detection of malicious network activity of IoT devices. We also present a detailed study of individual features and their relative importance to formulate a set of most useful features for network traffic classification problem.

- We present IoT-KEEPER, a hardware-agnostic gateway capable of detecting malicious network activity in real-time. IoT-KEEPER uses *adhoc overlay networks*—a novel mechanism to dynamically restrict network access for IoT devices, depending on their network activity.
- We study the performance of IoT-KEEPER in closed-world and open-world settings. Our evaluation shows that IoT-KEEPER achieves high detection accuracy (0.982), raises few false alarms (0.01), with low resource footprint, and minimal impact on latency experienced by users (increment $\approx 1.8\%$).
- We provide a comprehensive dataset for IoT traffic analysis research [12]. This dataset contains traffic traces collected from IoT devices commonly found in edge networks.

II. Challenges in Securing IoT

Conventional network and endpoint security solutions fall short in addressing the challenges of securing IoT ecosystem for a number of reasons.

C1 – Limited support for endpoint security solutions: Due to large diversity in IoT devices' firmwares, software stacks, and APIs, it is challenging to develop generic endpoint security solutions for IoT. Most IoT devices also lack the resources required to operate such endpoint security solutions. IoT-KEEPER does not require an agent, running on end hosts, to detect malicious behavior. Instead, it analyzes network traffic to identify malicious behavior of IoT devices. Since IoT devices are designed to use cloud services for majority of their operations, network activity of IoT devices gives fairly accurate representation of device state. This method is also device-agnostic as traffic footprint of malicious activity, such as port scanning attack, is almost similar, irrespective of the device used to perform the attack.

C2 – Securing device-to-device (D2D) communications: Traditional network security solutions are installed at network perimeter – where they only monitor network traffic entering or leaving the network. As a result, these systems do not secure D2D communications happening within local network. IoT-KEEPER is designed to operate at network gateways and access points, where it monitors D2D traffic within local network, as well as device-to-infrastructure (D2I) traffic between IoT devices and cloud services.

C3 – Diversity of IoT devices: Many traditional network security solutions rely on traffic signatures for anomaly detection. Due to large diversity of IoT devices, it is practically infeasible to obtain enough labeled data from IoT devices to generate these signatures [13], [14]. To address this limitation, IoT-KEEPER uses an unsupervised learning algorithm, which does not require labeled training data or device-specific anomaly detection model to perform traffic classification.

C4 – Deployment and operational costs: High deployment and operational costs are also limiting factors in using traditional network security solutions for securing edge networks [15], [14]. In comparison, IoT-KEEPER has lower deployment costs as it can be deployed using low-cost devices, with limited resources. It also has lower operational costs because the traffic filtering mechanism is automatically updated based on network activity of user devices.

C5 – Privacy and performance: To ensure privacy of user-data and good performance (in terms of latency), traffic classification tasks need to be performed at the edge. Due to large resource footprint [16], [17] and special hardware requirements [14], [15], use of existing traffic analysis methods is not scalable for securing edge networks. IoT-KEEPER performs traffic analysis at edge network gateways, using a lightweight traffic classification technique, to ensure privacy of user-data and low latency.

C6 – Ease of management: It is well known that due to poorly designed interfaces and lack of support for automated configuration, edge networks are rarely configured by users [14], [18], [19]. As a result, most edge networks are vulnerable to external attacks due to improper configurations. *Adhoc overlay networks*—introduced in this paper—resolve this issue by automating security policy management. It enables the gateway to apply appropriate network access control for connected devices, depending on their network activity.

III. Related Work

A number of techniques have been proposed to identify anomalies in network traffic [20], [21], [22], [23]. Researchers have studied various feature analysis and machine learning techniques to identify anomalies in network traffic; due by botnet activity [24], denial-of-service [25] and other attacks [21]. Typically, anomaly detection techniques can be divided into two categories: *offline* and *online* analysis technique.

Offline analysis techniques: These techniques use labeled data for model training and the classification model is obtained after several iterations of training and evaluation, using complete training dataset. These techniques have high resource footprint because model training process consumes lots of time and resources. Most anomaly detection techniques perform offline analysis and require labeled data [20], [26]. Given the diversity of IoT devices, it is challenging to collect sufficient labeled data for model training. Using crowd-sourcing model for collecting training data has its own limitations [17], [27].

Signature-based solutions are another example of offline analysis techniques. In addition to anomaly signatures, these solutions may also require custom hardware, and have high deployment and operational costs. Meanwhile, their performance is limited by the quality and volume of data available for generating anomaly signatures.

Recently proposed anomaly detection techniques use *recurrent neural networks* [28], [29] and *gated recurrent units* [17], [30]. These techniques model network traffic as *symbols* in a language and use a frequency based model to identify anomalous sequence of symbols—indicating network attacks [31], [17]. Such techniques have prohibitively high resource footprint and can not be deployed using edge network gateways.

Online analysis techniques: These techniques do not have access to complete dataset during training phase, and they mostly use unlabeled data to train classification model. Ideally, online techniques should be efficient enough to ensure high detection accuracy, at high packet arrival rates, using limited

resource. Among online traffic analysis techniques, Securebox was, to the authors' knowledge, the first to propose a two tier model, where a lightweight network gateway uses a cloud service to analyze network traffic [19]. Securebox suffers from privacy and latency concerns as traffic analysis is performed in a remote environment. IoT-KEEPER addresses these concerns by performing traffic analysis locally at network gateways.

IoT Sentinel [18] uses traffic traces from IoT devices, captured during device setup, to identify device model and manufacturer information. This information is later used to set up (one time) network access control for identified devices. Since IoT Sentinel uses traffic traces collected during device setup, it can not identify (and set up access control) for IoT devices which have already been taken into use. AuDI [16] has been recently proposed to identify device-type information by analyzing the packet timing information from IoT traffic.

Kitsune [32] is a lightweight, online anomaly detection technique, which uses an ensemble of autoencoders for anomaly detection, whereas DIöT [17] uses the periodicity in IoT device traffic and device-specific anomaly models to detect malicious network activity.

IoT anomaly detection techniques mostly detect volumetric attacks that produce large volume of network traffic, such as Mirai [17]. However, these techniques fall short in detecting attacks with sporadic network activity, such as Man-in-the-Middle (MitM). Such techniques also use default behavior of IoT devices as normal behavior and assume that all devices are inherently benign [17], [18], [32]. Hence, they are unable to detect malicious behavior of an IoT device that is inherently compromised. Since our classification technique is not device dependent, IoT-KEEPER can detect discrepancies in a devices' "normal" network behavior and flag it as anomalous activity.

IV. THREAT MODEL

Edge networks typically contain a mixture of IoT and PC-like devices, such as smartphones and tablet computers. With IoT devices, the assumption about implicit trustworthiness of connected devices does not hold, because it is fairly easy to exploit vulnerable IoT devices [33], [34], and use them to attack other devices in the network.

This section discusses some of the attacks commonly observed in edge networks. Modern IoT malware, such as Mirai [35], can use a combination of these attacks to achieve desired results.

T1 – Network scanning: These attacks are commonly used to scan target nodes before launching dedicated attacks against the scanned targets. An attacker can use scanning attacks to recognize TCP and UDP services running on target hosts, detect firmware/operating-system version on target device, and identify what kind of traffic filtering is being performed in the network. In this paper, we consider three variants of network scanning attack, namely, *address-sweep*, *port-sweep*, and *port-scan* attacks.

T2 – Vulnerability scanning: These attacks are designed to catalogue vulnerabilities in target devices based on their software version and open services. This information can be used to perform targeted attacks against target hosts.

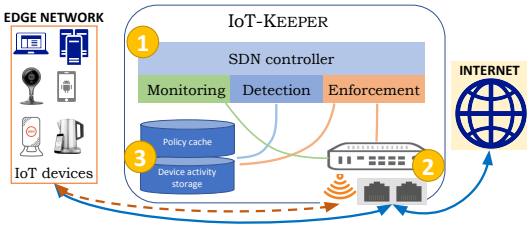


Fig. 1: IoT-KEEPER architecture. The controller (1) is responsible for traffic monitoring, anomaly detection, and policy enforcement, managing OF switch (2), and maintaining cache (3) for security policies and device activity

T3 – Man-In-the-Middle (MitM): MitM attacks are performed to snoop-in on network communication of user devices, and modify network traffic to perform injection and replay attacks. For example, an attacker can replay traffic, intercepted from users' smartphone to disable home security system without users knowledge.

T4 – Data theft: Health IoT, smart appliances, and similar devices collect a lot of data about their users. Typically, users do not have discrete control over how this data is collected and transmitted [36]. An attacker can compromise IoT devices to steal user data.

T5 – Botnets: Modern botnets are generally comprised of compromised devices deployed in edge networks [37], [26]. An infected device can also compromise and enroll other devices in the botnet. Distributed denial-of-services (DDoS) attacks are a common example of how seemingly benign user devices in edge networks can be used to launch large scale attacks [11], [35].

IoT-KEEPER focuses on detecting and blocking different variations of these attacks in edge networks.

V. SYSTEM ARCHITECTURE

In this section, we describe the internal architecture of IoT-KEEPER, as shown in Figure 1. Our proposed system design is agnostic of underlying hardware as it can be deployed using single board computers or consumer-grade edge network gateways, such as Linksys [38] or Netgear routers [39].

IoT-KEEPER uses SDN controller and Open vSwitch (OVS) [40] to monitor and analyze incoming traffic flows and perform traffic filtering. In our prototype implementation, both the controller and OVS run on same node. However, IoT-KEEPER architecture supports hierarchical deployments, where a single instance of IoT-KEEPER manages multiple OpenFlow-enabled switches in the network. In such case, a single controller can monitor, analyze and control traffic from multiple switches.

IoT-KEEPER operates three modules for *monitoring* network traffic, *analyzing and detecting* malicious traffic, and *enforcing* network access control for IoT devices in the network. When a new traffic flow is detected by monitoring module, it requests enforcement module to analyze given traffic flow. If there is a security policy available in cache, which matches given flow, it is used to setup flow table entries to allow or deny given flow. Otherwise, detection module will analyze the

given flow to detect if the flow is malicious or not. The result of analysis will be used to setup flow table entries in OVS, and stored in policy cache for later use.

We now discuss the internal design of each of these three modules in detail.

VI. Monitoring Module

This module is responsible for monitoring all traffic flows processed by the gateway. It maintains up-to-date information about traffic generated by devices connected to the network. This information is consumed by detection module to analyze how a devices' network behavior changed over time.

VII. Detection module

This module analyzes network traffic data to detect malicious activity in the network. The result of traffic analysis is used to generate security policies, which are then used by enforcement module to set up device-level network access control in the network.

In the following sections, we describe our proposed feature analysis and traffic classification technique, which enable IoT-Keeper to perform online traffic classification, with low resource footprint. The classification model trained using our proposed technique can be represented as a set of rules, and shared across multiple deployments of IoT-Keeper. This mechanism helps to improve classification performance of existing deployments and speed up traffic classification process on new deployments.

A. Feature Analysis

To make our traffic classification scheme device-agnostic and lightweight, we only extract features from traffic data observable on network link. We study the variance and modality of each feature to identify its contribution to classification process. Any feature that does not contribute to the clustering and classification process is removed from final feature set. This results in speeding up classification tasks and reducing the resource footprint of traffic classification process.

Table I lists the 38 attributes extracted from network metadata. We collect total number of observations (N) for all these features. We also compute sum of observations (S_o) and sum of squares of observations (S_{sq}), using Eq. 1, for *data* features. These statistics are collected for all traffic streams, and summarized for individual device using source and destination MAC, IP and ports.

$$\mu = S_o/N \qquad \sigma = \sqrt{\mid S_{sq}/N - (S_o/N)^2 \mid} \tag{1}$$

We study the variance for each feature using cumulative distribution function (CDF), as shown in Fig. 2. We observed that these feature distributions are not Gaussian but heavy tailed, where smaller values constitute majority of probability mass. For example, Fig. 2a shows that more than 70% IoT devices connect to fewer than 20 unique destination IPs, whereas only few devices (tail of distribution) may connect to more than 300 unique destinations. The data points comprising tail of distributions are of primary importance because they capture anomalous behavior of IoT devices, and this information is helpful in identifying malicious behavior.

TABLE I: Feature extracted from network traffic data

Type	Feature
Source, Destination	[Total, Unique] destination IP addresses
Connection counters	[Total, Unique] source ports, destination ports, connections, (same source, same destination, same service) connections, connection durations (binned)
Packet counters	ARP, LLC, IP(v6), ICMP(v6), EAPoL, TCP(v6), UDP(v6), HTTP, FTP, HTTPS, DHCP, (M)DNS, NTP, Router Alert, (SYN, REJ) (errors), Urgent, Padding
Data	Total data, source to destination (SRC2DST) data, destination to source (DST2SRC), packet size

The study of feature value distributions also reveals possible correlations among different features. For example, when an attacker performs scanning attack, both total number of connections initiated by attacker node, and number of connections between source (attacker) and destination (target) increases. Such correlations help us to identify and remove features containing redundant information.

We use *correlation-based feature selection* (CFS) to identify and remove any features which contain redundant information. We calculate Pearson correlation coefficient R to measure the dependencies among all features and discard one of any two features that are strongly correlated.

B. Clustering

We use fuzzy C-means (FCM) clustering algorithm [41] to partition the data points based on their mutual likeliness. During clustering, all data points $X_j \ (j = 1, 2, ..., n)$ are initially assigned a membership value (μ) for all clusters $C_i \ (i = 1, 2, ..., c)$. Each data point X_j is represented as $\left(f_j^{(1)}, f_j^{(2)}, ..., f_j^{(k)}, ..., f_j^{(h)}\right)$, where $f_j^{(k)}$ is value for k^{th} feature in X_j and $1 \leq k \leq n, \quad n = len(\vec{F})$.

The membership value for $X_j \in C_i$ is given as μ_{ij}, where $0 \leq \mu_{ij} \leq 1$, and $\sum_{i=1}^{c} \mu_{ij} = 1 \quad \forall\ 1 \leq i \leq c \wedge 1 \leq j \leq n$. The membership value μ_{ij} (Eq. 2) for each data points and cluster centers V_i (Eq. 3) for each cluster are optimized to minimize objective function given in Eq. 4, where m is fuzziness index [42] and $\left\|V_i - X_j\right\|$ is the Euclidean distance between cluster center V_i (for cluster C_i) and data point X_j. This algorithm is proven to converge to local minimum or saddle point of objective function with linear rate of convergence [43], [44], [45].

$$\mu_{ij} = \left(\sum_{d=1}^{c} \left(\frac{\left\|V_i - X_j\right\|}{\left\|V_d - X_j\right\|} \right)^{\frac{2}{m-1}} \right)^{-1}, \ {}^{1 \leq i \leq c}_{1 \leq j \leq n} \tag{2}$$

$$V_i = \frac{\sum_{j=1}^{n} (\mu_{ij})^m \times X_j}{\sum_{j=1}^{n} (\mu_{ij})^m}; \ {}^{1 \leq i \leq c}_{1 \leq j \leq n} \tag{3}$$

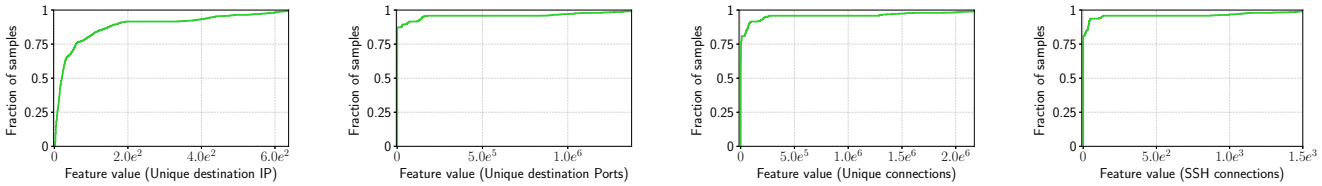


Fig. 2: Feature value distributions observed in network traffic of IoT devices, used in our testbed network.

$$J_m = \sum_{i=1}^{c}\sum_{j=1}^{n} \mu_{ij}^{m} \left\| V_i - X_j \right\|^2 \tag{4}$$

In order to make sure that no feature over-influences clustering, all features are normalized to range $[0, 1]$ before clustering. After clustering, normalized feature score are computed for each feature, in all clusters. Any features with same scores (within a defined tolerance) in multiple clusters, such as REJ errors, are considered non-contributing features and removed from the final feature set.

At the end of clustering, a label is assigned to each cluster based on normalized feature scores observed in the given cluster. These labels correspond to different types of benign and malicious traffic flows. Each cluster is represented as a rule, where feature scores represent antecedent variables (f^*) and cluster label is the consequent variable (y). This set of rules will be used by FIS to perform traffic classification.

These rules capture the patterns observed in network traffic used for training classification model. The patterns in network traffic can vary across networks due to several factors, such as number and types of connected devices, network configuration. By sharing these rules among multiple deployments, the classification performance of IoT-KEEPER can be improved. A new gateway can also use these rules to bootstrap traffic classification process.

C. Parameter Selection

The choice of number of clusters (i) can affect the performance of traffic classification technique. Therefore, we use both direct and statistical testing methods to choose an optimal value of i.

Initially, we use `NbClust` package [46] to compute 30 different indices for a range of possible values for i. We used *agglomeration* method for cluster analysis using Wards' linkage method and *euclidean* distance metric. Figure 3a shows the number of votes (minimum 3 votes) received by different values of i, where one vote represents that one of the 30 indices suggests that the given value of i is an optimal choice. A detailed discussion on the indices computed by `NbClust` package is out of scope for this paper.

Based on the voting results of `NbClust`, we select top eight candidate values of i and analyze them using *elbow method* and *average silhouette heuristic* [47] to get a measure of global clustering characteristic. For elbow method, within-cluster-sum-of-distances (WCSD) is calculated using Eq. 5, where c is the number of clusters, S_i is the set of data points belonging to i^{th} cluster, and x_{ki} is the k^{th} variable of V_i.

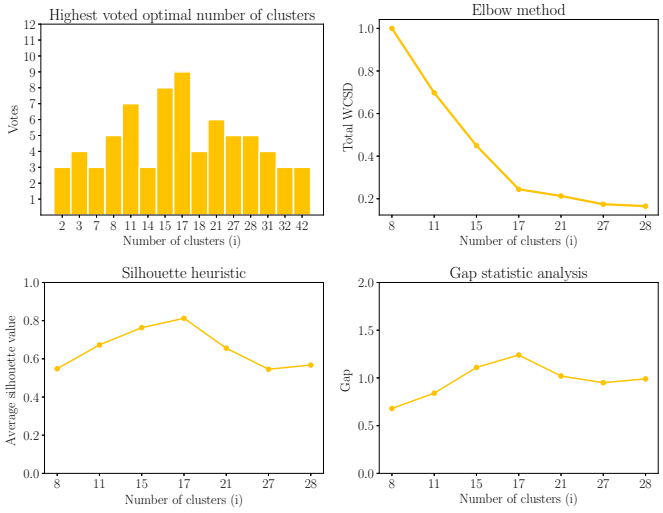


Fig. 3: The top 8 highest voted values of i (from `NbClust`) voting are analyzed using elbow method, average silhouette heuristics, and gap statistics to find the optimal value of i.

$$WCSD = \sum_{i=1}^{c}\sum_{j\in S_i}\sum_{k=1}^{p} \|x_{ki} - x_i\| \tag{5}$$

Silhouette heuristics are calculated using Eq. 6, where $a(x) = \frac{1}{k}\sum_{j=1}^{k}\left\|x - p_j\right\|$, $p_j \in C_i \wedge x \in C_i$. Similarly, $b(x) = \frac{1}{k}\sum_{j=1}^{k}\|p_k - x\|$, where $p_k \in C_i'$ and C_i' is the closest neighboring cluster for x such that $C_i' = C_i \in C$ with $\min(\|x - V_i\|) \ \forall \ C_i \in C \wedge x \notin C_i$. Figure 3 shows that both elbow and silhouette method suggest $i = 17$ as optimal number of clusters.

$$s(x) = \frac{(b(x) - a(x))}{\max\left(a(x), b(x)\right)} \tag{6}$$

We also studied *gap statistic method* [48] to get a statistical formulation of WCSD and silhouette statistics. In general, the optimal value for i should maximize gap statistic as well as silhouette values, while minimizing WCSD. Using 1-standard-error method [48], gap statistics analysis suggests $i = 17$ as optimal number of clusters for given scenario. Our evaluation shows that this value of i works effectively (to identify attacks discussed in this paper) irrespective of the dataset.

D. Anomaly detection

We use fuzzy interpolation scheme [49], [41] (FIS) to classify whether a given traffic flow is malicious or benign. FIS

uses the sparse fuzzy rule base, consisting of n rules ($n = c$), obtained from clustering, to detect malicious traffic flows. This set of rules can be represented as.

$$\begin{array}{c} \textit{Rule 1: if } f_1 \in A_{11}, f_2 \in A_{21}, \ldots, f_k \in A_{k1}, \ldots, f_h \in A_{h1} \Longrightarrow y \in O_1 \\ \textit{Rule 2: if } f_1 \in A_{12}, f_2 \in A_{22}, \ldots, f_k \in A_{k2}, \ldots, f_h \in A_{h2} \Longrightarrow y \in O_2 \\ \vdots \\ \textit{Rule Q: if } f_1 \in A_{1q}, f_2 \in A_{2q}, \ldots, f_k \in A_{kq}, \ldots, f_h \in A_{hq} \Longrightarrow y \in O_q \\ \hline \textit{Observation: } f_1 \in A_1^*, f_2 \in A_2^*, \ldots, f_k \in A_k^*, \ldots, f_h \in A_h^* \ , \\ \textit{Conclusion: } y = O^* \end{array}$$

where R_i ($1 \leq i \leq Q$) is i^{th} rule generated from cluster C_i. A_{ki} and O_i are triangular fuzzy sets for k^{th} antecedent feature $f_k, 1 \leq k \leq h$ and consequent variable y respectively. For a new observation, A_k^* and O^* are triangular fuzzy sets for antecedent and consequent variable obtained as a result of interpolation of spare fuzzy rule base.

A fuzzy triangular set A is represented using three characteristic points a, b, and c, where b is *center point* with maximum membership value and a, c are *left*, *right* points respectively, with minimum membership value [41]. The characteristic points a_{ki}, b_{ki}, c_{ki} for fuzzy set A_{ki} of k^{th} antecedent feature f_k in rule R_i are calculated as:

$$b_{ki} = f_q^{(k)}, \quad where\ \mu_{iq} = \max_{1 \leq j \leq n} \mu_{ji}, \tag{7}$$

$$a_{ki} = \frac{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \leq b_{ki}} \mu_{ij} \times f_j^{(k)}}{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \leq b_{ki}} \mu_{ij}} \tag{8}$$

$$c_{ki} = \frac{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \geq b_{ki}} \mu_{ij} \times f_j^{(k)}}{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \geq b_{ki}} \mu_{ij}} \tag{9}$$

where b_{ki} has membership value of 1 and a_{ki} and c_{ki} have membership value of 0. $f_j^{(k)}$ is the k^{th} feature's value in sample X_j with $1 \leq k \leq h$. The defuzzified value of a triangular set A is calculated as

$$D_f(A) = \frac{(a + 2 \times b + c)}{4} \tag{10}$$

The membership value for input feature $f_j^{(k)}$ is $\mu_{A_{k,i}}(f_j^{(k)})$, where $\min_{1 \leq k \leq h} \mu_{A_{k,i}}(f_j^{(k)}) > 0$, $1 \leq i \leq p$, and p is the number of activated fuzzy rules. The inferred output O_j^* based on fuzzy rules activated by $f_j^{(1)}, f_j^{(2)}, ..., f_j^{(h)} \in X_j$ is calculated as,

$$O_j^* = \frac{\sum_{i=1}^{p} \min_{1 \leq k \leq h} \mu_{A_{k,i}}(f_j^{(k)}) \times D_f(B_i)}{\sum_{i=1}^{p} \min_{1 \leq k \leq h} \mu_{A_{k,i}}(f_j^{(k)})} \tag{11}$$

$D_f(B_i)$ is defuzzified value for consequent fuzzy set, in R_i activated by X_j inputs and it can be calculated using Eq. 10. We calculate the weight W_i of activated rule R_i, such that $0 \leq W_i \leq 1$, $\sum_{i=1}^{c} W_i = 1$, on the basis of input observations $x_1 = f_j^{(1)}, x_2 = f_j^{2)}, ..., x_h = f_j^{(h)}$ as:

$$W_i = \left(\sum_{d=1}^{c} \left(\frac{\|r^* - r_i\|}{\|r^* - r_d\|} \right)^2 \right)^{-1}, \tag{12}$$

where r^* is the input feature vector $\left(f_j^{(1)}, f_j^{(2)}, ..., f_j^{(h)}\right)$ and r_j is set of defuzzified values of A_{ki} in R_i. $\left(D_f\left(A_{1,i}\right), D_f\left(A_{2,i}\right), ..., D_f\left(A_{h,i}\right)\right)$, $1 \leq k \leq h$. The final inferred output is calculated as

$$O_j^* = \sum_{i=1}^{c} W_i \times D_f\left(B_i\right) \tag{13}$$

In practice, clustering produces n clusters from the training dataset. Based on these clusters, we generate a set of rules containing r ($r = n = 17$ in this case) rules. For a new observation (traffic flow), each feature triggers some fuzzy rule from spare rule set. We predict the type of given observation by calculating the weighted sum of all activated rules. For example, if a traffic flow activates three fuzzy rules, two representing benign traffic and one representing malicious, the final prediction is made according to the weight of each rule.

VIII. Enforcement Module

IoT-Keeper includes an enforcement module, which automatically restricts network access of a device exhibiting malicious behavior. For example, if a smart bulb is performing network scanning attack, IoT-Keeper restricts its network access such that it can only communicate with its manufacturers' cloud service.

IoT-Keeper enforcement module sets up network access control using security policies. A *security policy* is generated based on the result of traffic analysis performed by detection module. It is used to generate flow table rules—deployed at OVS—for handling network traffic. During operation, security policies are stored in an in-memory *policy cache* with a predefined *time-to-live* (TTL). Every time a security policy is used to handle a traffic flow, its TTL is refreshed. If a security policy remains unused until TTL expires, it will be removed from cache. This mechanism ensures that majority of traffic flows are handled using security policies from cache (resulting in low latency overhead), while preventing cache size from growing too large. Policy cache is also backed up periodically to a file to persist after reboot. Since majority of IoT traffic is destined to a handful of cloud services [50], caching the security policies for such frequently visited cloud services reduces the number of traffic analysis operations. This reduces the resource footprint of detection module, minimizes (additional) latency experienced by users, and improves overall performance of IoT-Keeper.

A. Adhoc Overlay Networks

Adhoc overlay networks (AON) are a novel approach for blocking malicious activity of IoT devices—without severely

affecting user experience. These are virtual networks overlayed on same physical network, where each virtual network has restricted network access and contains one or more devices. Instead of completely blocking network access for a connected device, AONs allow us to restrict network access for IoT devices on local network and Internet.

IOT-KEEPER uses three types of AONs:

- *No Access*—where devices are completely blocked from connecting to the network.
- *Restricted Access*—where devices can communicate with other device(s) in the same (restricted) AON.
- *Safe/Full Access*—where devices have no restrictions on network access.

All AONs allow IoT devices to connect to their respective cloud services so that IoT devices can perform their basic operations. This minimal connectivity is specified based on the type and manufacturer of IoT device. The details of obtaining this information are discussed in our earlier work [18].

Although it is possible to run multiple networks with legacy gateways using VLANs or multiple SSIDs, there is only a limited number of VLANs and SSIDs supported by routers or access points available commercially. It is also difficult to automatically setup and manage VLANs. In case of multiple SSIDs, client devices need to (re)associate every time SSID is updated, resulting in bad user experience.

Since AONs operate at software layer, the number of AONs and their functionality is not limited by hardware support. They are easy to set up, tear down, and update. The network access for an AON is modified by updating security policies, and this process does not require action from connected devices. It is also possible to share the security policies (used to setup AONs) with other gateways to achieve consistent network access control across multiple networks.

IX. DATASET

This section presents the testbed and the process used to collect the dataset used for evaluation of IOT-KEEPER.

A. Limitations of existing IoT traffic datasets

We found various limitations in existing IoT traffic datasets [32], [18]. These datasets are collected for specific device activity, and do not capture network activity of IoT devices for D2D communication and user interactions. In addition, these datasets contain traffic traces of short duration, which do not capture the evolution of IoT devices' network behavior over a long period of time.

Most IoT datasets are collected in *closed-world* setting with all IoT devices in a dedicated (isolated) network, and no other device(s) connected to this network [18]. In contrast, real world edge networks (*open-world* setting) are expected to contain PC-like device(s)—generating large volume of traffic—connected to same network as IoT. Even if traffic analysis techniques use short duration traffic traces from IoT devices only, it is important to use realistic open-world setting to capture randomness in network traffic due to varying traffic loads.

IoT datasets generally overlook D2D communication between IoT and PC-like devices. However, this data is useful to detect attacks from malicious IoT, targeted at PC-like devices. In some cases where PC-like devices were used during data collection [32], their purpose was to introduce background traffic only, and the datasets did not study the interaction between IoT and PC-like devices.

To address these limitations, we have set up a real world test network to collect long term network traces. In this testbed, we also include PC-like devices to emulate a realistic edge network and collect data for D2D communication among IoT and PC-like devices.

B. Testbed Setup

Our testbed uses IOT-KEEPER as a network gateway. We use `tcpdump` to collect all incoming and outgoing traffic, from both wired and wireless interfaces on the gateway. In case an IoT device communicates to Internet via an IoT hub using Weave, ZigBee, or similar protocols, its D2I communication are monitored by collecting network traffic of IoT hub. We collect traffic data for both benign and malicious network activity, where malicious activity is generated using Raspberry Pis and Mirai-infected IoT devices. We configure the gateway to drop all unfiltered outgoing traffic to prevent spread of malicious traffic on public Internet.

C. Data collection

We divide data collection process in three phases:

Device setup: When an IoT device is being set up and it connects to network for the first time, we collect all traffic generated by IoT device itself and the device used to set it up, such as a smart phone. During data collection, IoT device(s) were reset and booted up from factory default state prior to every setup.

Device background activity: For background activity, we collect the traffic generated by an IoT device during its normal operation, including the time when it connects or disconnects from the network. For this purpose, we use an already set up IoT device (i.e., setup phase traffic not included) and leave it in connected state for a given time interval—ranging from 10 minutes to 72 hours No explicit user interaction is performed with the IoT device during this time. The background activity may vary with the kind of device, for example, single-purpose IoT devices may only generate heartbeat or status update messages, whereas multi-purpose IoT devices may periodically fetch application updates, generate notifications, and communicate with third party services.

Device activity: We also collect the traffic from IoT devices when they are communicating with other devices (D2D) or cloud services (D2I). This traffic is generated as a result of device-to-device or user-to-device interactions. The network activity during this phase also varies with the functionality supported by the device. For example, smart power plugs only support on/off functions, whereas a security camera allows user to toggle video feed, video quality, and motion detection.

To collect IoT device activity traffic, we use an already set up IoT device. For benign activity, a user repeatedly interacts

Classification	Threats	Activity	Tool	Description
Scanning	T1	Port Scan	ZenMap, NMap	Scanning network for open ports on different hosts in the network
	T1	Port Sweep	ZenMap, NMap	Scanning all TCP/UDP ports on one or more target hosts
	T1	Address sweep	ARPing, ARP scan, Skipfish	Scanning all hosts on the network and service running on them
Vulnerability scanning	T1, T2	OS Scan, Password attacks	Nmap, WFuzz, Brutus, Python	Discovering devices and their operating systems, guessing password for open services
Botnet	T1, T4, T5	Mirai	Telnet	Find and infect devices by deploying Mirai malware
MitM	T3	ARP Poisoning	ARPspoof, EtterCap	Using ARP poisoning attack to capture all LAN traffic
Data theft	T4	Data hijacking	Telnet	Gain privileged access to other hosts and download collected data.
Malware	T4, T5	Malware injection	Metasploit	Upload malware to target hosts
Denial of Service	T1, T5	SYN Flooding	Python scapy, Hyenae	Flood the target host with many SYN requests to block it from performing any other task
	T3, T5	SSL renegotiation	tls-dos	Flood the target with SSL renegotiation packets to disable its packet stream

TABLE II: Types of network attacks executed by compromised and malicious devices.

with IoT device over a period of time, in no specific order, with irregular wait intervals between repetitions. For malicious traffic, we use Mirai infected IoT devices and Raspberry Pis. During data collection, the network setup was reset after each iteration to recover virgin state before subsequent iteration. Table II gives a high level classification of IoT attacks considered in this paper, and the tools used to simulate these attacks [51], [52], [53].

X. Evaluation

A. Implementation

Our prototype implementation of IoT-Keeper uses a Raspberry Pi set up as a WiFi access point using `hostapd` module [54]. It runs a DHCP server and manages NAT for both wired and wireless network. IoT and user devices can be connected to this access point via wired or wireless network.

The SDN controller used by IoT-Keeper prototype is based on Floodlight [55], where we have implemented additional modules for traffic monitoring, traffic filtering, state management, security policy enforcement, and cache management. All wired and wireless interfaces on IoT-Keeper are bridged to an OVS, managed by the SDN controller. With this setting, IoT-Keeper is able to monitor and restrict D2D traffic among devices connected to same SSID. The feature engineering and anomaly detection schemes are implemented using Python libraries. The Rest-API used to manage gateway functionality is implemented as part of SDN controller.

We use this prototype to evaluate IoT-Keeper performance in both closed-world and open-world settings. In closed-world setting, we have complete information about IoT devices' functions and attack mechanisms. The open-world setting provides minimal prior information about IoT devices and attacks, as it uses different attack nodes and attack mechanism compared to the ones used for collecting training data.

B. Anomaly Detection

We study the performance of proposed traffic classification technique in terms of *true positive rate* (TPR), *false positive rate* (FPR), and F1-score. TPR gives a measure of reliability of correctly identifying the type of traffic flows, and FPR gives an estimate of false alarms (benign activity classified as malicious). Meanwhile, *false negative rate* (FNR) explains what percentage of malicious traffic was not detected correctly.

There is a trade-off between FPR and FNR. In some cases, low FPR may be preferred as it improves user experience by preventing false alarms. However, highly sensitive installations require low FNR, so that no malicious traffic goes undetected. Using IoT-Keeper, *false positives* do not significantly impact user experience because IoT-Keeper only blocks malicious activity and maintains minimal network access for IoT devices, so that they can perform their normal operation. Therefore, IoT-Keeper can achieve lower FNR, for better security, without affecting user experience, because any false positives will not block network access for benign IoT devices.

We divide device activity recognition problem in two sub-problems:

- *Binary-class problem*—differentiates between benign and malicious network activity.
- *Multi-class problem*—detects the type of malicious activity exhibited by the device.

After detecting some malicious activity (binary-class problem), we detect the *type* of given malicious activity. This provides us with more information that is used to enforce appropriate network restrictions against the device generating this activity. For example, an IoT device executing network scanning attack should be allowed to only access its respective cloud service, whereas an IoT device stealing user data should be completely blocked from accessing the network.

To evaluate classification performance, we first study the overall accuracy achieved by IoT-Keeper for binary-class and multi-class problem, in closed-world (CW) and open-world (OW) settings. Table III shows that IoT-Keeper achieves better accuracy for activity recognition in closed-world setting. The difference in TPR is more prominent for multi-class problem because the number of classes (types of attacks) is higher (10 versus 2), for multi-class problem, and some attacks are misclassified as another attack. Later in this section, we describe how this lower TPR does not affect the security guarantee of IoT-Keeper. Meanwhile, FNR for binary-class problem in open-world setting is 0.04 compared to 0.02 for closed-world setting. It shows that our technique

	Binary class		Multi class	
	Closed world	Open world	Closed world	Open world
TPR	0.98	0.96	0.95	0.93
FPR	0.06	0.09	0.20	0.29
F1-score	0.97	0.95	0.92	0.87

TABLE III: Results for binary-class and multi-class problem in open and closed world setting

captures salient features of malicious activity, and is able to detect variations of network attacks in open-world setting.

Table IV shows that IoT-KEEPER achieves good performance in identifying different types of network attacks. It should be noted that Tab. IV only presents the results for distinguishing between different types of malicious activity, and benign activity is not discussed here. Therefore, any misclassifications only mean that one type of attack is detected as another type of attack. As discussed in Sect. VIII, IoT-KEEPER restricts network access for a device generating malicious activity. Hence, these misclassifications do not undermine the security provided by IoT-KEEPER because it can distinguish malicious activity from benign activity with high accuracy (see Tab. III).

		TPR	FPR	FNR	F1
Port scan	CW	0.95	0.15	0.05	0.93
	OW	0.91	0.24	0.09	0.88
Port sweep	CW	0.98	0.16	0.02	0.93
	OW	0.97	0.23	0.02	0.89
Address sweep	CW	0.98	0.19	0.02	0.93
	OW	0.97	0.31	0.03	0.89
Botnet	CW	0.98	0.08	0.01	0.97
	OW	0.97	0.18	0.03	0.95
Data theft	CW	0.88	0.48	0.13	0.79
	OW	0.85	0.52	0.15	0.76
Malware injection	CW	0.88	0.41	0.12	0.85
	OW	0.85	0.48	0.15	0.81
DoS	CW	0.97	0.07	0.04	0.93
	OW	0.92	0.15	0.08	0.86

TABLE IV: Performance of IoT-KEEPER for identifying different types of attacks.

Although we achieve high detection rate for network scanning attacks—due to substantially different traffic footprint compared to normal IoT traffic—we observed lower TPR and high FPR when distinguishing between different variations of scanning attacks, such as *port scan*, *port sweep*.

To investigate this discrepancy, we studied the feature value distributions in network traffic generated by these attacks. In case two attacks have similar traffic footprint, their feature value distributions will be overlapping. This results in misclassification of one attack as another. This phenomenon is prominent for variants of network scanning attacks. For example, *port scan* attacks were classified as *port sweep* because both attacks open large number of connections between source and target nodes, resulting in similar traffic footprint. Since the network restrictions for a device performing any type of network scanning attack are nearly similar, resulting security implication of these misclassifications is negligible in this case.

	Closed world	Open world
TPR	0.98	0.96
FPR	0.04	0.1
F1	0.98	0.93

TABLE V: Performance of IoT-KEEPER when port scan, port sweep and address sweep attacks are considered together as network scanning attacks.

IoT anomaly detection techniques register deviations from *normal* network behavior as *malicious*. In case of volumetric attacks, such as network scanning and botnet activity, these deviations are clear because such attacks have voluminous network activity. On the other hand, MitM and data theft attacks have sporadic network activity, similar to normal IoT device activity. Therefore, it is difficult to detect such attacks with low network activity, without additional information. However, IoT-KEEPER achieves good performance ($accuracy \approx 0.74$) in detecting these attacks, which often go undetected by anomaly detection systems.

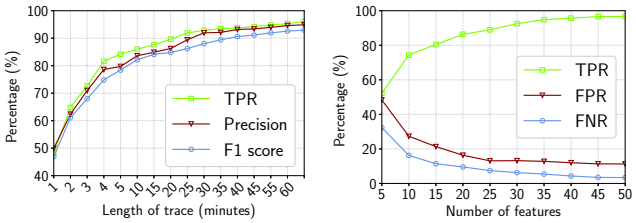
To assess the robustness of our classification technique in open-world setting, we test IoT-KEEPER in scenario where classification model is trained in one network, and is used in another network to perform traffic classification. For this purpose, we use publicly available IoT datasets to represent different networks. The first dataset, YTY2018, reported by Yisroel *et al.* [32], contains traffic traces for similar attacks as listed in Tab. II. These network traces have been collected from connected security cameras and digital video recorders. Yisroel *et al.* used PC and IoT devices in their testbed to have noise in network, but did not investigate D2D communication among PC, IoT and security cameras [32]. MSI2017 dataset, reported by Miettinen *et al.* [18], contains traffic traces collected from IoT devices. Some of the IoT devices used in MSI2017 are also included in our testbed. This dataset does not contain traffic traces from PC-like devices.

In first experiment, we simulate closed-world (CW) setting by using data from our testbed (Keeper dataset) for model training, and YTY2018 for model testing. Open-world setting uses YTY2018 for model training and Keeper dataset for testing. The results reported in Tab. VI shows that our classification scheme is invariant of the data source used for model training. We achieved better performance for CW setting because Keeper dataset—used for model training—contains traces for most attacks included in YTY2018. Compared to Keeper dataset, YTY2018 has less diversity in attack traffic traces, as this dataset was collected using IP cameras only. However, we achieved good performance when model was trained using YTY2018 (open-world setting). This shows that our classification technique is independent of the type of devices used for training data collection. Therefore, we can train our model using traffic traces collected from different networks, irrespective of the devices connected to that network.

Further investigation (Fig. 5a) showed that the number of IoT devices used for collecting training data does not significantly affect the performance of IoT-KEEPER because most attacks have similar network activity irrespective of the device used to perform these attacks. On the other hand, using dif-

		TPR	FPR	F1
Network scanning	CW	0.98	0.07	0.96
	OW	0.95	0.1	0.93
DoS	CW	0.96	0.02	0.93
	OW	0.93	0.3	0.88
MitM	CW	0.92	0.03	0.84
	OW	0.88	0.4	0.78
Botnet	CW	0.99	0.08	0.98
	OW	0.97	0.2	0.95

TABLE VI: Classification performance using YTY2018 dataset.



(a) (b)

Fig. 4: IoT-KEEPER classification performance using (a) different length traffic traces (b) different combination of features, for model training.

ferent types of IoT devices improves detection performance—especially for attacks with low network activity—as it helps the system to learn variations of IoT devices' network activity.

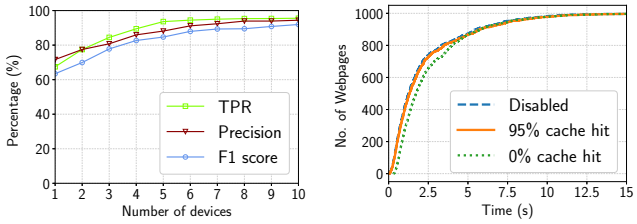
To study binary-class problem using public datasets, we combined YTY2018 (contains network traces for malicious activity) and MSI2017 (contains network traces for benign activity) datasets. In closed-world setting, we trained classification model using Keeper dataset and tested it using combined dataset, and did vice versa for open-world setting. The slight decrease in performance, for closed-world setting (see Tab. VII), can be attributed to the limitations of MSI2017 dataset that contains only two minutes long traces.

We studied the relationship between classification performance and duration of traffic traces used for model training. Figure 4a shows that IoT-KEEPER can detect volumetric network attacks using traces of short duration. This is because the traffic footprint for volumetric attacks is significantly different from normal IoT traffic. On the other hand, longer duration traces are useful for detecting attacks with small network footprint, such as MitM, because over a longer period of time, classification model learns network footprint of IoT device in several modes and use this information to detect these attacks with better accuracy.

When testing classification performance using different combination of features, we noticed that IoT-KEEPER can achieve good classification performance using a smaller subset of features, namely packet count and data length features. Figure 4b shows that IoT-KEEPER achieves $TPR \approx 0.81$ and $FNR \approx 0.11$ using only 15 features extracted from packet timing and packet size information. This analysis shows that packet size and packet timing related features are most helpful in detecting attacks, especially when network traffic

	Closed-world	Open-world
TPR	0.97	0.96
FPR	0.05	0.08
FNR	0.03	0.04
F1	0.97	0.95

TABLE VII: Performance achieved for Binary class problem using a combination of YTY2018 and MSI2017.



(a) (b)

Fig. 5: (a) IoT-KEEPER performance with different number of IoT devices used for data collection. (b) Page load times for top 1000 websites ranked by Majestic.

is encrypted. Meanwhile, a detailed feature set lowers FNR and FPR as it enables the classification model to learn about different types of IoT device activity.

C. System Performance

To study the impact of traffic classification on latency experienced by users, we collected page load times (PLT) for top 1000 websites ranked by Majestic [56]. The measurements were taken for three different scenarios;

1) IoT-KEEPER disabled.
2) IoT-KEEPER enabled with 0% cache hit rate.
3) IoT-KEEPER enabled with 95% cache hit rate.

IoT-KEEPER disabled is the scenario where monitoring, detection, and enforcement modules are disabled. 0% cache hit rate means that the security policy cache is empty and all traffic flows are analyzed by detection module, whereas 95% cache hit rate means that 95% of traffic flows are handled using a matching security policy available in the cache.

Figure 5b shows that when IoT-KEEPER is enabled, on average, PLT increased by upto 4.76% and 15.89% for 95% and 0% cache hit rate, respectively. The increase in PLT is mainly due to analysis performed by detection module. In our experiments, analyzing a single flow took, on average, $223ms(\pm 67.4ms)$, which accounts for $13.93\%(\pm 9.55\%)$ percent of PLT. In comparison, feature extraction, security policy generation and cache-lookup are inexpensive operations, taking $1.59ms(\pm 0.534ms)$, $0.788ms(\pm 0.121ms)$, and $0.007ms(\pm 0.003ms)$, respectively. To study the additional delay in terms of PLT, we use *relative increase in PLT* (PLT_{ri}), which gives the additional delay due to traffic analysis in terms of original PLT. PLT_{ri} is higher (up to 40%) for websites with very small PLT ($\leq 0.5s$), such as google.com, which means loading google.com can be up to 40% slower. However, there are only a handful of webpages with such low PLT,

therefore, high PLT_{ri} is not experienced commonly by users. Meanwhile, PLT_{ri} is low ($\leq 7\%$) for websites with larger PLT ($\geq 1s$), such as instagram.com, qq.com, which is general case (see Fig. 5b).

This additional delay (PLT_{ri}) is only experienced every time a new traffic is analyzed. Since IoT-KEEPER caches analysis results and only the pages that are not accessed for a long time (cache TTL expired) need to be re-analyzed, users will experience (almost) no additional delay for frequently accessed pages. Our experiments show that for 100% cache hit, PLT_{ri} is $1.8\%(\pm 1.49\%)$ only. Meanwhile, PLT_{ri} does not depend on the volume of data loaded for webpage because IoT-KEEPER does not analyze payload data.

Clustering performance: We analyzed the hardware footprint of IoT-KEEPER classification technique by calculating the time required to perform clustering over traffic data. We collected 30 and 60 minute long traffic traces from a 1Gbps network link. Using a Raspberry Pi (model 3B), we were able to perform clustering on 30-min sample in less than 5 minutes, when number of clusters (i) is 17. Average memory and CPU consumption during this process were $64.1\%(\pm 6.9\%)$ and $71.2\%(\pm 7.3\%)$, respectively. This performance is attributed to the observation that, in most cases, network footprint of benign and malicious activity is non-overlapping (see Fig 2). This results in clusters with clearly defined boundaries, where adding a new data point will require fewer computations. The time for clustering decreases to less than 60 seconds if we use a consumer-grade laptop with 2.6GHz 4-core processor with 32Gb memory. The time required for clustering increases linearly with increase in number of data points and number of clusters. In worst case—60-min sample and $i = 50$—it took nearly 21 minutes to perform clustering using Raspberry Pi. This experiment shows us that our classification process is lightweight enough to perform online traffic analysis, using resources available on an edge network gateway.

Memory consumption: In IoT-KEEPER, SDN controller is responsible for traffic monitoring, analysis, and network access control. Therefore, we studied the memory and CPU utilization of IoT-KEEPER controller process. During experimentation, we observed that vanilla Floodlight SDN controller process consumes 12.0% (121MB) of total memory (1GB) available on Raspberry Pi as *resident set size* (RSS) [59], and 38.1% as *virtual memory size* (VMS) [60]. With additional modules used by IoT-KEEPER, RSS and VMS increased to 12.13% (122MB) and 38.9%, respectively.

IoT-KEEPER consumes additional memory for storing device data and security policies (used by AONs). It can consume up to 5MB memory for storing data for 5,000 devices and 5,000 security policies. Since majority of user traffic is destined to a handful of websites and cloud services, we expect to require few security policies to handle majority of network traffic. We have observed that almost 600 security policies (consuming 308 KB memory) can handle up to 95% of network traffic from a typical smart home edge network.

CPU consumption: In our prototype, (fifteen minute) average CPU load, when running vanilla SDN controller, was 38%. Average CPU load increased by 4% when IoT-KEEPER was enabled. The increase in CPU utilization is small because monitoring and enforcement modules use the information that is already extracted by SDN controller during normal packet processing, irrespective of IoT-KEEPER functionality. Only the detection module performs additional processing to analyze traffic flows. Since IoT-KEEPER only analyzes new flows in the network, the resulting increase in CPU utilization is small. In worst case, that is, 0% cache hit rate, average CPU load increases up to 83.3%.

Adhoc overlay networks (discussed in Sect. VIII-A) do not incur CPU overhead as they do not perform additional packet processing. Instead, AONs only introduce minor increase in latency ($\leq 0.01ms$) because of reading security policy (used to set up AON) from policy cache.

Network throughput: We studied the network throughput performance of IoT-KEEPER by measuring layer-4 goodput using `iperf3` [61] and layer-7 goodput (for bulk data transfer) using `curl`. We calculate TCP and UDP latencies using `qperf` [62]. The experiments were conducted for D2D (LAN↔LAN) and D2I (LAN↔WAN) communications, and the performance was compared for scenarios where IoT-KEEPER is enabled (*secure*) versus disabled (*insecure*).

Table VIII shows that traffic classification performed by IoT-KEEPER does not introduce significant deterioration in network performance in comparison to baseline performance achieved using same hardware. It should be noted that our testbed uses a (non-optimized) reference implementation of IoT-KEEPER. As a result, network performance results may vary with different hardware and software stacks used for implementation.

XI. COMPARISON

Table IX presents a qualitative comparison of IoT-KEEPER with state of the art in IoT anomaly detection and device-type recognition.

Anomaly detection: We first compare the anomaly detection performance of IoT-KEEPER with Kitsune [32]. Kitsune was deployed using a Raspberry Pi to detect anomalies in traffic generated by connected security cameras. Here, we also use traffic from security cameras to compare anomaly detection performance of Kitsune and IoT-KEEPER. Table X presents the performance of both techniques to detect Mirai, DoS and ARP poisoning attacks using following three datasets;

1) *YTY2018*: Kitsune dataset.
2) *Keeper*: Data collected from our testbed.
3) *Combined*: Combination of Kitsune and Keeper dataset.

We observed that the performance gap between Kitsune and IoT-KEEPER widens when single-purpose IoT devices are used to generate malicious traffic. We attribute this phenomenon to low network activity of single-purpose IoT devices which were not considered by Kitsune. However, it requires further exploration to identify the factors affecting performance of Kitsune in detecting malicious activity of single-purpose IoT devices.

While Kitsune can detect anomalies in network traffic, it does not provide a mechanism to protect the network against the devices generating malicious traffic. Kitsune also does not support classification model sharing among multiple deployments. Therefore, each new deployment needs to bootstrap

		D2D		D2I	
Metric	**Direction**	**Insecure**	**Secure**	**Insecure**	**Secure**
Layer 4 goodput	Up	89.97 (±0.77)	89.69 (±0.03)	90.11 (±0.80)	88.91 (±0.10)
	Down	90.46 (±0.34)	89.70 (±0.02)	91.01 (±1.53)	89.70 (±0.15)
Layer 7 goodput	Up	87.67 (±1.32)	84.152 (±0.12)	89.94 (±0.60)	86.23 (±0.34)
	Down	88.60 (±1.52)	88.17 (±2.42)	89.12 (±0.89)	87.78 (±1.22)
Bufferbloat latency (ms) (speedtest [57])	Up	2.11 (±0.40)	3.02 (±0.36)	3.77 (±0.24)	3.01 (±0.36)
	Down	90.71 (±2.01)	92.02 (±2.31)	81.41 (±2.67)	82.83 (±2.10)
Bufferbloat latency (ms) (RRUL test [58])	Up	2.11 (±0.13)	2.82 (±0.44)	2.92 (±0.89)	3.22 (±0.77)
	Down	45.81 (±1.73)	50.13 (±1.44)	54.11 (±1.87)	55.93 (±2.44)
Latency (ms)	TCP	0.37 (±0.004)	0.42(±0.003)	0.38 (±0.003)	0.38 (±0.004)
	UDP	0.38 (±0.003)	0.40 (±0.003)	0.39 (±0.004)	0.39 (±0.003)

TABLE VIII: Network performance achieved by IoT-KEEPER using Raspberry Pi based deployment

Technique	**Anomaly detection**	**Device Identification**	**Traffic filtering**	**Security policy sharing**	**Model sharing**	**Learning method**	**Resource utilization**	**Secure D2D**
IoT Sentinel [18]	✖	✔	✖	✖	✖	S	H	⊘
AuDI [16]	✖	✔	✖	✖	✖	U	L	⊘
Securebox [19]	✖	✖	✔	✔	⊘	⊘	L	✔
Kitsune [32]	✔	✖	✖	⊘	✖	U	L	✖
DIöT [17]	✔	✖	✖	✖	✔	S	H	✖
IoT-Keeper	✔	✔	✔	✔	✔	U	L	✔

TABLE IX: Qualitative comparison with state of the art in IoT device identification and anomaly detection. ✖: no support, ✔: full support, ✔: partial support, L: low, H: high, ⊘: not applicable, U: Unsupervised, S: Supervised

Dataset		**YTY2018**		**Keeper**		**Combined**	
Technique		Kitsune	IoT-KEEPER	Kitsune	IoT-KEEPER	Kitsune	IoT-KEEPER
Mirai	AUC	0.99	0.97	0.86	0.99	0.92	0.98
	EER	0.003	0.01	0.17	0.01	0.1	0.01
DoS	AUC	0.92	0.92	0.87	0.97	0.87	0.94
	EER	0.13	0.09	0.21	0.03	0.18	0.04
ARP	AUC	0.79	0.81	0.62	0.82	0.73	0.82
	EER	0.23	0.24	0.44	0.21	0.32	0.22

TABLE X: Performance comparison of IoT-KEEPER and Kitsune [32] for anomaly detection. AUC = area under curve, EER = equal error rate.

a new classification model. This increases bootstrap time for new deployments and limits scalable deployments especially in networks containing multiple edge gateways.

DIöT [17] enables classification model sharing among multiple nodes using federated learning. It uses GRUs [30] to achieve high detection rates for DoS ($TPR = 0.89$) and scanning attacks ($TPR = 1.0$). However, this technique has high resource footprint and can not be deployed using resource constrained devices, such as network gateways [38], [39]. In comparison to DIöT, IoT-KEEPER achieves $TPR = 0.99$ and $TPR = 0.98$ for different variants of DoS and scanning attacks, and it can also detect other network attacks, such as MitM.

The anomaly detection scheme used by DIöT is dependent on device-type information. DIöT requires additional services to first detect IoT device-type, and provide device-type specific anomaly models which are used for anomaly detection. Given the huge variety of devices, it is difficult to develop and maintain such device-type-specific anomaly models. Meanwhile, any errors in device-type identification will severely affect anomaly detection performance as it will be using wrong anomaly detection model. DIöT has been evaluated against volumetric attacks, such as Mirai, whose network footprint is very different from regular IoT traffic. It is not currently tested for variations of Mirai, and other attacks with small network activity. Like Kitsune, DIöT also does not provide a mechanism to protect against IoT devices performing malicious activity.

Device-type identification: Since IoT-KEEPER is designed to analyze network traffic of IoT devices, we tested its performance in terms of detecting IoT device types. For this purpose, we analyze *setup* and *background activity* traffic to distinguish between IoT devices based on their traffic patterns. We used IoT-KEEPER dataset and MSI2017 to compare device identification performance. The results reported in Tab. XI show that IoT-KEEPER achieves better performance compared to IoT-Sentinel [18] for device identification. AuDI [16] also used a similar set of IoT devices and achieved $TPR = 0.97, FNR = 0.02$ with 98% accuracy. However, we can not compare IoT-KEEPER performance with AuDI because AuDI dataset is not publicly available yet.

IoT-Sentinel is designed to identify model and manufacturer information, such as DLink D942L camera, for IoT devices. While it can identify such information for 10 (out of 27) IoT devices with high accuracy ($Acc \approx 1$), IoT-Sentinel can not differentiate between IoT devices with similar network footprint, resulting in false identifications. IoT-KEEPER and AuDI identify the type of IoT devices, such as security camera. Therefore, the number of false identifications are lower (≈ 0.02) compared to IoT-Sentinel (≈ 0.1). While both IoT-KEEPER and AuDI identify device-type information autonomously, AuDI requires more computational resources for this purpose [16]. Since AuDI uses packet timing information for device-type identification, it is susceptible to performance degradation as packet arrival rates vary due to unexpected delays in packet processing at source and network gateways. Although AuDI agent runs on gateway deployed in edge network, it requires a cloud based service to perform

	TPR	FPR	TNR	FNR	Accuracy
IoT Sentinel [18]	0.59	0.1	0.86	0.4	0.91
IoT-KEEPER	0.98	0.02	0.98	0.01	0.99

TABLE XI: Device-type identification performance

traffic analysis and device identification. This requirement raises similar latency and user privacy concerns as Securebox (discussed in Sect. III).

XII. DISCUSSION

Our evaluation demonstrates that IOT-KEEPER is able to detect and block malicious network activity with high accuracy. In this section, we discuss further aspects of IOT-KEEPER, along with possible limitations of the system.

Deep packet inspection: For privacy concerns, IOT-KEEPER does not perform deep packet inspection or analyze user level data. Therefore, encrypted payload and user data is not currently secured by IOT-KEEPER. This is an open topic, which we intend to investigate further.

Evolution in device behavior: IOT-KEEPER can identify firmware upgrades and configuration changes in IoT devices by detecting change in their network behavior. The knowledge of firmware versions of IoT devices allows us to readily update security policies and to prevent attempts to exploit known issues and vulnerabilities in the given firmware version(s). It also helps in preventing false alarms raised due to changed network behavior. Although IOT-KEEPER can identify firmware and configuration updates, it does not detect minor updates such as updating screen brightness or alarm sound level, which do not affect device' network behavior.

Securing Zigbee and BLE communications: Our classification technique mainly uses features extracted from packet counters, packet timing and inter-arrival delays. If similar information is available from cellular, Zigbee or similar protocols, IOT-KEEPER can be extended to work with these communications as well. We intend to explore this problem in future work.

MAC address spoofing: IOT-KEEPER sets up network access restrictions based on layer-2 MAC addresses. An attacker can circumvent these restrictions by spoofing device MAC address. In such scenarios, as long as the attacker does not exhibit malicious activity, it will have the network access. However, such behavior has no incentive for the attacker. On the contrary, if attacker engages in malicious activity with spoofed MAC address, IOT-KEEPER will detect this and limit network access for spoofed MAC address.

Denial of Service: An attacker can exploit MAC address spoofing to perform DoS attack against IOT-KEEPER. In that case, caching will limit the number of times similar traffic flows, coming from different MAC addresses, are analyzed by the system as redundant requests will be handled by cache. Moreover, our evaluation shows that IOT-KEEPER is able to perform anomaly detection at line speeds without becoming a bottleneck. An attacker can flood the upstream link but such an attack will block all traffic flows in the network, including attacker's own traffic, giving no incentive to the attacker.

XIII. CONCLUSION

This paper presents IOT-KEEPER, a system capable of detecting network attacks and enforcing necessary security measures to prevent IoT devices from performing these attacks. This system secures both network-local (D2D) and remote (D2I) communication of IoT. IOT-KEEPER is a lightweight, hardware-agnostic solution, which can be deployed using network gateways or single board computers. It uses custom feature analysis technique to lower resource footprint of traffic classification method, without compromising classification performance. Our traffic classification scheme uses unlabeled network traffic metadata for feature extraction, and does not require side-channel information, such as device-type data, to detect malicious activity of IoT devices. Our evaluation shows that IOT-KEEPER can detect different types of malicious activity, as well as IoT device-types, in both closed-world and open-world settings. The evaluation results also demonstrate that IOT-KEEPER can effectively integrate traffic analysis functionality in network gateways, without significantly affecting network performance and user experience. Moreover, IOT-KEEPER does not require sophisticated hardware or modifications on existing IoT for its operations. We expect that the dataset provided with this work will be helpful to researchers working on IoT device identification, activity recognition, anomaly detection, and other traffic analysis techniques.

REFERENCES

[1] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, vol. 76, pp. 146 – 164, 2015.

[2] M. Nitti, V. Popescu, and M. Fadda, "Using an iot platform for trustworthy d2d communications in a real indoor environment," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 234–245, March 2019.

[3] D. Pauli, "414,949 d-link cameras, iot devices can be hijacked over the net," https://www.theregister.co.uk/2016/07/08/414949_dlink_cameras_iot_devices_can_be_hijacked_over_the_net/, The Register, 2017, [Accessed: 2018-07-21].

[4] Z. Zorz, "Exploitable gsoap flaw exposes thousands of iot devices to attack," https://www.helpnetsecurity.com/2017/07/19/exploitable-gsoap-flaw-iot-devices-exposed/, HelpNetSecurity, 2017.

[5] M. Patton, E. Gross, R. Chinn, S. Forbis, L. Walker, and H. Chen, "Uninvited connections: A study of vulnerable devices on the internet of things (iot)," in *2014 IEEE Joint Intelligence and Security Informatics Conference*, Sept 2014, pp. 232–235.

[6] M. Wall, "How 'the invisible network' poses a major security threat," http://www.bbc.com/news/business-41252203, BBC News, 2017.

[7] M. B. Barcena and C. Wueest, "Insecurity in the internet of things," *Security Response, Symantec*, 2015.

[8] E. Brown, "Linux and open source hardware for iot," https://www.linux.com/news/linux-and-open-source-hardware-iot, Linux.com, 2016.

[9] Microsoft, "Windows 10 internet of things," https://developer.microsoft.com/en-us/windows/iot, Microsoft, 2017.

[10] A. D. Rayome, "The stakes have changed: No end in sigh for ddos attack size growth," https://pages.arbornetworks.com/rs/082-KNA-087/images/WISR_Infographic_NoEndInSight_FINAL.pdf, Tech Republic, 2018.

[11] A. Networks, "Ddos attacks increased by 91% in 2017 thanks to iot," https://www.techrepublic.com/article/ddos-attacks-increased-91-in-2017-thanks-to-iot/, 2017.

[12] "Iot dataset," https://bit.ly/31uuNK7, 2019.

[13] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 266–282, First 2014.

[14] N. Feamster, "Outsourcing home network security," in *Proceedings of the 2010 ACM SIGCOMM Workshop on Home Networks*. ACM, 2010.

[15] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, Aug. 2012.

[16] S. Marchal, M. Miettinen, T. D. Nguyen, A. Sadeghi, and N. Asokan, "Audi: Toward autonomous iot device-type identification using periodic communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, June 2019.

[17] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A. Sadeghi, "Dïot: A crowdsourced self-learning approach for detecting compromised iot devices," *CoRR*, vol. abs/1804.07474, 2018.

[18] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2177–2184.

[19] I. Hafeez, A. Y. Ding, L. Suomalainen, A. Kirichenko, and S. Tarkoma, "Securebox: Toward safer and smarter iot networks," in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. ACM, 2016.

[20] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, Fourth 2008.

[21] A. Morichetta and M. Mellia, "Longitudinal exploration for network traffic analysis from passive data," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 814–827, Sep. 2019.

[22] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, Sep. 2019.

[23] A. Lara and B. Ramamurthy, "Opensec: Policy-based security using software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, March 2016.

[24] W. Chen, X. Luo, and A. N. Zincir-Heywood, "Exploring a service-based normal behaviour profiling system for botnet detection," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 947–952.

[25] R. Mohammadi, R. Javidan, and M. Conti, "An sdn-based lightweight countermeasure for tcp syn flooding attacks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, June 2017.

[26] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th Conference on Security Symposium*. USENIX Association, 2008, pp. 139–154.

[27] H. Garcia-Molina, M. Joglekar, A. Marcus, A. Parameswaran, and V. Verroios, "Challenges in data crowdsourcing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 4, April 2016.

[28] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[29] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, June 2019.

[30] N. N. Thi, V. L. Cao, and N. Le-Khac, "One-class collective anomaly detection based on long short-term memory recurrent neural networks," *CoRR*, vol. abs/1802.00324, 2018.

[31] B. J. Radford, B. D. Richardson, and S. E. Davis, "Sequence aggregation rules for anomaly detection in computer network traffic," *CoRR*, vol. abs/1805.03735, 2018.

[32] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *CoRR*, vol. abs/1802.09089, 2018.

[33] D. Pauli, "Connected kettles boil over, spill wi-fi passwords over london," https://www.theregister.co.uk/2015/10/19/bods_brew_ikettle_20_hack_plot_vulnerable_london_pots/, 2015, [Accessed: 2017-05-07].

[34] M. Kumar, "How to hack wifi password from smart doorbells," https://thehackernews.com/2016/01/doorbell-hacking-wifi-pasword.html, 2016.

[35] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017, pp. 1093–1110.

[36] R. Mortier, J. Zhao, J. Crowcroft, L. Wang, Q. Li, H. Haddadi, Y. Amar, A. Crabtree, J. Colley, T. Lodge, T. Brown, D. McAuley, and C. Greenhalgh, "Personal data management with the databox: What's inside the box?" in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. ACM, 2016, pp. 49–54.

[37] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *2014 IEEE Conference on Communications and Network Security*, Oct 2014, pp. 247–255.

[38] Linksys, "Wrt32x ac3200 wifi router," https://www.linksys.com/us/p/P-WRT32X/, 2019.

[39] Netgear, "Nighthawk x4s smart wifi gaming router," https://www.netgear.com/home/products/networking/wifi-routers/R7800.aspx, 2019.

[40] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015, pp. 117–130.

[41] Y. C. Chang and S. M. Chen, "Temperature prediction based on fuzzy clustering and fuzzy rules interpolation techniques," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, Oct 2009.

[42] K. Zhou, C. Fu, and S. Yang, "Fuzziness parameter selection in fuzzy c-means: The perspective of cluster validation," *Science China Information Sciences*, vol. 57, no. 11, pp. 1–8, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11432-014-5146-0

[43] L. Groll and J. Jakel, "A new convergence proof of fuzzy c-means," *IEEE Transactions on Fuzzy Systems*, vol. 13, no. 5, Oct 2005.

[44] R. J. Hathaway and J. C. Bezdek, "Local convergence of the fuzzy c-means algorithms," *Pattern Recognition*, vol. 19, no. 6, 1986.

[45] M. Yang, "Convergence properties of the generalized fuzzy c-means clustering algorithms," *Computers & Mathematics with Applications*, vol. 25, no. 12, pp. 3 – 11, 1993.

[46] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs, "Nbclust: An r package for determining the relevant number of clusters in a data set," *Journal of Statistical Software, Articles*, vol. 61, no. 6, pp. 1–36, 2014.

[47] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987.

[48] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of Royal Statistical Society, Statistical Methodology*, vol. 63, no. 2, pp. 411–423, 2001.

[49] Z. Huang and Q. Shen, "Fuzzy interpolation and extrapolation: A practical approach," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 1, pp. 13–28, Feb 2008.

[50] G. Marks, "Netflix and youtube now consume 50% of the internet as the argument for net neutrality weakens," https://bit.ly/362KYR9, Forbes, 2018.

[51] D. M. Mendez, I. Papapanagiotou, and B. Yang, "Internet of things: Survey on security and privacy," *CoRR*, vol. abs/1707.01879, 2017.

[52] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *CoRR*, vol. abs/1708.05044, 2017.

[53] K. Lab, "Kaspersky iot scanner: How to keep your home network and its smart devices safe," https://www.kaspersky.com/blog/kaspersky-iot-scanner/18449/, Kaspersky, 2017.

[54] P. Martin, "Using your new raspberry pi 3 as a wifi access point with hostapd," https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/, Firllip's Blog, 2016, [Accessed: 2018-10-12].

[55] Big Switch Networks, "Project floodlight - floodlight OpenFlow controller," http://www.projectfloodlight.org/floodlight/, Oct. 2016, [Accessed: 2016-09-17].

[56] Majestic, https://majestic.com/reports/majestic-million, Majestic.

[57] Betterspeedtest, https://github.com/richb-hanover/CeroWrtScripts/blob/master/betterspeedtest.sh.

[58] Netperfrunner, https://github.com/richb-hanover/CeroWrtScripts/blob/master/netperfrunner.sh.

[59] Corbet, "Elc: How much memory are applications relaly using?" https://lwn.net/Articles/230975/, LWN.net, 2019.

[60] Linux, "Proc(5) linux programmer's manual," http://man7.org/linux/man-pages/man5/proc.5.html, Linux, 2019.

[61] IPerf3, https://iperf.fr/.

[62] qperf, https://github.com/HewlettPackard/netperf.