Detecting Behavioral Change of IoT Devices using Clustering-Based Network Traffic Modeling

Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman

***Abstract*—The Internet-of-Things (IoT) is increasingly becoming a major challenge for network administrators to manage connected devices and sensors ranging from smart-lights to smoke-alarms and security-cameras, at scale. IoT devices use an extensive variety of firmware, and provide little (or no) access for management of their operating systems and configurations. Operators of IoT infrastructure, therefore, need to employ traffic classification models (trained by historical data) to automatically detect their assets on the network, and ensure the health of devices against cyber-attacks by monitoring their network behavior. On the other hand, IoT manufacturers often automatically perform firmware upgrades from cloud servers to devices that are operational in the field. This can potentially lead to a change of device behavior which makes it difficult for network operators to maintain classification models (incorporating changes without retraining the entire model).**

In this paper, we develop a modular device classification architecture that allows operators to automatically detect IoT devices by their network activity and dynamically accommodate legitimate changes in assets (either addition of new device profile or upgrade of existing profiles). Our contributions are threefold: (1) We identify key traffic attributes that can be obtained from flow-level network telemetry to characterize the behavior of various IoT device types. We develop an unsupervised one-class clustering method for each device to detect their normal network behavior; (2) We tune device-specific clustering models and use them to classify IoT devices from their network traffic in real-time. We enhance our classification by developing methods for automatic conflict resolution and noise filtering; and (3) We evaluate the efficacy of our scheme by applying it to traffic traces (benign and attack) from ten real IoT devices, and demonstrate its ability to detect behavioral changes with an overall accuracy of more than 94%.

***Index Terms*—IoT devices, traffic modeling, clustering.**

I. Introduction

The Internet-of-Things (IoT) continues to expand its reach into homes, offices, enterprise campuses, and even cities, as more devices are rapidly connected to networks for collecting and sharing data. Management of these connected devices has increasingly become challenging, particularly from a security standpoint, for large networks such as those found in large enterprises and university campuses, for example. Such networks may include thousands of IoT devices which largely remain unidentified [2], and hence pose significant security risks to the network. Most IoT devices are relatively simple, and cannot defend themselves against cyber-attacks. There are many examples of unmonitored IoT devices which have already caused data breaches or been hijacked to carry out large-scale attacks on the Internet [3].

A promising approach to vulnerabilities of IoT devices is to embed solutions at the network-level, whereby network traffic to/from IoT devices is monitored to ensure they operate normally and detect abnormal behaviors (attacks) [4]. IoT devices are typically purpose-built with limited functionalities – they communicate with a specific set of endpoints (*i.e.,* servers) using a small number of TCP/UDP flows [5], [6]. Therefore, a growing number of traffic classification proposals are emerging based on supervised machine-learning techniques (*e.g.,* multi-class decision-trees or neural-networks) that use packet-level [7], flow-level [8], or a combination of packet-level and flow-level [9] traffic attributes to determine network traffic patterns (*i.e.,* behavior) of IoT devices on the network, and to use those network traffic patterns as fingerprints to automatically identify the type of IoT devices (typically including their function, manufacturer, and even model number) on the network

Unlike traditional non-IoT devices, the behavior of IoT devices does not change much with user interactions. However, IoT manufacturers update the firmware of their devices in the field via automatic/semi-automatic processes [10] to enhance the functionality of the devices. These firmware updates pushed by vendors can change the device behaviors to some extent. In addition to legitimate changes (due to firmware upgrades), cyber-attacks would also alter the activity patterns of connected devices, and hence are expected to affect the output or confidence of machine learning-based classifiers.

In our prior work [9] we showed that generating a multi-class classifier becomes practically challenging at scale when a new device type is added to the network or the behavior of even one of the previously classified device types legitimately changes due to a firmware upgrade by the device manufacturer – it is needed to regenerate the entire model of all classes. Moreover, to reduce the impact of over-fitting the trained multi-class model to specific classes, it is necessary to carefully balance the training dataset by representing classes equally, which is a nontrivial task. IoT devices with richer features and diverse functionalities require more instances to capture their normal behavior than other types of IoT devices.

In order to address these difficulties, in this paper, we replace the multi-class (IoT device type) classifier with an "inference engine" having a set of one-class clustering models (one model per IoT device), each of which can be independently trained and updated when required. Our **first** contribution

A. Sivanathan, H. Habibi Gharakheili, and V. Sivaraman are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: a.sivanathan@unsw.edu.au, h.habibi@unsw.edu.au, vijay@unsw.edu.au).

This submission is an extended and improved version of our paper presented at the IEEE LCN 2019 [1].

identifies IoT traffic attributes that are computed from real-time flow-level telemetry. We show how clusters of attributes can characterize network behavior of each device type. Our **second** contribution develops a classification scheme using a set of device-specific clustering models augmented by an automatic conflict-resolution method and dynamic consistency scores to realize real-time network monitoring. For our **third** contribution, we apply our classification solution to real traffic traces (mix of benign and attacks) from ten IoT devices and demonstrate its accuracy of more than 94% in detecting behavioral changes.

II. Related Work

Traffic Classification and Monitoring: Automatic traffic classification and monitoring using machine learning techniques have been the subject of networking research over the past decades [11]. In the context of IoT, these techniques are used to: (a) distinguish IoTs from a mixture of IoT and non-IoT devices [9], [12], (b) classify IoT device types [13], [14], and (c) detect abnormal behavior of IoTs [15], [16]. The machine learning techniques and algorithms used in prior work are either supervised or unsupervised.

Supervised Classification Techniques: Supervised classification algorithms (*e.g.,* Support Vector Machines (SVM), naive Bayes, decision trees, neural network) are commonly used for classification purposes. In the literature these algorithms are used in two different modes: (a) multi-class classification – classify the data across three or more classes (*e.g.,* IoT device type classification/ operating states identification), and (b) binary classification – make decision across only two classes (*e.g.,* IoT versus non-IoT).

One of our earlier works [9] developed IoT device type classification using the Random-Forest ensemble tree classifier (multi-class) to achieve the accuracy over 99%. Such attractive performance is achieved because multi-class models are discriminative (they learn the decision boundaries between different classes). However, the multi-class classification comes with a practical issue of scalability – a large number of classes (device types) make it difficult to update the model (regenerating the entire model).

Binary classifiers are trained by data of two distinct classes. In the context of classifying general network traffic, this approach is used for various purposes like distinguishing known attacks from benign traffic [17], or recognizing IoT traffic versus non-IoT traffic. Authors of [15] train a binary classifier to identify the DDoS attack traffic (generated by Mirai botnets) and benign traffic of the IoT devices. Although this method performs well in detecting "known" attacks (trained signature), its efficacy is not guaranteed for new/unknown attacks. Work in [12] builds a specific binary classification model for each of device types – "one IoT device type" and "rest of device types". It is well understood that supervised machine learning models can suffer from unbalance dataset. This makes is difficult for "one vs. rest" binary classifiers to scale when the number of device types increases (in large networks) resulting in growth of "rest" labeled data, and hence unbalanced dataset.

Unsupervised Classification Techniques: Unsupervised machine learning techniques are generative – are able to learn the distribution of training data, and detect any changes (anomalies) during testing phase. Work in [18] employs an unsupervised clustering approach to distinguish botnets command-and-communication (C&C) from benign traffic for a general-purpose devices (traditional IT environment). The study in [19] shows that unsupervised algorithms can suffer from the curse of high dimensional data. Therefore authors propose to reduce the dimension using Principal Component Analysis (PCA) which converts the correlated attributes into a reduced set of attributes. In the IoT domain, authors of [20] use deep neural network-based auto-encoders to detect the anomalies. However, this approach is computationally expensive. In [3] authors develop a learning-based method using one-class classifiers for real-time detection of volumetric attacks on IoT devices by measuring the activity of MUD-compliant network flows of individual devices on the network – authors employ a specialized model per each flow of every IoT devices. Unsupervised learning methods are not discriminative and hence not easy for classification problems. Our preliminary work in [1] resolves this issue by developing a probability-based conflict resolver.

Network-based Intrusion Detection Systems (NIDS): Network-level security and monitoring solutions for traditional IT networks have been extensively studied [21]–[23] over the past few decades by the research community. NIDS use three techniques to detect malicious network behaviors: (a) signature-based methods, (b) specification-based methods, and (c) anomaly-based methods [24].

Signature-based Methods: This category of techniques is commonly used in some NIDS products such as Bro [25], Snort [26], and commercial hardware appliances. They check observed traffic (of general-purpose computers) against a set of already known attacks signatures collected from the sandbox environment and honeypots. This approach has poor resilience to morphed or zero-day (never seen before) attacks that can render known signatures useless. Also, due to heterogeneity of IoTs and the growing diversity of attacks on these special-purpose devices, generating attacks signature becomes practically infeasible – unlike in general-purpose computers where operating systems are fairly limited to a handful of options like Windows, Linux, macOS, or Android.

Specification-based Methods: This approach monitors the activity of connected devices given certain rules, explicitly specifying allowed or not-allowed network activities [22], [27]. In [28], authors apply a specification-based approach to a wireless sensor network where specifications are defined and enforced by the network operator. However, generating specifications for every device can become a tedious task, and may require human experts [29]. The IETF has recently standardized a scheme called Manufacturer Usage Description (MUD) for IoT vendors to formally describe the intended network behavior of their devices [30]. Work in [31] develops a NIDS by enforcing MUD profile of devices to the network and detect attacks by inspecting traffic that does not conform

TABLE I: Flow rules (per-device) needed for network traffic telemetry.

Flow description	srcETH	dstETH	srcIP	dstIP	srcPort	srcPort	proto
DNS↑	`<devMAC>`	*	*	*	*	53	17
DNS↓	*	`<devMAC>`	*	*	53	*	17
NTP↑	`<devMAC>`	*	*	*	*	123	17
NTP↓	*	`<devMAC>`	*	*	123	*	17
SSDP↑	`<devMAC>`	*	*	*	*	1900	17
remote↑	`<devMAC>`	`<gwMAC>`	*	*	*	*	*
remote↓	`<gwMAC>`	`<devMAC>`	*	*	*	*	*
local↓	*	`<devMAC>`	*	*	*	*	*

to MUD rules. Specification-based IDSes are effective in reducing attack surfaces, but sophisticated attacks can still be launched.

Anomaly-based Methods: This approach learns legitimate (normal) behavior of network traffic and detects deviations from the expected boundaries – these methods are potentially able to flag zero-day attacks [32], [33]. Although there is an extensive body of literature [34]–[36] on this topic, the success of this has been fairly limited [37] in IT environments (network of general-purposed devices). This is mainly because [38] legitimate traffic shows high variability in IT networks. For IoT networks, instead, this approach seems more promising as network activity of IoT devices (unlike servers and computers) can be captured by a limited set of identifiable patterns, and hence it become easier to characterize the entire behavioral profile of the devices [16] from their network traffic.

III. Clustering Flow-Level Attributes of IoT Network Traffic

In this section, we first outline our IoT dataset, network telemetry, and traffic attributes. We, next, show how clusters of attributes will characterize network behavior of individual IoT devices.

A. Flow-Level Telemetry and Traffic Attributes

Dataset: There are limited datasets publicly available that consist of IoT benign and attack traffic traces, and hence in this paper we use packet traces (two sets) collected from our lab environment to demonstrate the basis and performance IoT traffic modeling. In our previous research works [3], [9], we analyzed both of these datasets namely, DATA1 (benign traffic) and DATA2 (mix of benign and attack traffic).

The first dataset (*i.e.,* DATA1) was collected from a testbed consisting of more than thirty IoT devices for a duration of 6 months (*i.e.,* 01-Oct-2016 to 31-Mar-2017). This data was collected for our previous work [9] which presented a detailed analysis of this dataset. We select 12 IoT devices namely Amazon Echo, Belkin motion sensor, Belkin switch. Dropcam, HP printer, LiFX bulb, Netatmo weather station, Netatmo camera, Samsung camera, Smart Things, Triby speaker, and Withings sleep sensor those showed significant activities during the early period of the dataset (*i.e.,* 1-Oct-2016 to 15-Nov-2016). We evaluate (in §IV) on DATA1 the efficacy of our inference engine in classifying profiles of IoT devices as well as detecting their behavioral changes.

Our second dataset (*i.e.,* DATA2) contains more than 8 weeks' worth of PCAP traces collected from 10 IoT devices (in a different environment) over two months in 2018. The DATA2 was collected and analyzed for another work in [3] done by our research group. DATA2 includes normal traffic (covering boot, active, and idle operating states) and also attack traffic (direct and reflective) on these IoT devices. We use DATA2 (in §V-C) to evaluate the performance of our scheme in real-time detection of cyber-attacks which cause behavioral changes in IoT network traffic.

TABLE II: Summary of partial DATA1 (benign traffic): Device instances and clustering parameters.

	Instance count		**Unsupervised classifier parameters**	
Device	Training (1-month)	Testing (2-week)	# Principal components	# clusters
Amazon Echo	40843	18694	19	256
Belkin motion	35153	18780	17	256
Belkin switch	40991	18771	18	256
Dropcam	41089	18787	9	128
HP printer	40713	18693	13	128
LiFX bulb	36952	18707	14	256
Netatmo cam	40788	18706	15	512
Netatmo weather	24896	17473	9	128
Samsung cam	40841	18696	16	256
Smart Things	41073	18799	13	256
Triby speaker	31898	18694	15	256
Withings sleep sensor	32033	10877	12	128

Flow-Level Telemetry and Attributes: We showed in our prior work [9] that individual IoT devices exhibit identifiable patterns in their traffic flows such as activity cycles and volume patterns, and profiles of signaling protocols such as DNS, NTP, SSDP. To monitor IoT behavior on the network in real-time, we identify a set of flows (specific to each device) that collectively capture its entire traffic. These flow rules can be programmed into an SDN-enabled switch [39], [40] through which the traffic of IoT devices passes – rules of different devices are distinguished by a match field corresponding to device identifier (*i.e.,* MAC or IP address). Counters (bytes and packets) of these flow rules are periodically (configurable, say, every minute) retrieved from the network SDN switch, and will form traffic attributes of individual devices.

Table I shows eight flow rules that we use to measure network traffic of each IoT device with the following order: **(1,2)** DNS outgoing queries and incoming responses on UDP 53, **(3,4)** NTP outgoing queries and incoming responses on UDP 123, **(5)** SSDP outgoing queries on UDP 1900, **(6,7)** other "remote" traffic (*e.g.,* Internet) outgoing from and incoming to the device that passes through the gateway, and **(8)** all "local" traffic (*i.e.,* LAN) incoming to the device. Note that we do not monitor SSDP traffic incoming to IoT devices to avoid capturing (and mixing) the discovery activities of other devices on the local network. Also, we do not monitor local traffic coming to IoT device as this traffic is assumed to be originated from another IoT device locally – this way, activity of local flows is counted only for one device (receiver). We

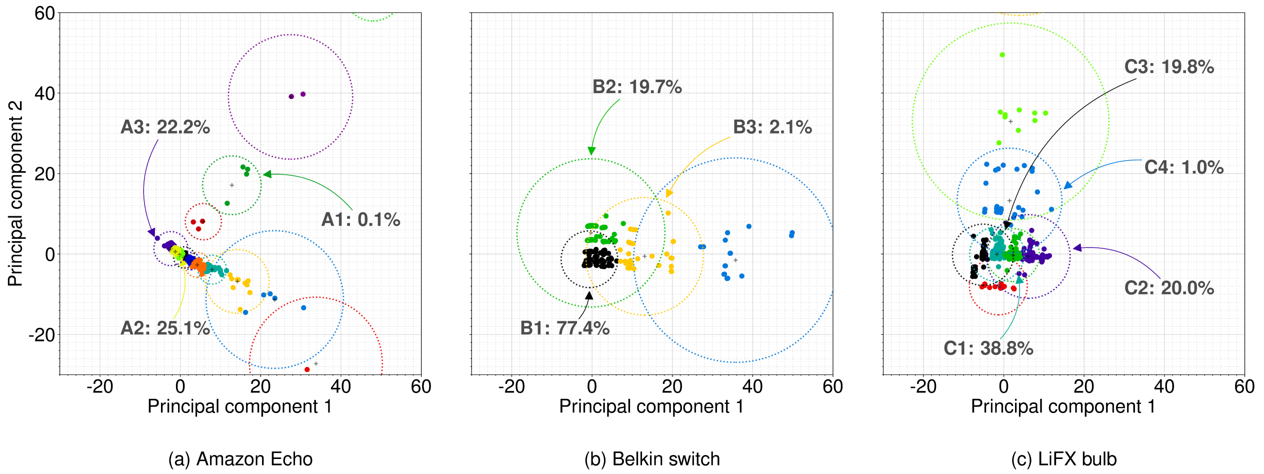


Fig. 1: Clusters of data instances in two-dimensional space for representative IoT devices: (a) Amazon Echo, (b) Belkin switch, and (c) LiFX bulb.

have used MAC address as the identifier of a device – one may use IP address (*i.e.,* without NAT), physical port number, or VLAN for a one-to-one mapping of a physical device to its traffic trace.

We use two key attributes [41] namely ***average packet size*** and ***average rate*** for each of the eight flows mentioned above. We also note that traffic attributes can better characterize individual devices if they are computed at multiple time-scales [42] particularly in the characterization of long-range dependent traffic. Per-flow packet and byte counts are generated every minute (as they would be retrieved from the network switch in a live network environment), and flow attributes are generated for time-granularities of 1-, 2-, 4-, and 8-minutes, providing eight attributes (*i.e.,* average packet size and rate) for each flow, and a total of 8×8=64 attributes per IoT device.

Extracting Attributes: In order to synthesize flow entries and thereby extract attributes from the traffic traces, we use our native packet-level parsing tool [41]. It takes raw PCAP files as input, develops a table of flows (like in an SDN switch) and exports byte/packet counters of each flow at a configurable resolution (*e.g.,* 60 sec). Lastly, we generate a stream of instances (a vector of attributes periodically generated every minute) corresponding to each of individual devices.

We begin with DATA1, and use a month's worth of its data (*i.e.,* 01-Oct-2016 to 31-Oct-2016) for training and the following 2 weeks for testing our models – the second column in Table II summarizes the number of training /testing instances per each device type contained in this part of DATA1. Later in §IV, we will use the rest of DATA1 (spanning a longer period of traffic traces) to show how our models detect changes in IoT behaviors.

B. Attributes Clustering

Our primary objective is to train a number of one-class models (one per IoT device) where each model recognizes traffic patterns of only one particular device type (*i.e.,* one class) and rejects data from all other classes – *i.e.,* a one-class classifier generates "positive" outputs for known/normal instances, and a "negative" output otherwise. This use of one-class models means that each model can be re-trained independently of the other models of the set (in cases of legitimate changes). Also, It has been shown that device-specialized models can better detect anomalous traffic patterns (outliers) [3]. There are a number of algorithms for one-class classification. One of the most common and efficient methods is K-means [43] which finds groups of instances (*i.e.,* "clusters") for a given class that are similar to one another. Each cluster is identified by its centroid, and an instance is associated with a cluster if the instance is closer to the centroid of that cluster than any other cluster centroids.

To provide insights into the traffic characteristics of IoT devices, we show in Fig. 1 the resulting clusters of instances for three representative devices from our dataset namely, the Amazon Echo, Belkin switch, and LiFX bulb. Note that our instances are multi-dimensional (*i.e.,* each instance includ-

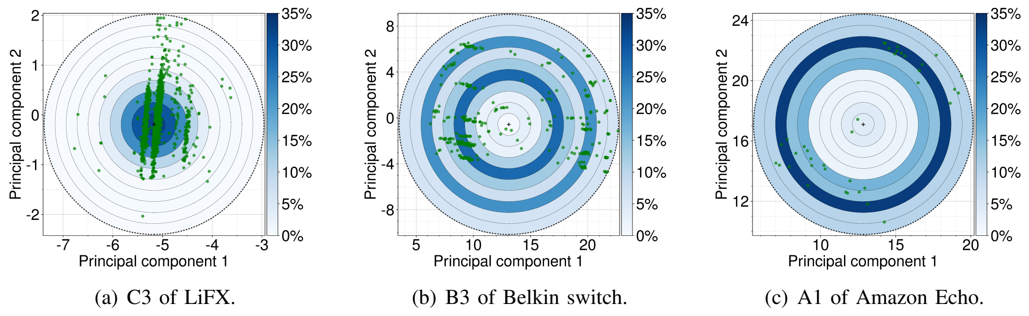


Fig. 2: Distance probability of clusters: (a) C3 of LiFX bulb, (b) B3 of Belkin switch, (c) A1 of Amazon Echo.

ing 64 attributes), and thus can not be easily visualized. Therefore, for illustration purposes only, we employ Principal Component Analysis (PCA) to project the data instances onto two-dimensions – data instances are shown as dots and cluster centroids are shown as crosses. Note that only 10% of instances (in each cluster) are shown for better visualization – as an example, four dots in cluster A1 of the Amazon Echo, shown in Fig. 1(a), represent approximately 40 instances.

Dotted circles depict the boundary of clusters. These boundaries are used to determine whether a test instance belongs to the clusters of a class or not. As per a rule of thumb for finding outliers [44], a boundary for each cluster is chosen in a way to exclude data points whose distance from the centroid is relatively large, specially values more than 1.5 times the interquartile range from the third quartile). In other words we define the boundary for each cluster to include the first 97.5% [45] of data points closest to the cluster center and to exclude farther instances (avoiding impurities in our training dataset).

It is important to note that an actual cluster forms a contour (enclosing associated data points) which could be a complex shape. In practice, we have found that each model for a corresponding IoT device consists of some tens of clusters, and consequently to make the IoT device type classification process computationally cost-effective and more efficient, the shapes of the cluster contours are approximated. Noting also that the K-Means algorithm attempts to partition the training dataset into spherical clusters when it is tuned optimally (*i.e.,* equal distance from centroids in all dimensions), the cluster boundaries are approximated as being spherical for the purpose of determining whether a test instance belongs to the clusters of a class.

As shown in Fig. 1, instances of the Amazon Echo, Belkin switch, and LiFX bulb IoT devices are grouped into 16, 4, and 8 clusters, respectively. We observe in Fig. 1(a) that instance clusters of the Amazon Echo are fairly spread across the two component space. For the Belkin switch device, the clusters in Fig. 1(b) are mainly spread across the principal-component-1 while their principal-component-2 values are narrowly confined between -20 and 20. Lastly, the LiFX bulb instances in Fig. 1(c) are spread along the principal-component-2, but are narrowly confined between -20 and 20 in the principal-component-1. Note that each cluster of a class has a probability (referred to as "cluster likelihood") of covering training instances from the corresponding device type, depending upon device traffic patterns seen in the training dataset. As annotated in Fig. 1, highly probable clusters for the Amazon Echo are A2 (25.1%) and A3 (22.2%), for the Belkin switch are B1 (77.4%) and B2 (19.7%), and for the LiFX bulb are C1 (38.8%) and C2 (20.0%). These clusters highlight the dominant traffic characteristics of their corresponding device.

We also note that the distribution of instances within each cluster also varies across clusters. Fig. 2 shows a zoomed views of one cluster for each of the three representative IoT devices, with instances shown by green dots. Each cluster is divided into 10 concentric annular bands of the same area, starting from the centroid to the cluster boundary. Each band in Fig. 2 is shown with a shading that indicates the fraction of training instances it covers, as indicated by the linear scale at the right-hand side of each Figure (*e.g.,* dark blue indicates a higher probability).

```
Algorithm 1: Training a model.
input  : Training instances of a given device type.
output: A trained model, consisting of:
         Scaler for each attribute,
         Principal components,
         Cluster centers,
         Cluster boundaries and probabilities,
         Associate probability distribution.
1  Record Z-Score scalers for each attribute;
2  Normalize training instances using Z-Score scalers;
3  Obtain and record optimal principal components of all
     attributes;
4  Reduce dimensions of training instances;
5  Compute optimal number of clusters using Elbow
     method;
6  Obtain and record cluster centers;
7  Record boundaries covering nearest 97.5% instances
     for each cluster;
8  Record probability of each cluster;
9  Record probability of distance band inside individual
     clusters;
10 Record a CDF of associate probability for training
     instances.
```

It can be seen in Fig. 2(a) that 95% of LiFX instances inside cluster C3 fall within the four central bands of this cluster. Regarding the Belkin switch in Fig. 2(b), 81% of instances of cluster B3 fall within the middle bands (from the 4th to the 8th bands). Lastly, looking at a less probable cluster A1 of the Amazon Echo in Fig. 2(c), 85% of instances are covered by the last five bands farthest from the centroid. It is importation to note that the 2D space is used here for illustration purposes only. In our classification scheme, we employ a hyper-sphere in 64-dimensional space for clustering instances of IoT traffic attributes.

IV. Unsupervised Classification of IoT Devices

In this section, we describe the architecture of our inference engine which consists of a set of one-class models for individual device types. Next, we develop methods to resolve conflicts between multiple models for device classification. Finally, we develop a scoring technique to measure the consistency of the models in classifying IoT devices, identify two monitoring phases namely initial and stable, and detect behavioral changes.

A. Clustering Models: Generation, Tuning, and Testing

We summarize in Algorithm 1 and Algorithm 2, all the steps required for generating and testing our device-specific models. The rest of this subsection describes the details of each step. Prior to generating clustering models, we pre-process the raw data as follows. First, we normalize each attribute independently to avoid outweighing large-value attributes (*e.g.,* average bytes rate of incoming remote traffic at 8-min timescale) over smaller attributes (*e.g.,* average packet size of outgoing NTP traffic at 1-min timescale) [46] as the magnitudes of different attributes varies significantly (*i.e.,* over several orders of magnitude). We employ Z-score method (*i.e.,* computing the mean μ and standard deviation σ from the training dataset, normalizing by calculating the deviation from the mean divided by the standard deviation) to scale individual attributes. Second, we project data instances onto a lower dimensional space by using PCA [47] to generate linearly uncorrelated principal components. This is because the direct use of 64-dimensional attribute instances for classification can be computationally expensive for real-time prediction, and can also degrade the clustering performance (possibly causing bias towards less significant attributes). The use of orthogonal principal components enables the K-means clustering to generate clusters more clearly by removing redundant and noisy attributes from the training dataset. We choose the number of PCA components to retain the optimum "cumulative variance" [48] for our dimension reduction engine.

Following dimension reduction, we apply K-means clustering, with K values varying as powers-of-2 (*i.e.,* 2^i where $i = 1, ..., 10$). Setting K to small values would not generate an accurate model of network behavior for IoT devices, and large values increase the computational cost in both training and testing phases. Also, a very large K results in smaller-size clusters, and hence a rigid classifier which cannot correctly detect normal (legitimate) instances with small deviations from the training data – *i.e.,* over-fitting. We determine the optimal number of clusters using the elbow method [49].

Fig. 3 shows the average square distance of instances from the cluster centers (*i.e.,* Inertia per instance) as a function of the number of clusters for each of two representative device types. The optimal number of clusters (marked by '$\times$' on each curve) is deemed to be when the first derivative of inertia per instance exceeds a very small negative value -0.01 (the curve

```
Algorithm 2: Testing an instance
input : A test instance,
        Trained models
output: Cluster boundary in (+ve) with a confidence,
        or out (-ve) with N/A confidence.
1 Normalize test instance using scalers;
2 Reduce dimensions of normalized instance;
3 Find nearest cluster center for test instance;
4 if instance is inside a cluster boundary then
5     Compute confidence level;
6     return +ve and confidence level.
7 else
8     return -ve and N/A.
9 end
```

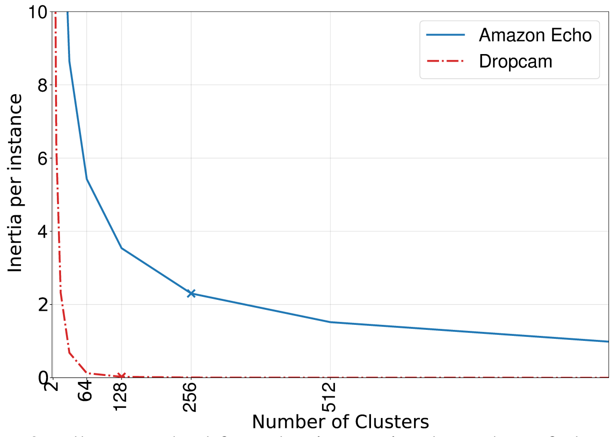


Fig. 3: Elbow method for selecting optimal number of clusters.

is becoming a reasonably saturation state). It can be seen that the model for the Amazon Echo device needs 256 clusters for a close-to-optimal performance (first derivation of inertia per instance is $-0.0097 > -0.01$), whereas the Dropcam device needs only 128 clusters (first derivation of inertia per instance is $-0.0015 > -0.01$). We show in the rightmost column of Table II, the optimal model parameters for individual device types, determined as described above.

Having generated the clustering models, we process each instance of IoT traffic attributes (after scaling and dimension reduction) as shown by the sequence of steps in Fig. 4. The test instance is provided to all of the device-specific models (one-class classifiers) to determine the nearest centroid of each model, using the Euclidean distance between the test instance and each cluster centroid. Given a nearest centroid, the instance is checked against the corresponding cluster to determine whether it falls inside or outside of that cluster boundary, and if inside, a confidence level is calculated. Note that finding the distance between the test instance and the cluster centers of individual models is not computationally expensive, and hence can be done in real-time.

To better illustrate this process, let us consider the two-dimensional space of clusters described earlier in Fig. 1. Assume that a test instance has its principal component-1 and component-2 equal to 0 and 20, respectively. The nearest cluster centroids to this test instance are cluster A1 of the Amazon Echo, cluster B2 of the Belkin switch, and cluster C4 of the LiFX bulb. Since the test instance falls outside of the A1 boundary, the Amazon Echo model provides a negative output while the other two models both give positive outputs. . In such cases where multiple models provide positive outputs for the same instance, a conflict resolution process (what follows next) is used to select the "winner" model.

B. Conflict Resolution

Although each model learns the normal behavior of one device type, different devices can display somewhat similar traffic behavior (*e.g.,* DNS, NTP or SSDP) for a short period of time [8], which can result in multiple positive outputs generated by the clustering models for an instance. In such cases, "confidence" values are generated for the models that gave

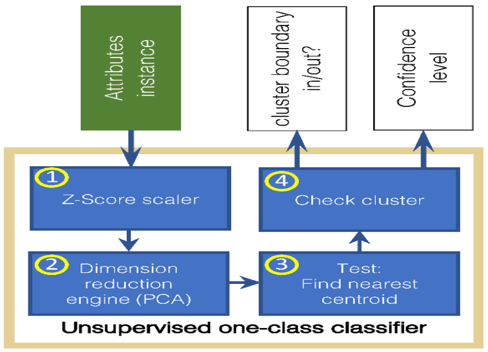

Attributes instance
cluster boundary in/out?
Confidence level
1 Z-Score scaler
4 Check cluster
2 Dimension reduction engine (PCA)
3 Test: Find nearest centroid
Unsupervised one-class classifier

Fig. 4: Use of each clustering model for a test instance.

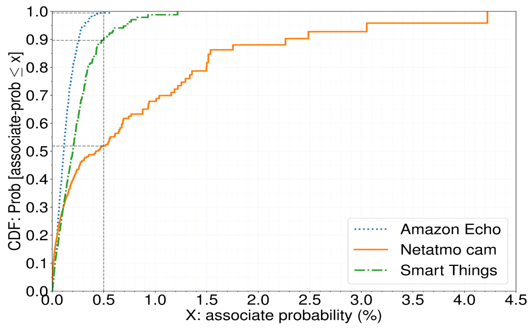


Fig. 5: Distribution of clustering probability for training instances of three device types.

positive outputs, and the model with the highest confidence value is selected as the winner.

Confidence-level: Each "confidence" value (also referred to herein as "associate probability") represents a probability value for an instance to be associated with a cluster of that model. Given an instance Ins receiving a positive output from a model M_i and falling within a distance band D_l of the nearest cluster C_j (of the model M_i), the corresponding associate probability is estimated by:

$$P^{test}_{[Ins|M_i(C_j(D_l))]} = P^{train}_{[C_j|M_i]} \times P^{train}_{[D_l|C_j]} \qquad (1)$$

where $P^{train}_{[C_j|M_i]}$ is the likelihood of the nearest cluster C_j within the model M_i and $P^{train}_{[D_l|C_j]}$ is the probability of distance band D_l inside the cluster C_i – both of these probability values being obtained from the training dataset. We note that $P^{train}_{[C_j|M_i]}$ is always non-zero (by optimal tuning [50]), but it is possible to have $P^{train}_{(D_l|C_j)}$ equals to zero when none of the training instances fall inside a band D_l (*i.e.,* unexplored distance bands in the training data). To avoid a zero confidence for test instances, we slightly modify the band probability using the Laplacean prior [51], priming each band instances count with a count of one, as given by:

$$P^{train}_{[D_l|C_j)]} = \frac{1 + N_{D_l}}{L + N_{C_j}} \qquad (2)$$

where N_{D_l} is the number of training instances inside the band D_l; N_{C_j} is the total number of instances in the cluster C_j; and L is the total count of distance bands in the cluster – we use ten bands in every cluster (*i.e.,* $L = 10$).

The associate probability, to some extent, indicates the model confidence. However, it becomes challenging to select the winner among multiple models giving positive outputs since the number of clusters and also the distribution of distance bands vary across models, and hence the associate probability is scaled differently. For example, models with large numbers of clusters may have relatively smaller values of $P^{train}_{(C_j|M_i)}$, or a cluster with highly sparse bands would result smaller values of $P^{train}_{(D_l|C_j)}$.

To obtain a metric of confidence for comparison across different models, we first obtain the distribution of associate probabilities in the training dataset. We, then, use this distribution to scale the associate probability of test instances. To better illustrate this scaling process, we show in Fig. 5 the cumulative distribution function (CDF) of the associate probability for training instances of three representative IoT models namely, the Amazon Echo, Netatmo cam, and Smart Things models. Considering the Amazon Echo (shown by dotted blue lines), we note that an associate probability of 0.5% is a high value for this model since more than 99% of its training instances have a probability value lower than 0.5%. However, this percentage drops to 90% and only 52% for the Smart Things (dashed green lines) and Netatmo (solid orange) models, respectively. Therefore, during the testing phase, we use the associate probability resulted from a model (giving a positive output to a test instance) to determine the model confidence-level by computing the fraction of its training data with associate probability values that fall below that of the test instance (with respect to the model's empirical CDF of associate probability).

C. Consistency Score

Ideally, for monitoring individual IoT devices consistent outputs should be generated by the inference engine over time. However, a given device that is consistently and correctly classified by a model over a period of time (say, a week), may nevertheless occasionally be rejected (*i.e.,* negative output) by its intended model. To bootstrap the monitoring process for a newly connected (and possibly unknown) device, at least one of the corresponding device models should consistently generate positive outputs in order to accept the device and label it by the corresponding class (at which time the device is said to be in a "stable state"). Once a device becomes known ("accepted") and is in its stable state, receiving negative outputs frequently from its corresponding model indicates a change (legitimate or illegitimate) in the device behavior, which requires further investigations. To enable this detection, we generate a "consistency score" (between 0 and 1) representing the consistency of device classification. Fig 6 shows the architecture of the inference engine, with classification followed by consistency scoring.

For instances of a given IoT device, the consistency score is computed and stored for each model, and updated following classification of each instance – the consistency score of a model rises by its positive outputs and falls by its negative outputs over time. To better understand the dynamics of this score, let us consider an example. We take three days of

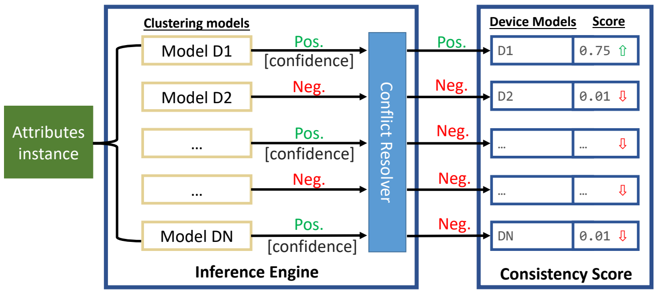


Fig. 6: Architecture of our inference engine.

instances from a Smart Things device, and replay them in real-time to four trained models respectively for the Smart Things, Netatmo, Amazon Echo, and Withing Sleep sensor IoT devices. We show in Fig. 7 the consistency scores of these four models over time. As expected, the score of the Smart Things model (shown by solid lines) is dominant, while the scores for the other three models are negligible ((and hence not apparent in the Figure). The Smart Things model score slowly rises and reaches to a high level of 0.8 after about 30 hours. We also observe that sometimes the score of the intended model, as in this example, falls slightly and then rises again because some instances display patterns closer to other models. The inset in Fig. 7 is a magnified view of the gray band region (corresponding to Nov 4, 11pm - Nov 5, 2am) which shows the scores of the other models. It is observed that once any other model gives a positive output, its score quickly spikes, but soon after drops back to zero (shown by dotted lines for the Netatmo cam device) as the intended model Smart Things wins again.

We update the consistency score with two rates: rising on positive outputs at rate λ_r and falling on negative outputs at rate λ_f – these rates can be configured by network operators. To update the consistency score, we use a sigmoid function which is commonly used in processes like trust management [52] as it exhibits soft start and end, and is bounded within 0 and 1. Specifically, a raw score is represented by sequence $\{S_t\}$ beginning at time $t = 0$, is dynamically updated by:

$$S_t = \frac{S_{t-1} \times e^{\lambda}}{1 + S_{t-1} \times (e^{\lambda} - 1)} \tag{3}$$

where, S_{t-1} is the previous value of the estimated score, and λ is set dynamically depending on the latest output of the model (*i.e.,* $\lambda_r > 0$ for a positive output, and $\lambda_f < 0$ for a negative output). Network operators may choose the values of the two parameters λ_r and λ_f based on their preferred policy in terms of how quickly (or slowly) they want the consistency score to rise and fall. For example, a higher value for λ_r may significantly increase the consistency in a short time interval, while the low value may take a longer time to reach the desired consistency score. Depending upon the time expected T to reach to a "target score" S^* (between 0 and 1) from the mid-level score 0.50, we derive the value λ by:

$$\lambda = \frac{\log(\frac{S^*}{1-S^*})}{T} \tag{4}$$

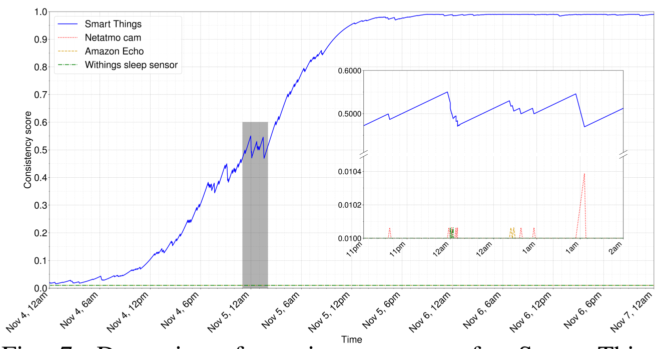


Fig. 7: Dynamics of consistency score for Smart Things instances in real-time.

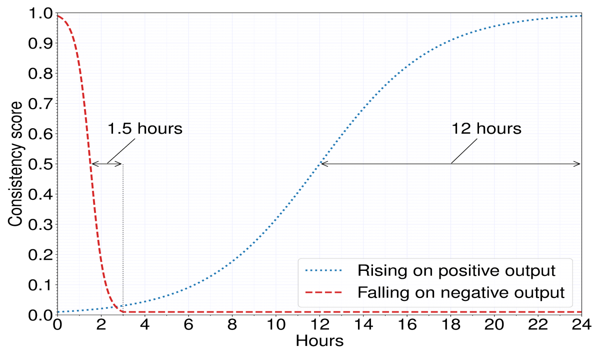


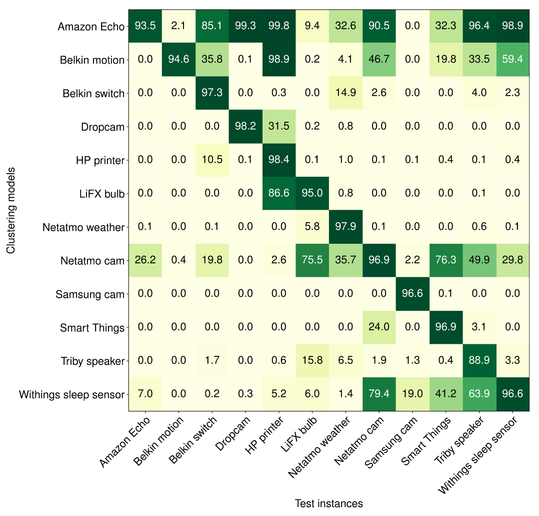
Fig. 8: Real-time update rate of consistency score – for a given model, it falls fast (from highest to lowest value in 3 hours) on continuous negative outputs, and rises slowly (from lowest to highest value in 24 hours) on continuous positive outputs.

In this paper, we choose a conservative approach whereby the consistency score rises at slower rate than it falls. The same values are used for all models, and are configured in such a way that it will take 12 hours to reach a very high score of 0.99 from a score of 0.50 ($\lambda_r = 0.0064$) in case of successive positive outputs from the model, whereas it needs only 1.5 hours to reach to a very small score of 0.01 from a score of 0.50 ($\lambda_f = -0.0511$) in case of successive negative outputs. Fig. 8 shows two sample curves of consistency score using these λ values, each monotonically rising and falling on successive positive and negative outputs, respectively. Both curves saturate (reaching the ultimate values of 0 and 1) in infinite time, and change very slowly beyond certain levels, *i.e.,* above 0.99 for the rising curve, and below 0.01 for the falling curve. In other words, entering into these regimes can stifle the agility of our real-time monitoring (especially for detecting attacks in real-time).

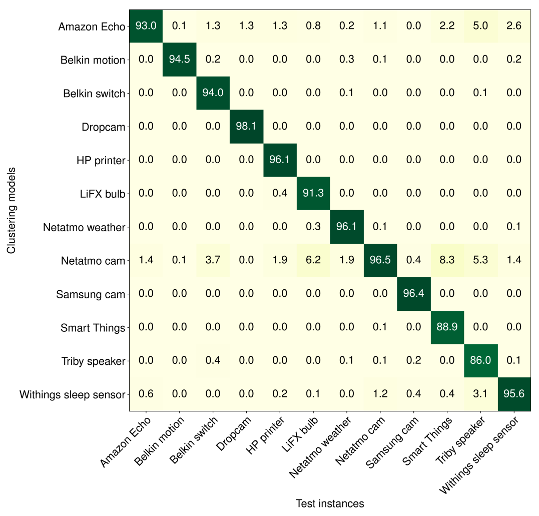
For example, it will take at least 45 minutes to fall from 0.999 to 0.99 (half the time needed to fall from 0.99 to 0.50). Similarly, it takes 6 hours to rise from 0.001 to 0.01. Therefore, we cap the scores at 0.99 and 0.01 as the saturation (minimum and maximum) levels, and also initialize each score by $S_0 = 0.01$.

D. Monitoring Phases

For monitoring the behavior of each IoT device we consider two phases: (1) the initial phase, and (2) the stable phase. The



(a) Raw outputs of clustering models.



(b) Refined outputs after conflict resolution.

Fig. 9: Confusion matrix of device classification: (a) raw outputs of clustering, (b) refined outputs after conflict resolution.

initial phase begins when a device connects to the network for the first time (discovery), and can therefore be considered to be "unlabeled". During this phase, our inference engine (shown in Fig. 6) aims to determine the device type (classification) by asking all of the existing models. To achieve this aim, every instance of the device traffic is fed to all models in real-time, and their outputs are obtained. If multiple models give a positive output, then our conflict resolution process (§IV-B) is applied to select one of those models as a winner. During the initial phase, the consistency score of all winner models for a device is tracked until the consistency score of one of these winner models reaches an acceptable level (*i.e.,* , a threshold value chosen by the network operator, say 0.90) at which time the device type is deemed to be verified. At this point the device is labeled as the corresponding known class (*i.e.,* device type), its intended model is determined, and its state changes from the initial phase to the stable phase. In the stable phase, the inference engine uses only the intended model to monitor the real-time behavior of the (labeled) device. However, as described below, the consistency score of the intended model is used to detect changes in behavior of that device.

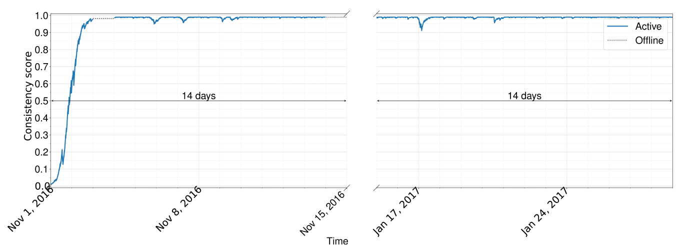
V. Performance Evaluation

We now evaluate the efficacy of our inference engine. First, we evaluate the performance of one-class models and conflict resolution in selecting an intended model for a given device during its initial phase of monitoring. Once the device type is classified (with a sufficiently high level of consistency), we next demonstrate behavioral changes using temporal consistency score of the intended model during its stable phase of monitoring. Finally, we show the efficacy of our inference engine in detecting practical abnormal scenarios that can be encountered in a real network such as service outages, firmware updates and volumetric attacks. We also compare our one-class classification with a multi-class classification method.

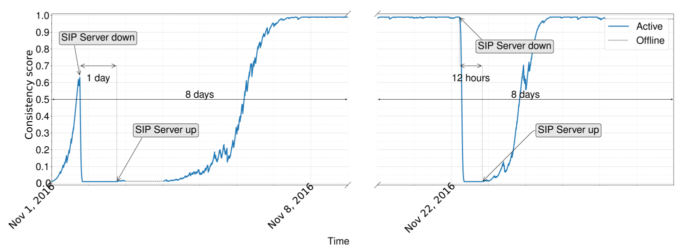
A. Device Classification

We begin by evaluating the performance of device classification using a subset of test instances from DATA1 corresponding to only two weeks spanning from 1-Nov-2016 to 14-Nov-2016). We show in Fig. 9 the resulting confusion matrix of device type classification respectively before and after resolving conflicts as described above. Every clustering model (listed in rows) is presented by test instances of IoT devices (listed in columns). For a given cell of the matrix, the value of that cell indicates the percentage of instances (from the device in corresponding column) that receive a positive output from the model in the corresponding row.

Starting from the raw outputs in Fig. 9(a), it can be seen that all models correctly detect the majority of instances from their own class as shown by the diagonal elements of the

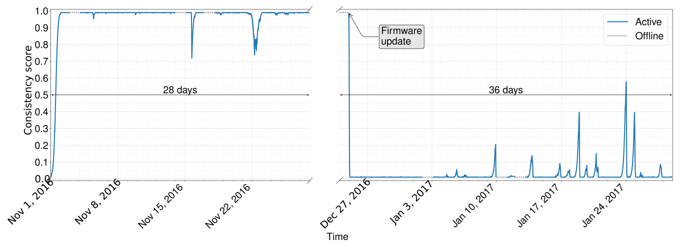


(a) Belkin switch.

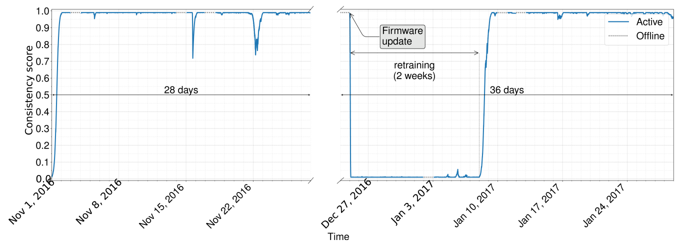


(b) Triby speaker.

Fig. 10: Time-trace of consistency score for normal behavior in: (a) Belkin switch, (b) Triby speaker.



(a) Original model (Dropcam).



(b) Re-trained model (Dropcam).

Fig. 11: Time-trace of consistency score due to firmware upgrade in Dropcam traffic: (a) original model, (b) re-trained model.

confusion matrix – except for the Triby speaker with 88.9%, the others display more than 93.5% of correct detection (*i.e.,* true positives). However, we observe that models incorrectly detect device instances from other classes (*i.e.,* false positives) as shown by the non-diagonal elements of the confusion matrix. For example, the models for the Amazon Echo and Belkin motion devices incorrectly give a positive output to 99.8% and 98.9% of instances of the HP printer. Considering the raw outputs of the various models, 70% of test instances were detected by more than one model (in addition to their expected model), and 2% of test instances were not detected by any of the models. We, next, select the winner model for each test instance using the model confidence-levels (§IV-B).

Fig. 9(b) shows the same confusion map, but after conflict resolution process. Comparison with Fig. 9(a) clearly demonstrates a significant enhancement in performance of IoT device type classification by selecting the model with the highest confidence. Note that the average false positive rate has reduced to less than 0.4% while the average true positive rate is 93.9%.

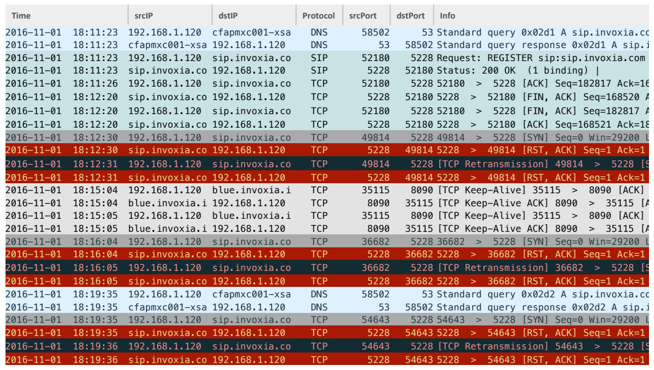
It is also observed that the conflict resolver slightly reduced the rate of true positives for almost all models. The Smart Things device is impacted more compared to other models by experiencing a drop from 96.9% to 88.9% in its true positive rate, largely because the Netatmo camera model which gives positive output with high confidence for 8.3% of Smart Things instances. Focusing on the model for the Netatmo camera, we found that its clusters overlap with a number of clusters of several other IoT devices, including the Belkin switch, LiFX, Smart things, and Triby speaker devices, and hence resulting in false positives. This is mainly due to aperiodic behavior of the Netatmo camera which is event-triggered – the camera transmits video to its cloud server whenever it recognizes a human face or detects a motion. As a result, it displays a wider range of activity patterns over longer time scales, overlapping with the traffic patterns of other devices. For example, the average byte rate of incoming NTP traffic of Netatmo camera over an 8-min timescale can take a value anywhere from 0 to 700 bytes-per-min. Such a wide range overlaps with the value range of the same attribute for the LiFX bulb (varying between $8-12$ bytes-per-min) and the Smart Things device (varying between $15-25$ bytes-per-min). Note that overlaps in IoT traffic patterns are expected and cannot be avoided, especially when we aim to classify a large number of different device types. Overlaps can be resulted due to various reasons such as event-triggered activities, or the use of common services or servers (especially in devices from the same manufacturer). We address this issue by monitoring the consistency of classification models over time, filtering out occasional overlapping incidents.

B. Detecting Behavioral Change

In the previous subsection, we showed the efficacy of our system in classifying individual device instances using an array of models. Once an IoT device has been classified as a particular type of IoT device, its activity is monitored in real-time. The models can be used to highlight behavioral changes by tracking dynamics of their consistency scores (§IV-C). To demonstrate this, we use a longer portion of DATA1, spanning dates from 01-Nov-2016 to 31-Mar-2017.

Fig 10(a) shows the consistency score generated by our inference engine for traffic instances of the Belkin switch device over a period between Nov 1, 2016 and Jan 28, 2017. The score ramps up to 99% within the first 48 hours before the device goes offline for a day, as shown by dashed gray lines, and then comes back online on Nov 4. After that, device instances are consistently detected by the intended model, and hence the consistency score remains high with only minor changes over this extended period.

Fig 10(b) illustrates a scenario where a consistency score drops for a relatively short period of time (due to a temporary change of behavior in traffic of a Triby speaker device), and then rises afterwards. Manual inspection of packet traces corresponding to these temporary drops of the score revealed that a remote SIP server (`sip.invoxia.com`), with which Triby speaker keeps a continuous TCP connection, was responding



```
Time                 srcIP           dstIP            Protocol  srcPort  dstPort Info
2016-11-01  18:11:23 192.168.1.120   cfapmxc001-xsa    DNS      58502       53 Standard query 0x02d1 A sip.invoxia.cc
2016-11-01  18:11:23 cfapmxc001-xsa  192.168.1.120     DNS         53    58502 Standard query response 0x02d1 A sip.i
2016-11-01  18:11:23 192.168.1.120   sip.invoxia.co    SIP      52180     5228 Request: REGISTER sip:sip.invoxia.com
2016-11-01  18:11:23 sip.invoxia.co  192.168.1.120     SIP       5228    52180 Status: 200 OK  (1 binding) |
2016-11-01  18:11:26 192.168.1.120   sip.invoxia.co    TCP      52180     5228 52180  >  5228 [ACK] Seq=182817 Ack=16
2016-11-01  18:12:20 sip.invoxia.co  192.168.1.120     TCP       5228    52180 5228  >  52180 [FIN, ACK] Seq=168520 A
2016-11-01  18:12:20 192.168.1.120   sip.invoxia.co    TCP      52180     5228 52180  >  5228 [FIN, ACK] Seq=182817 A
2016-11-01  18:12:20 sip.invoxia.co  192.168.1.120     TCP       5228    52180 5228  >  52180 [ACK] Seq=168521 Ack=18
2016-11-01  18:12:30 192.168.1.120   sip.invoxia.co    TCP      49814     5228 49814  >  5228 [SYN] Seq=0 Win=29200 L
2016-11-01  18:12:30 sip.invoxia.co  192.168.1.120     TCP       5228    49814 5228  >  49814 [RST, ACK] Seq=1 Ack=1
2016-11-01  18:12:31 192.168.1.120   sip.invoxia.co    TCP      49814     5228 [TCP Retransmission] 49814  >  5228 [S
2016-11-01  18:12:31 sip.invoxia.co  192.168.1.120     TCP       5228    49814 5228  >  49814 [RST, ACK] Seq=1 Ack=1
2016-11-01  18:15:04 192.168.1.120   blue.invoxia.i    TCP      35115     8090 [TCP Keep-Alive] 35115  >  8090 [ACK]
2016-11-01  18:15:04 blue.invoxia.i  192.168.1.120     TCP       8090    35115 [TCP Keep-Alive ACK] 8090  >  35115 [A
2016-11-01  18:15:05 192.168.1.120   blue.invoxia.i    TCP      35115     8090 [TCP Keep-Alive] 35115  >  8090 [ACK]
2016-11-01  18:15:05 blue.invoxia.i  192.168.1.120     TCP       8090    35115 [TCP Keep-Alive ACK] 8090  >  35115 [A
2016-11-01  18:16:04 192.168.1.120   sip.invoxia.co    TCP      36682     5228 36682  >  5228 [SYN] Seq=0 Win=29200 L
2016-11-01  18:16:04 sip.invoxia.co  192.168.1.120     TCP       5228    36682 5228  >  36682 [RST, ACK] Seq=1 Ack=1
2016-11-01  18:16:05 192.168.1.120   sip.invoxia.co    TCP      36682     5228 [TCP Retransmission] 36682  >  5228 [S
2016-11-01  18:16:05 sip.invoxia.co  192.168.1.120     TCP       5228    36682 5228  >  36682 [RST, ACK] Seq=1 Ack=1
2016-11-01  18:19:35 192.168.1.120   cfapmxc001-xsa    DNS      58502       53 Standard query 0x02d2 A sip.invoxia.cc
2016-11-01  18:19:35 cfapmxc001-xsa  192.168.1.120     DNS         53    58502 Standard query response 0x02d2 A sip.i
2016-11-01  18:19:35 192.168.1.120   sip.invoxia.co    TCP      54643     5228 54643  >  5228 [SYN] Seq=0 Win=29200 L
2016-11-01  18:19:35 sip.invoxia.co  192.168.1.120     TCP       5228    54643 5228  >  54643 [RST, ACK] Seq=1 Ack=1
2016-11-01  18:19:36 192.168.1.120   sip.invoxia.co    TCP      54643     5228 [TCP Retransmission] 54643  >  5228 [S
2016-11-01  18:19:36 sip.invoxia.co  192.168.1.120     TCP       5228    54643 5228  >  54643 [RST, ACK] Seq=1 Ack=1
```

Fig. 12: Wireshark capture of Triby speaker packets showing outage of SIP server.

with ACK/RST packets during those periods, as highlighted by red rows in the wireshark screenshot of Fig. 12, indicating that the expected SIP service was not operational. Other services for the Triby speaker were functioning normally.

The behavior of a device can also change permanently due to a firmware upgrade. Fig. 11(a) illustrates this scenario for a Dropcam device. We can see that the consistency score of Dropcam model remains high throughout Nov 2016 until Dec 5, 2016 when the device goes off-line, as shown by dotted gray lines. A couple of slight drops are observed on Nov 15 and Nov 22, but are restored fairly quickly (infrequent misclassification is not surprising due to minor overlaps between clusters of various models). However, once the Dropcam device comes back online on Dec 25, the score drops steeply to its lowest possible value of 0.01 and stays at that level during the whole of January. The score sometimes jumps up to 0.20 or even 0.30, but it quickly drops back to its minimum value. The corresponding packet traces of the Dropcam device were manually inspected, and its behavior was found to have permanently changed. Dropcam network activity is dominated by a single TLS connection which the device establishes with its cloud server (`nexus-us1.dropcam.com`) [9], typically sending packets of size 156 bytes and receiving packets of size 66 bytes. Manual inspections revealed that the rate of packets for this flow changed in both directions (while packet sizes remained unchanged), resulting in a decrease of upstream bitrate from 1896 bps to 1120 bps and a decrease of downstream bitrate from 584 bps to 424 bps. Firmware upgrades of the Dropcam device are performed automatically when it reboots. Once the firmware upgrade of the device was confirmed, the Dropcam model was retrained using an additional 2-weeks of data (between Dec 25, 2016 and Jan 07, 2017) following the firmware upgrade. Adding new instances to the training dataset resulted in an increase in the number of PCA components (from 9 to 11) for the Dropcam device, while the number of clusters remained the same. Fig. 11(b) shows how the consistency score returns back to its perfect level after augmenting the Dropcam model with attributes of the upgraded firmware – the score is shown by dashed a gray line during the two week re-training period.

TABLE III: Summary of DATA2: device instances and clustering parameters – benign traces for training, and mix of benign and attack traces for testing.

	Instance count		**Unsupervised classifier parameters**	
Device	Training (4-week)	Testing (4-week)	# Principal components	# clusters
Amazon Echo	27102	27510	20	256
Belkin motion	38229	37216	13	256
Belkin switch	21038	12689	17	256
Chromecast	17396	24316	17	512
Hue bulb	17329	25830	19	512
LiFX bulb	25903	26181	15	256
Netatmo cam	13529	10639	16	256
Samsung cam	38227	36747	15	256
TPlink switch	38211	35205	14	128
iHome	37866	35761	16	128

C. Detecting Attacks

We now evaluate the performance of our inference engine against attack traffic traces. For this evaluation, we use our second dataset DATA2. It consists of well-annotated attack and benign traffic corresponding to 10 real IoT devices namely the Amazon Echo, TPlink switch, Belkin motion sensor, Belkin switch, LiFX bulb, Netatmo camera, Hue bulbs, iHome switch, Samsung Smart camera, and Google Chromecast. These attacks on IoT devices are in various types including directly targeted attacks such as ARP spoofing, TCP SYN flooding, Fraggle (UDP flooding), and Ping of Death, and also reflection attacks such as SNMP, SSDP, TCP SYN, and Smurf, each type at three different rates (*i.e.,* low: 1 packet-per-second, medium: 10 pps, and high: 100 pps). Additionally, attacks are diversified in terms of the location of attacker being remote or local to victim/reflector IoT devices. In total, DATA2 contains 200 attack sessions, each lasts for about 10 minutes.

Note that DATA2 was collected from a different IoT environment, and therefore we need to re-generate our clustering models using data of IoT behaviors specific to that environment. From DATA2 traces, we choose 4 weeks worth of data (*i.e.,* May 28-31, Jun 8-19, Oct 9-19) containing pure benign traffic for training, and the remaining 4 weeks (Jun 1-8, Jun 19-20, Oct 19-Nov 10) containing mix of benign and attack traffic for testing. Table III shows the number instances (training and testing) for each device type as well as parameters of the corresponding clustering models.

Let us now evaluate the efficacy of individual models against a traffic mix of attack and benign instances. We measure four metrics: fraction of attack instances getting negative output (TN: true negatives), fraction of benign instances getting negative output (FN: false negatives), fraction of benign instances getting positive output (TP: true positives), and fraction of attack instances getting positive output (FP: false positives). On average, our models yield acceptable performance metrics – TN, FN, TP, and FP equals to 92.0%, 6.1%, 93.9%, and 8.0%, respectively.

Focusing on attacks, Table IV and V show detection rate of our models for direct and reflection attacks. Each attack type-location scenario shown in columns (*e.g.,* ARP Spoofing R$\rightarrow$D: remote attacker launching direct spoofing attack to device) is repeated three times at rates 1, 10, and 100 pps.

Starting from Table IV corresponding to direct attacks, it can be seen that the average detection rate for ARP Spoofing, Ping of Death, TCP SYN flooding, and Fraggle is 84.3%, 89.4%, 91.3%, and 86.2%, respectively. However, we observe that the Belkin motion model displays a poor performance in detecting attacks launched from from local attackers (highlighted cells). For example, the detection rates of Ping of Death, TCP SYN flooding, and Fraggle are 43.3%, 13.3%, and 3.0% respectively. This is mainly because Belkin motion typically communicates with its mobile App locally by UPnP messages reporting current state of the sensor, and hence local attacks

TABLE IV: Detection rate (%) of direct attacks: per model (in rows) and per attack-type (in columns).

Attack	**ARP Spoofing**			**Ping of Death**			**TCP SYN**						**Fraggle**					
Attacker	**L→D**			**L→D**			**L→D**			**R→D**			**L→D**			**R→D**		
Rate	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**
Amazon Echo	100	75	88										100	100	100	100	100	100
Belkin motion	80	70	80	70	50	10	10	30	0	75	100	95	0	0	10			
Belkin switch	100	90	100	100	100	100	100	100	100	100	100	100						
Chromecast	80	80	90				100	100	100	50	70	90						
Hue bulb	70	90	90	90	90	100	100	100	100	100	100	100						
LiFX bulb	88	100	100	100	100	100							100	100	100	100	100	100
Netatmo cam	25	88	75				100	100	100	80	100	100						
Samsung cam	100	90	90	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
TPlink switch	60	70	60	100	100	100	100	100	100	100	100	100						
iHome	100	100	100															

are not seen so abnormal by the corresponding model – soon we will further investigate and address this issue.

Moving to Table IV to check the performance of models against reflection attacks, we see the rate of detection for Smurf, SNMP, SSDP and TCP SYN reflection attacks on average is 99.1%, 58.8%, 88.5%, and 92.0%, respectively. Again, we observe that some of broadcast attacks (*i.e.,* SSDP reflection attack on Chromecast) and local attacks (*i.e.,* TCPsyn on Belkin motion and SNMP on Samsung cam) are missed. This is primarily because we only monitor local traffic targeted to IoT devices (§III-A), and hence broadcast traffic and reflected outgoing local traffic get missed. It is important to note that models of Belkin motion and Hue bulb are detecting local SSDP reflection attacks (L→D→L) only because these two devices have limited processing power, and hence under local SSDP attacks their normal operation (activity pattern of other flows) gets impacted leading to abnormal behavior.

Enhancing Detection Rate: As discussed earlier in this section, models in general perform well for a mix of benign and attack traffic except in certain situations. Among all models, we found that the Belkin motion model does not perform well especially for attack instances, and results a relatively high FP 32.1%. To further investigate such performance, we look at its confidence-level. Fig. 13 shows the CDF of the Belkin model confidence for incorrectly classified attack instances (FP) as well as correctly classified benign instances (TP). We note that the model gives a very low confidence-level (less than 2.5%) for a majority (61%) of the FP instances while such low confidence is seen for a tiny fraction (3%) of the TP instances. Again the acceptable confidence-level will be chosen by the network operator depending on their desired sensitivity. In our case, choosing confidence threshold 2.5%, the performance metrics is significantly enhanced for Belkin model – FP is improved down to 12.5% while TP is slightly degraded (from 95.5% to 92.6%).

Such enhancement is observed across all models after filtering model outputs with confidence less than 2.5%, and thus

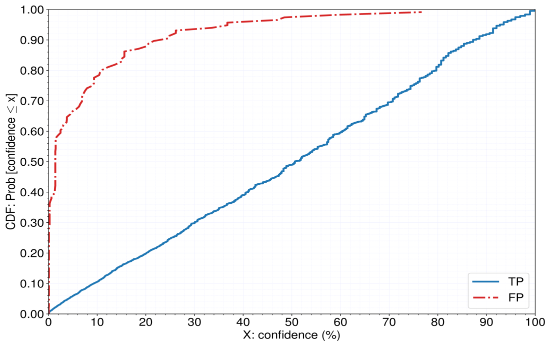


Fig. 13: CDF: distribution of confidence-level for Belkin motion instances.

overall TN, FN, TP, and FP reaches to 94.7%, 9.03%, 92.0%, and 5.3%, respectively across all models. We show in Table VI the performance of individual models after this enhancement. We can see that every model now displays acceptable value in performance metrics (high rate of true alarms and low rate of false alarms).

Performance Comparison of One-Class vs. Multi-Class: We, lastly compare the performance of our one-class classifier scheme versus previously studied multi-class classifiers (including ours [9]). For our comparison, we use Random Forest algorithm (based on decision trees) to generate and tune a multi-class model using the training instances (same as for our one-class models) from DATA2.

Before comparing the two schemes, we need our devices to operate in their stable phase of monitoring. Note that, the first two days of testing data contains pure benign traffic from all of the ten devices. This amount of data is sufficient for all of intended one-class models to be selected (consistency score of winner models exceeds our chosen threshold 0.90). In other words, every device passes its initial phase and enters into the stable phase. In the stable phase, the inference engine is expected to give negative output whenever attack traffic instances are present and generate positive output for pure benign traffic.

TABLE V: Detection rate (%) of reflection attacks: per model (in rows) and per attack-type (in columns).

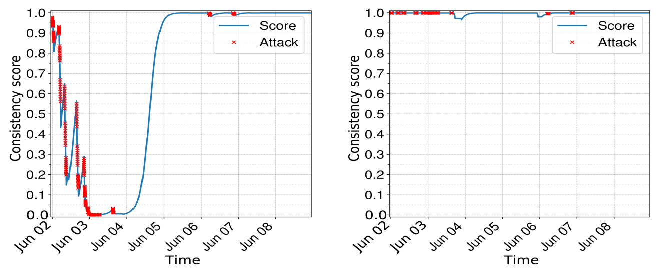
Attack	**Smurf**			**SNMP**									**SSDP**									**TcpSynReflection**					
Attacker & Victim	**L→D→L**			**L→D→L**			**L→D→R**			**R→D→R**			**L→D→L**			**L→D→R**			**R→D→R**			**L→D→L**			**R→D→R**		
Rate	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**	**1**	**10**	**100**
Amazon Echo																											
Belkin motion													100	90	80	100	100	100	100	100	100	30	30	0	85	95	100
Belkin switch																						100	100	100	100	95	100
Chromecast													0	30	0	100	100	100	100	100	100	100	100	100	40	90	100
Hue bulb	90	100	100										100	90	100	100	100	100	100	100	100	100	100	100	100	100	100
LiFX bulb	100	100	100																								
Netatmo cam																						100	100	100	100	100	100
Samsung cam	100	100	100	0	0	0	30	100	100	100	100	100										100	100	100	100	100	100
TPlink switch	100	100	100																			100	100	100	100	100	100
iHome																											

TABLE VI: Performance of one-class classifiers for mix of attack and benign traffic.

	Detected as			
	attack		benign	
	TN (%)	FN (%)	TP (%)	FP (%)
Amazon Echo	98.8	6.0	94.0	1.2
TPlink switch	95.9	5.8	94.2	4.1
Belkin motion	87.5	7.4	92.6	12.5
Belkin switch	99.2	8.1	91.9	0.8
LiFX bulb	99.3	7.3	92.7	0.7
Netatmo cam	94.6	6.0	94.0	5.4
Hue bulb	97.3	15.7	84.3	2.7
iHome	100.0	7.1	92.9	0.0
Samsung cam	92.4	6.7	93.3	7.6
Chromecast	81.9	10.2	89.8	18.1

Fig. 14 illustrates the consistency of the two schemes for a sample of traffic from Samsung smart cam during a week period with 380 instances of attack traffic – each instance worth a minute of traffic. In Fig. 14(a) we plot the real-time consistency score for Samsung smart cam during attack periods – attack instances are marked by red '×'. It can be seen that the model correctly detects attack traffic instances by giving them negative outputs, causing drop in the consistency score. We note that sometimes the consistency score keeps falling down even when attack finishes. This is because the impact of some attacks persist in attributes of a few following instances (up to 8 minutes). We can see that during intense attack periods (Jun 2 and Jun 3), the consistency score of the one-class model of Samsung camera drops to its lowest level, well highlighting a significant change of behavior in device traffic.

On the other hand, it is seen in Fig. 14(b) that the multi-class model is insensitive to attacks. For multi-class model, we consider outputs as positive whenever they indicate the expected label with a high confidence from the model (*i.e.,* above a threshold of 80% which is obtained from our previous work [9]) and as negative otherwise ($<$ 80% confidence). The consistency score of the model remains high during the whole week, and does not noticeably get affected by attack instances, as shown in Fig. 14(b). Even though the Random Forest model gives negative output to some attack instances (34.6%), but each negative output is immediately followed by a sequence of positive outputs keeping the real-time consistency score at a very high value – incorrectly suggesting that the device behaves normally.



(a) one-class clustering model. (b) multi-class decision-tree model.

Fig. 14: Performance comparison for traffic of Samsung cam during attack: (a) one-class model, (b) multi-class model.

We observe that the one-class clustering model has by far more ability to highlight (detect) anomalies in device behavior compared to the multi-class decision-tree model – detection rate of 92.6% compared to 34.6%. We also note that multi-class model correctly classifies 98.0% of benign instances (TP) while this metric is slightly lower (94.7%) for one-class model.

Note that these two approaches are fundamentally different in their way of modeling: one-class models are generative (learn distribution of each class) while multi-class models are discriminative (learn decision boundary between various classes). As a result, one-class models become sensitive to changes in any attribute while multi-class models become sensitive to changes in only discriminative attributes.

VI. Conclusion

Real-time traffic monitoring is of paramount importance for network operators who manage a diverse range of IoT devices. In this paper, we have developed a modular classification scheme to infer type of IoT devices from their network behavior using a set of clustering models. We trained individual classification models to classify IoT devices and detect the cyber-attacks from network traffic. We augmented our machine learning-based models by developing a conflict resolver and dynamically updating the consistency state of device models. Finally, we evaluated the efficacy of our system by applying it to traffic traces from twelve IoT devices, and demonstrated its ability to detect behavioral changes and cyber-attack with an overall accuracy of more than 94%.

References

[1] A. Sivanathan *et al.*, "Inferring IoT Device Types from Network Behavior Using Unsupervised Clustering," in *IEEE LCN 2019*, Osnabrück, Germany, Oct 2019.

[2] Cisco, "Cisco 2017 Midyear Cybersecurity Report," Tech. Rep., 2017.

[3] A. Hamza *et al.*, "Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity," in *Proc. ACM SOSR*, San Jose, California, USA, Apr 2019.

[4] H. Habibi Gharakheili *et al.*, "Network-Level Security for the Internet of Things: Opportunities and Challenges," *Computer*, vol. 52, no. 8, pp. 58–62, Aug 2019.

[5] B. Cheng, S. Zhao, J. Qian, Z. Zhai, and J. Chen, "Lightweight Service Mashup Middleware With REST Style Architecture for IoT Applications," *IEEE TNSM*, vol. 15, no. 3, pp. 1063–1075, Sep 2018.

[6] B. Cheng, M. Wang, S. Zhao, Z. Zhai, D. Zhu, and J. Chen, "Situation-Aware Dynamic Service Coordination in an IoT Environment," *IEEE/ACM TON*, vol. 25, no. 4, pp. 2082–2095, Aug 2017.

[7] Y. Meidan *et al.*, "Detection of Unauthorized IoT Devices Using Machine Learning Techniques," *arXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1709.04647

[8] J. Ortiz *et al.*, "DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices," in *Proc. ACM IoTDI*, Montreal, Quebec, Canada, 2019.

[9] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE TMC*, no. 8, pp. 1745–1759, Aug 2018.

[10] J. Zhu, "A Secure and Automatic Firmware Update Architecture for IoT Devices," Working Draft, IETF Secretariat, Internet-Draft, March 2018. [Online]. Available: https://tools.ietf.org/id/draft-zhu-suit-automatic-fu-arch-00.html

[11] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE CST*, vol. 18, no. 2, pp. 1153–1176, Oct 2016.

[12] Y. Meidan *et al.*, “ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis,” in *Proc. ACM SAC*, Marrakech, Morocco, April 2017.
[13] M. Miettinen *et al.*, “IoT Sentinel Demo: Automated Device-Type Identification for Security Enforcement in IoT,” in *Proc. ICDCS*, Atlanta, GA, USA, Jun 2017.
[14] H. Guo and J. Heidemann, “IP-Based IoT Device Detection,” in *Proc. ACM IoT S&P*, Budapest, Hungary, Aug 2018.
[15] R. Doshi *et al.*, “Machine Learning DDoS Detection for Consumer Internet of Things Devices,” in *2018 IEEE SPW*, San Francisco, CA, May 2018, pp. 29–35.
[16] S. Nomm and H. Bahsi, “Unsupervised Anomaly Based Botnet Detection in IoT Networks,” *IEEE ICMLA*, pp. 1048–1053, Dec 2018.
[17] A. Kumar *et al.*, “EDIMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques,” *arXiv*, 2019. [Online]. Available: http://arxiv.org/abs/1906.09715
[18] G. Gu *et al.*, “BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection,” in *Proc. of Security Symposium*, San Jose, CA, Jul 2008.
[19] S. Zhao *et al.*, “A Dimension Reduction Model and Classifier for Anomaly-Based Intrusion Detection in Internet of Things,” in *DASC*, Nov 2017.
[20] Y. Meidan *et al.*, “N-BaIoT: Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders,” *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, Jul 2018.
[21] U. A. Sandhu *et al.*, “A Survey of Intrusion Detection & Prevention Techniques,” in *ICICM*, Singapore, 2011.
[22] C. Ko *et al.*, “Execution Monitoring of Security-Critical Programs in Distributed Systems: a Specification-based Approach,” in *Proc. IEEE SSP*, Oakland, CA, USA, USA, May 1997.
[23] S. Mukkamala *et al.*, “Intrusion Detection using Neural Networks and Support Vector Machines,” in *Proc. IJCNN*, Honolulu, HI, USA,, May 2002.
[24] L. H. Yeo *et al.*, “Understanding Modern Intrusion Detection Systems: A Survey,” *arXiv*, Aug 2017. [Online]. Available: http://arxiv.org/abs/1708.07174
[25] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-time,” *Computer Network*, vol. 31, no. 23-24, pp. 2435–2463, Dec 1999.
[26] M. Roesch, “Snort - lightweight intrusion detection for networks,” in *Proc. USENIX CSA*, Seattle, Washington, Nov 1999.
[27] P. Uppuluri and R. Sekar, “Experiences with Specification-Based Intrusion Detection,” in *Recent Advances in Intrusion Detection*, Berlin, Heidelberg, 2001.
[28] J. P. Amaral *et al.*, “Policy and network-based intrusion detection system for ipv6-enabled wireless sensor networks,” in *Proc. IEEE ICC*, Sydney, NSW, Australia, Jun 2014.
[29] S. Anwar, F. Al-Obeidat, A. Tubaishat, S. Din, A. Ahmad, F. A. Khan, G. Jeon, and J. Loo, “Countering Malicious URLs in Internet-of-Thing (IoT) using a knowledge-based approach and simulated expert,” *IEEE IoT Journal*, 2019.
[30] E. Lear *et al.*, “Manufacturer Usage Description Specification,” RFC 8520, IETF Secretariat, Tech. Rep., Mar 2019.
[31] A. Hamza *et al.*, “Combining MUD Policies with SDN for IoT Intrusion Detection,” in *Proc. ACM Sigcomm workshop on IoT S&P*, Budapest, Hungary, Aug 2018.
[32] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, “Static malware detection and attribution in android byte-code through an end-to-end deep system,” *Future Generation Computer Systems*, vol. 102, pp. 112 – 126, 2020.
[33] M. Amin, B. Shah, A. Sharif, T. Ali, K.-l. Kim, and S. Anwar, “Android malware detection through generative adversarial networks,” *Transactions on Emerging Telecommunications Technologies*, 2019.
[34] D. P Vinchurkar and A. Reshamwala, “A Review of Intrusion Detection System Using Neural Network and Machine Learning Technique,” *International Journal of Engineering Science and Innovative Technology*, vol. 1, pp. 54–63, Nov 2012.
[35] R. Patel *et al.*, “A Survey and Comparative Analysis of Data Mining Techniques for Network Intrusion Detection Systems,” *IJSCE*, vol. 2, pp. 265–260, Jan 2012.
[36] L. Portnoy *et al.*, “Intrusion Detection with Unlabeled Data Using Clustering,” in *Proc. of ACM CSS Workshop on DMSA*, 2001, pp. 5–8.
[37] P. Garcia-Teodoro *et al.*, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, Feb 2009.
[38] R. Sommer and V. Paxson, “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection,” in *IEEE SSP*, Washington, DC, USA, May 2010.
[39] A. Sivanathan *et al.*, “Low-cost flow-based security solutions for smart-home IoT devices,” in *IEEE ANTS*, Bangalore, India, Nov 2017.
[40] H. Habibi Gharakheili *et al.*, “iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification,” *IEEE TNSM*, 2019.
[41] A. Sivanathan *et al.*, “Characterizing and Classifying IoT Traffic in Smart Cities and Campuses,” in *Proc. IEEE INFOCOM workshop on smart cities and urban computing*, Atlanta, USA, May 2017.
[42] H. Jiang and C. Dovrolis, “Why is the Internet Traffic Bursty in Short Time Scales?” *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 241–252, Jun 2005.
[43] F. Tegeler *et al.*, “BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection,” in *Proc. ACM CoNEXT*, Nice, France, Dec 2012.
[44] J. N. Miller, “Tutorial Review—Outliers in Experimental Data and Their Treatment,” *Analyst*, vol. 118, no. 5, pp. 455–461, 1993.
[45] D. Ruan *et al.*, *Intelligent Data Mining: Techniques and Applications*. Springer Science & Business Media, 2005, vol. 5.
[46] I. Bin Mohamad and D. Usman, “Standardization and Its Effects on K-Means Clustering Algorithm,” *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, no. 17, pp. 3299–3303, Sep 2010.
[47] C. Ding *et al.*, “K-means Clustering via Principal Component Analysis,” in *Proc. ICML*, Banff, Alberta, Canada, Jul 2004.
[48] C. H. Achen, “What Does ”Explained Variance” Explain?: Reply,” *Political Analysis*, vol. 2, p. 173–184, 1990.
[49] D. Ketchen *et al.*, “The Application of Cluster Analysis in Strategic Management Research: An Analysis and Critique,” *Strategic Management Journal*, vol. 17, no. 6, pp. 441–458, 1996.
[50] M. K. Pakhira, “A Modified k-means Algorithm to Avoid Empty Clusters,” *IJRTE*, vol. 1, no. 1, pp. 220–226, May 2009.
[51] A. McCallum and K. Nigam, “A Comparison of Event Models for Naive Bayes Text Classification,” *AAAI/ICML-98 WLTC*, pp. 41–48, 1998.
[52] M. Firdhous *et al.*, “A Memoryless Trust Computing Mechanism for Cloud Computing,” in *Networked Digital Technologies*, Berlin, Heidelberg, 2012, pp. 174–185.

Arunan Sivanathan received his B.Sc (Eng) in 2012 from the Faculty of Engineering, University of Peradeniya specializing in Computer Engineering. Soon after, he joined the University of Jaffna, Sri Lanka as a Lecturer. He received his Ph.D. in Electrical Engineering and Telecommunications from UNSW Sydney, Australia in 2020. His primary research interests include cyber-security of Internet of Things and data analytics on machine-to-machine communication.

Hassan Habibi Gharakheili received his B.Sc. and M.Sc. degrees of Electrical Engineering from the Sharif University of Technology in Tehran, Iran in 2001 and 2004 respectively, and his Ph.D. in Electrical Engineering and Telecommunications from UNSW in Sydney, Australia in 2015. He is now a lecturer at UNSW Sydney. His current research interests include programmable networks, learning-based networked systems, and data analytics in computer systems.

Vijay Sivaraman received his B. Tech. from the Indian Institute of Technology in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000. He has worked at Bell-Labs as a student Fellow, in a silicon valley start-up manufacturing optical switch-routers, and as a Senior Research Engineer at the CSIRO in Australia. He is now a Professor at the University of New South Wales in Sydney, Australia. His research interests include Software Defined Networking, network architectures, and cyber-security particularly for IoT networks.