Efficient IoT Traffic Inference: from Multi-View Classification to Progressive Monitoring

ARMAN PASHAMOKHTARI, UNSW Sydney, Australia
GUSTAVO BATISTA, UNSW Sydney, Australia
HASSAN HABIBI GHARAKHEILI, UNSW Sydney, Australia

Machine learning-based techniques have proven to be effective in IoT network behavioral inference. Existing works developed data-driven models based on features from network packets and/or flows, but mainly in a static and ad-hoc manner, without adequately quantifying their gains versus costs. In this paper, we develop a generic architecture that comprises two distinct inference modules in tandem, which begins with IoT network behavior classification followed by continuous monitoring. In contrast to prior relevant works, our generic architecture flexibly accounts for various traffic features, modeling algorithms, and inference strategies. We argue quantitative metrics are required to systematically compare and efficiently select various traffic features for IoT traffic inference.

This paper[1] makes three contributions. **(1)** For IoT behavior classification, we identify four metrics, namely cost, accuracy, availability, and frequency, that allow us to characterize and quantify the efficacy of seven sets of packet-based and flow-based traffic features, each resulting in a specialized model. By experimenting with traffic traces of 25 IoT devices collected from our testbed, we demonstrate that specialized-view models can be superior to a single combined-view model trained on a plurality of features by accuracy and cost. We also develop an optimization problem that selects the best set of specialized models for a multi-view classification; **(2)** For monitoring the expected IoT behaviors, we develop a progressive system consisting of one-class clustering models (per IoT class) at three levels of granularity. We develop an outlier detection technique on top of the convex hull algorithm to form custom-shape boundaries for the one-class models. We show how progression helps with computing costs and the explainability of detecting anomalies; and, **(3)** We evaluate the efficacy of our optimally-selected classifiers versus the superset of specialized classifiers by applying them to our IoT traffic traces. We demonstrate how the optimal set can reduce the processing cost by a factor of six with insignificant impacts on the classification accuracy. Also, we apply our monitoring models to a public IoT dataset of benign and attack traces and show they yield an average true positive rate of 94% and a false positive rate of 5%. Finally, we publicly release our data (training and testing instances of classification and monitoring tasks) and code for convex hull-based one-class models.

CCS Concepts: • **Security and privacy → Network security**; • **Computing methodologies → Machine learning**.

Additional Key Words and Phrases: IoT traffic classification, behavior monitoring, anomaly detection, optimization.

[1]This submission is an extended and improved version of our paper presented at the ACM ETSec IoT workshop [54].

Authors' addresses: Arman Pashamokhtari, UNSW Sydney, Australia, a.pashamokhtari@unsw.edu.au; Gustavo Batista, UNSW Sydney, Australia, g.batista@unsw.edu.au; Hassan Habibi Gharakheili, UNSW Sydney, Australia, h.habibi@unsw.edu.au.

XXXX-XXXX/2023/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Arman Pashamokhtari, Gustavo Batista, and Hassan Habibi Gharakheili. 2023. Efficient IoT Traffic Inference: from Multi-View Classification to Progressive Monitoring. 1, 1 (September 2023), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

New IoT devices are emerging each week in the market and are increasingly being deployed in a variety of smart environments like homes, buildings, enterprises, and cities. Many of these networked assets often lack in-built security measures and are ridden with security holes, as amply demonstrated by prior research work [38, 70], posing grave risks to the security and privacy of personal and organizational data gleaned from sensors [72], but also provides a launching pad for attacks to systems within [71] and outside [40] the organization – witness the attacks by a University's vending machines on its own campus network [8], and IoT botnets DDoS-attacking DynDNS [58]. In fact, 57% of IoT devices can become targets of medium to high impact cyber-attacks [53]. A survey [19] revealed that about two-thirds of enterprises do not have adequate visibility over IoT devices connected to their network, leading to at least five security incidents per year.

Obtaining visibility into IoT network activities is essential [53] to securing these networks. IoT devices often perform a limited set of tasks. They come with power and computing constraints, and their network activity is relatively less but more distinct than traditional IT devices [69]. This creates an opportunity to develop models [7, 15, 18, 25, 26, 30, 42, 47, 51, 62, 67, 69, 75, 76, 79] that learn IoT benign behavior relatively well. About 85% of organizations [13] are planning to utilize artificial intelligence or machine learning models to develop monitoring systems for their assets.

Automatic IoT traffic inference is often achieved in three distinct phases: (1) *discovery* where the identity (*e.g.,* MAC and/or IP address) of connected devices is determined (via passive analysis or importing an asset inventory list from active scans), (2) *classification* or *identification* that maps device identities to their types/roles (*e.g.,* `device D` $\longmapsto$ `Camera`), and (3) *monitoring* where their network behavior is continuously measured and checked against expected patterns, ensuring their cyber health or flagging any anomalous behaviors. Device discovery has been long studied in computer networks (especially using active scans for IT asset management), and there are tools like Nmap [39] for this purpose. Given the overheads and disruption risks introduced by active scanners [5], non-invasive or passive[2] inference methods sound more appropriate for IoT asset management. Some works like [7, 18, 42, 47, 62, 69, 75, 76, 79] focused on traffic classification while others like [15, 25, 26, 30, 51, 67] developed methods for traffic monitoring. A recent work [74] attempted to dynamically leverage the combined capabilities offered by these two approaches (*i.e.,* active and passive) to characterize IoT asset behaviors.

Machine learning algorithms have been widely applied to packet/flow-based traffic features for capturing recurring patterns of IoT devices. Most existing works used a single-combined-view approach (training a single model on a plurality of traffic features). Single-combined-view models have some drawbacks, like dealing with missing features [16], demanding more data [54], and overfitting [80]. On the other hand, a multi-view approach employs a collection of models ("views"), each specializing in certain aspects of data. For example, in image recognition, views can be images captured with varying angles, texture data, or color information of an object [80]. In the context of IoT traffic inference, the single-combined-view approach is dominantly used for developing inference models [7, 15, 25, 42, 47, 51, 67, 75, 79].

Prior research work developed IoT traffic inference models based on various traffic features, some are packet-based, and others are flow-based. The existing literature tends to incorporate traffic features solely based on their prediction accuracy for modeling purposes. We argue that more

[2]Analytical techniques are applied to network traffic from a SPAN (Switched Port Analyzer) port.

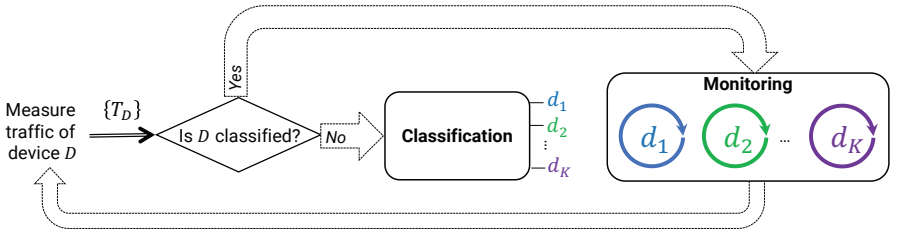


Fig. 1. A high-level overview of our system architecture for IoT traffic inference.

factors, like how prevalent is a feature in the network traffic across device classes, how frequently certain features emerge to measure, or how expensive it would be to compute features, should also be considered for developing inference models for IoT devices. For IoT behavioral monitoring specifically, models are applied in a static manner – all required network traffic telemetries are collected and checked in one shot for anomalies. Static approaches do not scale cost-effectively. We instead consider a dynamic approach that progresses in multiple stages, ranging from coarse-grained to fine-grained. Progressive monitoring assists IoT network operators in finding the cause of anomalies (explainable inference), particularly at scale with a large number of connected devices.

In our previous work [54], we experimented with the multi-view approach combined with a progressive inference only for classification. This paper builds upon and improves our previous conference paper [54] by focusing on both classification and monitoring objectives, performed in tandem as shown in Fig. 1. We enhance our multi-view approach for classification in two ways: (a) systematically quantifying and balancing the overall gains versus costs of each view (corresponding to traffic features) by four specific metrics, and (b) optimally selecting the best set of views. Moreover, we develop a novel architecture and set of models at varying granularity for progressively monitoring IoT behaviors on the network. Also, the methods and metrics we develop are extensively evaluated with larger and more diverse datasets (publicly released [56]). Our specific contributions are as follows.

- The **first** contribution (§3) focuses on IoT traffic classification. Unlike existing works that merely used the accuracy metric, we identify four metrics (including accuracy, cost, availability, and frequency) that help us systematically compare the characteristics of specific traffic (packet/flow) features. In §3, we will demonstrate how the accuracy metric can overshadow the importance of other key metrics for quantifying the impact of traffic features. By experimenting with traffic traces of 25 IoT devices collected from our testbed, we develop a specialized model for each traffic feature and demonstrate how specialized models outperform a single-combined-view model (trained on a plurality of features) in prediction accuracy and overall computing costs. We also develop a method to select an optimal set of specialized models for multi-view classification.
- The **second** contribution (§4) focuses on IoT behavioral monitoring. We develop a progressive architecture of one-class models at three levels of granularity, improving computing costs as well as the explainability of detecting anomalous traffic patterns. We employ the convex hull algorithm and develop an outlier detection technique on top of it for tightening the benign boundaries in our one-class models.
- Finally, the **third** contribution (§5) begins by evaluating the efficacy of our optimally-selected classifiers versus the superset of specialized classifiers by applying them to our IoT traffic traces. Our results show how the optimal set of models can reduce the processing cost by a factor of six with insignificant impacts on the prediction accuracy. We also apply our monitoring models to a public IoT dataset of benign and attack traces and show they yield an

average true positive rate of 94% and a false positive rate of 5%. Our data (training and testing instances of classification and monitoring tasks) and enhancement code for the convex hull algorithm are publicly released [56] to the research community.

The next section (§2) summarizes prior relevant work and highlights our novelty, and §6 concludes our paper and presents directions for future work.

2 RELATED WORK

Prior works on IoT traffic inference, by the type of network telemetry, can be categorized into two groups: (1) packet-based telemetries are extracted from packet contents – examples are domain names from DNS packets, TCP options from SYN packets, and User-Agent header from HTTP packets; and, (2) flow-based telemetries are statistical features computed from specific or collection of flows *e.g.,* number of TCP packets, the byte count of NTP traffic, or average inter-arrival time of DNS queries. Authors of [15, 47, 51] employed purely packet-based features, while work in [25, 67] used purely flow-based features, and works in [42, 69, 75] combined packet-based and flow-based features. Though they may supply some unique signatures (enabling a more accurate inference), packet-based features typically require more processing power as they need (deep/shallow) packet inspection. In contrast, flow-based features can be obtained using cost-effective programmable switches or flow extraction tools like NetFlow [12] or IPFIX [34]. We will have a detailed discussion about these two types of telemetry in §3.1.

For IoT traffic inference, we view a general system architecture, as illustrated in Fig. 1, consisting of two distinct modules: classification of device behaviors and monitoring expected behaviors. In what follows, we review how existing works realize these two objectives.

2.1 Classifying IoT Behaviors

Behavioral classification (also known as identification) essentially aims to create a mapping between device identities and their class (*e.g.,* type, or make/model). Device identity is referred to an identifier like MAC or IP address that is unique for each device on the network. In some cases where the network traffic is collected after crossing a middlebox (*e.g.,* a NAT-enabled router), device identities may be overwritten and obscured. A body of works [21, 46, 55, 62] addressed the classification of IoT device behaviors using obscured traffic (*e.g.,* post-NAT measurement and inference). This paper assumes no middlebox obscures network traffic, and we obtain device identities like MAC or IP addresses directly from the measured traffic. Next, we summarize some seminal works in the literature and their fundamental contributions to IoT traffic classification.

IoT Sentinel [47] identified more than twenty packet-based features; most are binary features representing well-known protocols, while others include packet size, IP options, and transport-player port numbers. Upon detection of a device type, it queries public vulnerability repositories like CVE [49] to find vulnerabilities associated with the detected device types and then isolates the vulnerable devices in the network. DEFT [75] used both packet-based and flow-based features to train a supervised and an unsupervised model to determine IoT devices connected to home networks from an Internet service provider's perspective. The supervised model is distributed across home routers, performing the classification process. Suppose the supervised model is not confident about a device type. In that case, it sends the corresponding telemetries to a centralized cloud-based controller, where the unsupervised model distinguishes the device type by clustering techniques, re-trains, and redistributes the supervised model to the home routers.

Similar to DEFT, AuDI [42] performs IoT traffic clustering using both supervised and unsupervised learning methods. AuDI uses flow-based time-series features to generate fingerprints of IoT devices by relying only on periodic autonomous traffic (like NTP and DNS) that is likely unaffected by

human interactions. The authors used Fourier transform and signal auto-correlation to filter periodic flows. Authors in [69] used a combination of packet-based and flow-based features to classify IoT behaviors on the network. Their packet-based features include server port numbers, domain names, and TLS cipher suites. The flow-based features were the statistical measure of volume, duration, and rate of flows, inactivity period, and frequency of DNS and NTP queries. They (similar to us) used a multi-view approach by developing a separate model for packet-based and a model for flow-based features. However, no systematic attempt is made on how a subset of these features could be considered given certain objectives and constraints (efficient/optimal selection of different views).

2.2 Monitoring Expected IoT Behaviors

Monitoring is another pillar of IoT traffic inference that looks for deviations from expected behaviors due to firmware/software updates, service outages, or cyber-attacks. Works in [25, 26] used Manufacturer Usage Description (MUD) [35] to profile benign network activities of a given IoT device type. MUD profiles are expected to be supplied by device manufacture res, but they can be automatically generated [27] at run-time by analyzing IoT traffic traces. MUD profiles essentially specify network metadata (*i.e.,* a list of access control entries, each indicating transport-layer protocol and port numbers as well as the endpoint IP address or domain name) – no specification of traffic contents, rates, and/or volumes. Thus, they are vulnerable to data exfiltration and volumetric attacks. Work in [57] attempts to develop a baseline (expected patterns) for the information content of packet payloads (whether unencrypted, encoded, or encrypted) in each MUD flow of an IoT device. Authors in [25] demonstrated how machine learning models augment foundational MUD profiles (given as input [26]) to flag deviations from expected activities of individual flows (detecting network-based volumetric attacks that still conform to MUD profiles).

Some existing works used purely benign traffic [51, 67], and others used a mixture of benign and attack traffic [15] to train machine learning models. Using attack traffic to train models is proven [73] to limit the detection capability to those known attacks, making models vulnerable to zero-day attacks (similar to signature-based anomaly detectors). Authors of [67] developed one-class clustering models, one for each device type, purely trained on benign traffic instances of each device type. Trained models were used for both classification and change detection (monitoring) purposes. We will train a collection of one-class models per each IoT type only for monitoring (after the device class is determined).

Our novelty: Prior works developed classifiers and anomaly/change detectors to passively analyze IoT network traffic. However, there are some missing pieces. **First**, the majority of existing works [6, 9, 29, 33, 42–45, 47, 50, 65, 66, 69, 75, 76, 78] just focused on one of the two tasks (classification or monitoring). Though some works [22, 27, 52, 67] developed a system that addresses both tasks, they used the same inference model for both of these purposes. This paper argues that classification and monitoring are different in nature but somewhat dependent. Hence, we best achieve them separately but in tandem. **Second**, to our knowledge, there is no work highlighting and quantifying various characteristics of traffic features except for their prediction accuracy. This paper quantitatively shows how features are unequal in prediction power, computation costs, availability, and frequency. **Third**, no existing work compared the efficacy of specialized models against that of a single combined model for IoT traffic inference. This paper demonstrates that specialized models (each trained on specific features) can outperform a single model that uses the union of those features and then develops an optimization technique to find the best set of specialized models for a multi-view inference. **Fourth**, existing works used a single-combined-view model with a static granularity level for the monitoring task. This paper develops a novel architecture that dynamically increases granularity utilizing a collection of models per device class.

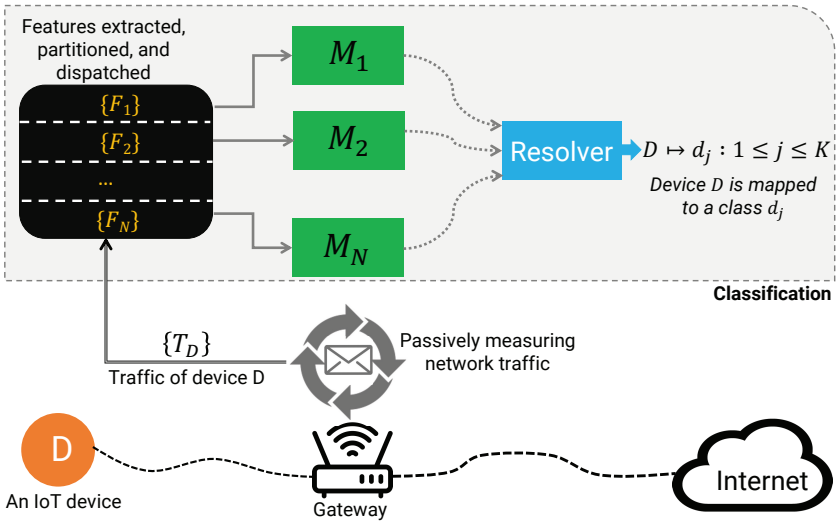


Fig. 2. Architecture of multi-view classification of IoT behaviors.

3 MULTI-VIEW CLASSIFICATION OF IOT TRAFFIC

Discovering the IoT devices connected to a network is the first step for traffic inference. Asset discovery can be made actively (a response from connected devices to probing packets [39]) or passively (the first packet sent by each connected device on the network). Let us assume a list of identities (*e.g.,* MAC/IP addresses) for the connected devices in the network is available.

As we saw earlier in Fig. 1, the inference begins with classification (determining the type of individual networked devices), followed by monitoring their network behavior which will be discussed in §4. Fig. 2 illustrates a zoomed-in version of the classification module. It is the system architecture we develop to infer the mapping between devices and their class. The input of the classification task is $\{T_D\}$, which is the network traffic of an IoT device D that we aim to predict its type. In the literature, various architectures, namely inline (actively processing original data) or out-of-band (passively analyzing copies of data), have been employed to measure device traffic $\{T_D\}$. Inline solutions are often used for critical use cases like firewalls and/or intrusion prevention, where an embedded module makes (security or management) inferences. On the other hand, out-of-band measurement is often employed for use cases like forensics or security information/event management, whereby inferences are typically made by a machine (on-prem or cloud-based) that runs corresponding software modules consuming measured traffic. For passive out-of-band measurement, traffic mirrors can be supplied by employing programmable switches [25–27], SPAN[3] ports, or TAP[4] devices [21, 42, 51]. In this paper, we configure the network gateway to mirror network packets to an on-prem machine with software modules to passively classify (in this section) and monitor (in §4) the traffic of IoT devices connected to the network. Our architecture is designed to be generic, catering to a variety of traffic features and behavioral modeling algorithms. Though we use representative features and models in this paper for evaluation, one may choose a subset of them or extend them with additional features.

For classification, we consider a multi-view approach. This means that a collection of models $(M_i : 1 \leq i \leq N)$, each specializing in a certain aspect (view) of network traffic. Examples of views can be (but are not limited to) domain names, TCP SYN/SYN-ACK signatures, or packet/byte count

[3]Switched Port Analyzer
[4]Traffic Access Point

of flows at different resolutions (*e.g.,* 3-tuple, 4-tuple, or 5-tuple). Multi-view learning comes with some benefits compared to a single-combined-view approach: (a) Having multiple distinct models helps with reducing the number of features needed to train each model and decreases the chance of overfitting [2]; and (b) In network traffic, certain features may not always be available or appear less frequently for some devices. For example, one IoT device may never use HTTP (unavailable) for network communications, or a device may generate TCP packets every minute (frequent) while it sends UDP packets once an hour (less frequent). Thus, combining features with diverse availability across heterogeneous classes of IoT assets (single-combined-view) can cause features to be missing in some of the traffic instances measured from the network, which is not desirable for machine learning algorithms [16].

We note that various modeling algorithms (multi-class, binary, one-class) come with their pros and cons in terms of training, re-training (updating), and application. For example, an advantage of multi-class models is that they can learn the patterns of different classes with a single training dataset relatively easier. However, training reliable binary (one-vs-rest) models can be practically challenging [11, 67], especially when dealing with the class-imbalanced datasets that this strategy often generates. A disadvantage of multi-class models is that when new device classes are added to the network, the entire model must be updated, perhaps with additional processing to balance the dataset. On the other hand, binary and one-class models (particularly when all intended features are embodied in one model) may seem more attractive at scale [36] as they can be updated or added independently on a per-class basis. That said, device-specific (binary or one-class) models may suffer from prediction conflicts among themselves since they are independent models. Thus, they may demand an extra processing step to unify or normalize prediction scores (confidence levels). All things considered, this work uses multi-class models for the classification phase. It is important to note that we take a different approach by employing specialized multi-class classifiers, whereby individual models do not know all traffic patterns. In other words, each multi-class classifier specializes in a unique traffic attribute that may not necessarily be present in the behavior of all device classes. We, therefore, may not always need to update all multi-class models when a new device class emerges on the network. Lastly, one may choose to implement these specialized classifiers using binary or one-class models, which is beyond the scope of this paper.

Given predictions provided by N models (a multi-view approach), we need a mechanism to consolidate results of (or perhaps, resolve conflicts among) models – we call it a "resolver" module. In this paper, we use multi-class models to minimize conflict resolution, but one may use one-class models [67] at this stage too. Different strategies, like majority voting, weighted voting, and average weighted voting, can be employed to resolve predictions obtained from N models. We will evaluate the impact and efficacy of several resolution strategies in §5. The classification phase ultimately maps each networked device D to one of K known classes d_j. We note that classical supervised learning algorithms (this paper included) aim to train a model in the closed-set world, where training and test samples share the same class space (*i.e., K* IoT devices classes). Open-set learning is a more challenging and realistic problem, where test samples from the IoT classes are unseen during training. Automatic recognition of open sets [59] is the sub-task of detecting test samples that do not come from the training, which is beyond the scope of this paper.

Needless to say that our classification architecture does not require or make any assumptions for traffic features, the type of models, or resolver techniques. For instance, one may choose a Random Forest model for TCP SYN signatures, a neural network model for domain names, and a Naïve Bayes model for TLS cipher suites. Additionally, views are extensible to incorporate other necessary or desired features.

Let us begin by identifying metrics to characterize traffic features leading to specialized views. These metrics can help us better understand the difference and quantify the efficacy of various

traffic features. Following that, we compare specialized IoT traffic classifiers against a single combined-view one. Finally, we develop an optimization problem for selecting the best set of views.

3.1 Characterization Metrics and Traffic Features

In order to assess various traffic features, we need quantitative metrics. Existing works mainly used the notion of accuracy to characterize and compare different types of features. Though it is an important factor, the accuracy metric is insufficient for a comprehensive assessment. In what follows, we identify four metrics that enable us to compare traffic features.

Cost: Measuring network traffic and extracting the required features incur processing and computing costs. Packet-based features demand packet inspection (by hardware or software). For a packet-based feature, we define cost as the average amount of CPU time required to extract it per packet. Flow-based features, on the other hand, can be obtained from programmable or software-defined networking (SDN) switches [25, 54, 68] without a need for packet inspection. Programmable/SDN switches (when programmed appropriately [24]) can provide inference engines with flow-level telemetries (packet/byte counters) at configurable granularities. Note that programmable switches come with a certain capacity of TCAM (Ternary Content Addressable Memory) which could be a limiting factor (cost) for flow-based features. Therefore, we use an approximation for the required TCAM size in bytes for flow-based features.

Accuracy: Traffic features differ in terms of the amount of information they carry. For example, some DNS domain names can be unique to a device type, whereas NTP packets typically have a size of 90 bytes (48 bytes of payload plus 42 bytes of UDP, IP, and Ethernet headers) according to the NTP standard specifications [48] as long as the optional authenticator field is not used. The amount of information can affect the quality of the predictions, which can be measured using various metrics like accuracy, precision, recall, and F1-score. This paper uses the average of correct predictions across device classes for our accuracy metric.

Availability: The heterogeneity of IoT devices makes it challenging to presume what network protocols they use. Therefore, certain traffic features may not be available for some IoT device types (classes). For example, the August doorbell cameras do not send any HTTP traffic, and Samsung smart cameras never send a User-Agent header in their HTTP requests. For a given traffic feature, we define availability as the fraction of total K device classes that we can find that feature from their network traffic.

Frequency: Although some traffic features are available for certain device classes, they may not be present all the time (appear infrequently). For instance, JA3 signature is extracted from TLS handshake, which occurs once per TLS session [54] whereas, Ethernet packet and byte counts are always present for measurement as long as the device is active on the network. Relying on less frequent features can increase the response time of the inference model – due to waiting time before they emerge in the traffic. For a given feature, we define frequency as the fraction of all time epochs across K classes it is present in training instances. Note that epoch is a configurable parameter. In our experimental evaluations, we use 1-min epochs for classification and 5-min epochs for monitoring.

To make our discussion more tangible, we next focus on a representative set of network telemetries, including four packet-based and three flow-based feature sets (for classifying IoT behaviors) to showcase how characterization metrics work. We also look at how specialized-view models (each trained on a set of features) perform versus a single-combined-view model (trained on a combination of all features).

Largely inspired by prior works [54, 62, 67, 68], we consider four packet-based traffic features as follows.

Domain Name: The name of cloud servers that IoT devices communicate with can be used to identify device types [62, 69]. Domain names (*e.g.,* "`chat.hpeprint.com`" for HP printers or "`broker.lifx.co`" for LiFX lightbulbs) are extracted from DNS requests sent by IoT devices, and they are less likely (compared with IP addresses) to change over time [23].

SYN signature: TCP standard [17] defines an optional field at the end of TCP packets called `TCP Options`. TCP options include a list of fields, some of which may have a value assigned to them. For example, the Maximum Segment Size option must have a value, while the SACK Permitted option (selective acknowledgment) has no value. Our recent work [54] showed that IoT devices often display unique and consistent SYN signatures, revealing their type.

HTTP User-Agent: HTTP packets may contain a User-Agent header that allows clients to introduce themselves to the servers they connect. The User-Agent header can highlight the browser and/or operating system so the server can customize its response to clients accordingly. Work in [33] employed the User-Agent header to distinguish IoT device types.

JA3 signature: TLS connections are initiated by a `Client Hello` handshake packet. This packet contains information about the client's capabilities and preferences for the TLS connection, including the TLS version, cipher suites, a list of extensions, elliptic curves, and elliptic curve formats. Work in [63] developed a signature format based on these values to detect malware activities in TLS traffic. For the first time (to our knowledge), this paper uses JA3 signatures to determine the class of IoT devices.

To automate the process of computing packet-based features (identified above), we employed port and protocol numbers to filter DNS (`IP proto=17`, `UDP/53`), SYN signature (`IP proto=6`), HTTP (`IP proto=6, TCP/80`), and TLS (`IP proto=6, TCP/443`) packets from a mix of traffic. Packets that match an intended port/protocol number (*e.g.,* `IP proto=6, TCP/443`) but do not supply the expected feature (*e.g.,* TLS JA3 signature in this example) are discarded. Note that port and protocol numbers are not among the features we use for IoT traffic inference (classification or monitoring) in this paper—our classifiers are not trained on port/protocol numbers since they cannot be used as reliable features for traffic classification [36].

In what follows, we explain three sets of flow-based features (some employed by prior works [67, 68]) that capture distinct patterns of IoT behaviors on the network. Note that "↓" and "↑" signs, respectively, refer to the incoming direction of traffic "to" the networked IoT device and the outgoing direction of traffic "from" the networked IoT device.

Datalink: This set captures statistical measures of total network activities per each device unit (discovered on the network), including four specific features: ↓total packet count, ↓total byte count, ↑total packet count, ↑total byte count. These coarse-grained features are obtained from statistics of two flow entries (one matching source MAC address and one matching destination MAC address of packets against a given device MAC address), covering all traffic of layer-2 and above, including (but not limited to) ARP, IPv4, IPv6, UDP, and TCP.

Transport: This set increases the granularity of statistics by extending its focus to transport-layer protocols. For this set, four flow entries are needed that match: (a) source/destination MAC address and TCP transport protocol[5] (two flows) and (b) source/destination MAC address and UDP transport protocol[6] (two flows). We obtain a total of eight medium-grained features:

[5]IP protocol number = 6.
[6]IP protocol number = 17.

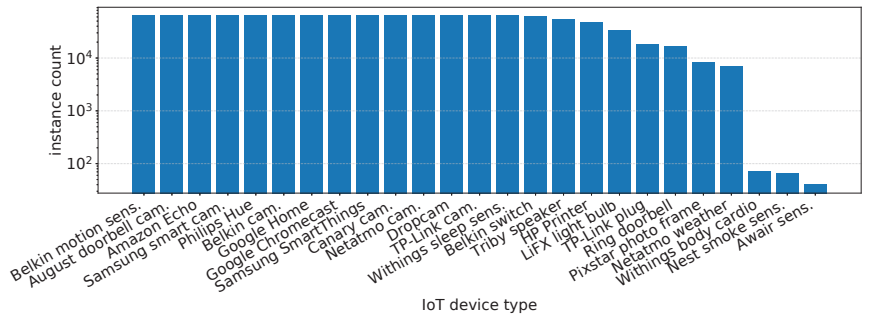


Fig. 3. The number of instances per each IoT device type in our benign dataset.

↓TCP packet and byte count, ↑TCP packet and byte count, ↓UDP packet and byte count, ↑UDP packet and byte count, per each device unit (a given device MAC address).

Application: This set focuses on fine-grained features specific to network activities of three application-layer protocols, namely TCP/80 (a proxy for HTTP), TCP/443 (a proxy for HTTPS), and UDP/53 (a proxy for DNS). Similar to what was discussed above for the other two sets, each protocol demands two flow entries (matching TCP/UDP based on IP protocol, a corresponding port number, source/destination MAC address), and provides four features meaning a total of 12 application-layer features.

It is important to note that all the flow-based features discussed above can be obtained concurrently from a programmable switch using multiple flow tables, each table corresponding to a specific flow granularity (one for datalink, one for transport, and one for application).

3.2 Quantifying the Efficacy of Specialized-View Classifiers

We now quantify the four characteristic metrics of features identified above using the traffic traces of our IoT testbed. We collected PCAP traces (consisting of about 100 million packets exchanged locally and remotely) on our testbed serving 25 consumer IoT devices (a unit per each device type) during February and March 2020. IoT devices selected for our testbed were among the popular ones in the Australian consumer market at that time. A subset of these devices can be found in a typical smart home environment. Our IoT devices include eight cameras (August doorbell camera, Samsung smart camera, Belkin camera, Canary camera, Netatmo camera, Dropcam, TP-Link camera, Ring doorbell), six sensors (Belkin motion sensor, Nest smoke sensor, Netatmo weather station, Awair air quality sensor, Withings sleep sensor, Withings body cardio scale), two power switches (TP-Link plug, Belkin switch), two bulbs (LiFX light bulb and Philips Hue), five entertainment/assistant devices (Google Chromecast, Pixtar photo frame, Triby speaker, Amazon Echo, Google Home), a smart hub (Samsung SmartThings), and a printer (HP printer). We use a script written in Golang to analyze the collected PCAP files and discretize them into 1-minute epochs (configurable). The script runs on a machine using an 8-core Intel Core i9 CPU, 32GB of memory, and storage of 1TB. Fig. 3 illustrates the number of instances (1-minute epochs) per each device type in our benign dataset. Most IoT devices were highly active (with thousands to tens of thousands of instances) during the capture period; however, three devices, namely the Withings body cardio scale, Nest smoke sensor, and Awair air sensor, displayed less activity (with less than 100 instances) during the data collection period.

Our Golang script measures the average CPU time needed for extracting packet-based features of each packet. For flow-based features, the cost is an estimated amount of TCAM required to monitor the activities of each device unit connected to the network. For a given set of flow-based features,

Table 1. Characteristics of feature sets we quantified by applying specialized classifiers to our data.

		Packet-based Views				Flow-based Views		
		Domain name	SYN sign.	User-Agent	JA3 sign.	Datalink	Transport	Application
Accuracy	μ	88%	85.9%	100%	98%	94.4%	97.2%	97.0%
	σ	29%	33%	0	5%	15%	9%	7%
	min	3%	0%	100%	75%	27%	55%	69%
Cost		127 ns	220 ns	234 ns	285 ns	12 bytes	28 bytes	54 bytes
Availability		96%	96%	36%	88%	100%	96%	96%
Frequency		28%	42%	34%	15%	100%	97%	62%

the amount of TCAM is proportional to the number of flow rules inserted into the programmable switch. Furthermore, the number of matching fields is another factor determining the amount of required TCAM. Let us approximate the total byte size of TCAM required by each set of flow-based features to monitor the network activities of a single IoT device. The size of each matching field comes from the standard size of the respective field. We note that a MAC address is six bytes, an IP protocol field is one byte, and a transport-layer port number field takes two bytes of memory. Therefore, the TCAM cost for datalink, transport, and application flow features are estimated as 12 bytes (two flows of 6 bytes each), 28 bytes (four flows of 1 plus 6 bytes), and 54 bytes (six flows of 2 plus 1 plus 6 bytes), respectively. Note that increasing the granularity (from datalink to application) requires more matching fields and hence more TCAM memory. We note the cost is measured in different units for packet-based versus flow-based features. One may attempt to unify the cost measures by converting CPU times and TCAM sizes to dollar costs. However, this is beyond the scope of this paper. We will use a simpler cost proxy for optimization in §3.3 in order to unify packet and flow-based costs.

Classification Dataset: During each 1-min epoch, we extract the traffic features discussed above for individual active devices. We use the MAC address to identify connected devices and associate traffic features with them. We maintain a list of measured values for packet-based features and compute packet/byte counters for flow-based features during 1-min epochs. Finally, we export the traffic features (list of packet values and flow counters) of a device per epoch to a CSV file row.

We split our entire processed data into three CSV files: (1) `TRAIN` dataset, containing instances from the month of February that we use to quantify the cost, availability, and frequency metrics, and also to train machine learning models (specialized ones and the single-combined one); (2) `TEST1` dataset, containing instances from the first ten days of March that are used to measure the accuracy metric for the rest of this section; and, (3) `TEST2` dataset, containing instances from March 11 to 15 that are used to evaluate the performance of the multi-view classification in §5. We publicly release these three CSV files to the research community [56].

Given four sets of packet-based features and three sets of flow-based features, we train a Random Forest model (proven to be effective in learning patterns from IoT traffic [47, 61, 68, 69]) per each feature set, meaning a total of seven specialized models. Also, for packet-based features, we employ the bag of words technique, which is a well-known method to handle categorical data in machine learning applications [32, 37, 69]. To obtain the baseline measure of accuracy for specialized models, we apply them to `TEST1` dataset. For each model, we compute three measures of accuracy: average accuracy per IoT class (μ), the standard deviation of accuracy across IoT classes (σ), and the minimum accuracy across IoT classes (*min*).

Performance of Specialized-View Classifiers: Table 1 summarizes the four characteristic metrics of our traffic features when we applied their corresponding specialized classifiers to our data. Each row is color-coded in which yellow cells highlight weaker performance, while greener

cells show stronger/better performance. Note that the color interpretation varies across metrics, *e.g.,* higher average accuracy, lower standard deviation of accuracy, lower cost, and higher availability are considered as better results and hence are highlighted with greener color.

Starting with the domain name model (under packet-based views), we see an average accuracy of 88% with relatively large variations across classes that is mainly due to certain domain names shared across different device types. Manual investigations of the **TRAIN** dataset revealed that Belkin camera, Belkin motion sensor, and Belkin switch communicate with the same domain of their manufacturer (Belkin: "**belkin.com**") and CDN service provider (CloudFront: "**cloudfront.net**"). We also found that devices like Google Home and Google Chromecast perform reverse DNS lookups for eight specific IP addresses, which result in DNS queries with a domain name of format "**<IPaddress>.in-addr.arpa**". Finally, we observe that queried domain names are available in the traffic of almost every (96%) IoT device and seem to be relatively predictive of IoT classes, but DNS packets are only seen in their traffic just less than 30% of the time (not highly frequent).

The specialized model trained on SYN signatures gives an accuracy slightly lower than the domain name model. We manually verified that this is again mainly attributed to the shared SYN signatures in our **TRAIN** dataset. We found ten IoT device types shared their SYN signature with another device type. Similar to what we saw above in domain names, Belkin devices display identical SYN signatures in their network traffic. For example, the Belkin switch emits three unique SYN signatures: two were found in SYN-ACK packets sent locally in response to Samsung SmartThings, Belkin camera, and Belkin motion sensor, and one in SYN packets sent to access cloud-based services remotely. From these signatures, the first two (local) are shared with the Belkin camera and the Belkin motion sensor, and the third one (remote) is shared with the TP-Link camera, leaving no unique SYN signature for the Belkin switch. After manually investigating the traffic traces of the TP-Link camera, we found no common cloud servers with those of the Belkin switch. We think the shared SYN signature (remote) is probably attributed to using similar firmware or software libraries in the TP-Link camera and the Belkin switch.

The two models trained on User-Agent headers and JA3 signatures yield fairly high accuracy (*i.e.,* 98-100%) for those classes in which these features are available (36% for User-Agent and 88% for JA3). Interestingly, no common User-Agent or JA3 signatures were found (by manual verification) across all IoT classes. The reason for an imperfect accuracy (98%) of the JA3 model is that the Philips Hue lightbulb comes with two different JA3 signatures in the **TEST1** dataset, of which one was not seen in the **TRAIN** dataset. The two signatures are used when the lightbulb communicates with two different IP addresses in the Google cloud. We manually verified that the contacted server in the **TRAIN** dataset was located in the Netherlands, and the one that was newly seen in the **TEST1** dataset was located in the US.

Moving to flow-based views, we see those three specialized models perform relatively well in terms of accuracy but cannot beat User-Agent and JA3 models. As expected, flow-based features are more available and frequent compared to their packet-based counterparts. The accuracy (particularly by measures of σ and min) increases steadily in granularity. Fine-grained application models outperform coarse-grained datalink models. Although a slight decrease (0.2%) in the average accuracy (μ) is seen from transport models to application models, the standard deviation and min measures highlight notable improvements.

Table 1 highlights the fact that none of our specialized models (and their corresponding features) is perfect by all four metrics. User-Agent and JA3 models seem highly accurate but not desirable by the metrics of availability and frequency. Flow-based models perform moderately better. However, all of them showed a relatively low accuracy in predicting the class of the Nest smoke sensor (the minimum value). However, the domain name model perfectly classified (100% accuracy) the DNS packets of the Nest smoke sensor. This highlights the fact that a reliable traffic classification for a

growing range of device types demands collective specialization – several models, each measuring and analyzing a unique aspect of the network traffic. Now, the question is "*how can/should one best employ these specialized features/models for IoT traffic inference?*". In what follows, we train a single combined model on all features discussed above and evaluate its performance against individual specialized models.

Evaluating Performance of Classifiers (Specialized versus Single-Combined-View): With the specialized models having their performance evaluated individually and compared with each other, let us look at how they compete with a single-combined model. To have a fair comparison, we train a Random Forest model on all the seven features: domain names, SYN signatures, User-Agents, JA3 signatures, and ↓↑ packet/byte counts for Datalink, Transport (TCP, UDP), Application (HTTP, HTTPS, and DNS) traffic flows, for the single-combined-view classifier. Like we did earlier in this section, we use the `TRAIN` and `TEST1` datasets for training and testing of the combined-view model.

As the combined-view model includes all features shown in Table 1, its cost becomes the sum of the cost of each feature set, meaning about 866ns CPU time and 94 bytes TCAM memory. This cost is far higher than any of the specialized models, as reported in Table 1. Moving to accuracy, the combined-view model gives an average accuracy (across all 25 IoT classes) of about 96% with a standard deviation of 9%. The lowest class accuracy (*min*) of the combined-view model is 55% for the Nest smoke sensor. Comparing these measures with those reported in Table 1, it can be clearly seen that three specialized-view models, namely User-Agent, JA3 signature, and Application models, outperform the combined-view model, even though their features are a subset of those used to train the combined-view model. This could be attributed to many instances with missing-value features due to the variability of features' availability and frequency. Missing values are known to be detrimental to the performance of machine learning models [31, 77].

3.3 Optimal View Selection

With some specialized-view models outperforming the single-combined-view model, we now aim to improve inference quality and efficiency by selecting the optimal set of specialized models (realizing a multi-view inference). The optimal set aims to minimize the total computation cost while it meets a certain level of accuracy, which is explained in detail in what follows. Needless to say that with sufficient computing and processing powers (CPU and TCAM) and relaxing the cost constraint, one can employ all the specialized models we have discussed so far. However, two considerations are of utmost importance in real practice (particularly at scale): (1) processing costs are always capped; therefore, only a subset of specialized models can be employed to meet constraints, and (2) improving the prediction power (accuracy of classification) realized by a multi-view approach must be balanced against other factors and additional costs. For example, IoT device types that generate HTTP User-Agent headers can be perfectly classified by the corresponding model. However, those types are a subset of devices that generate JA3 signatures, which can also be classified perfectly by the JA3 model. In this scenario, no accuracy gain will be realized if we select the User-Agent model along with the JA3 model.

To formulate an optimization problem, one may attempt to minimize the total cost and/or maximize the prediction accuracy. An important factor to note here is the cost unit is different for packet-based features (CPU time) compared to that of flow-based features (TCAM size). Therefore, incorporating various costs into a complete optimization requires some forms of unification and/or adaptation (*e.g.,* converting CPU and TCAM usage to Dollar cost that can be directly used in the objective function), which we leave for future work. This paper demonstrates the possibility and impact of approximate optimization. We simplify our objective function by minimizing the number of specialized models considered. Let $X = [x_1, x_2, \ldots, x_N]$ denote a vector of binary values, where

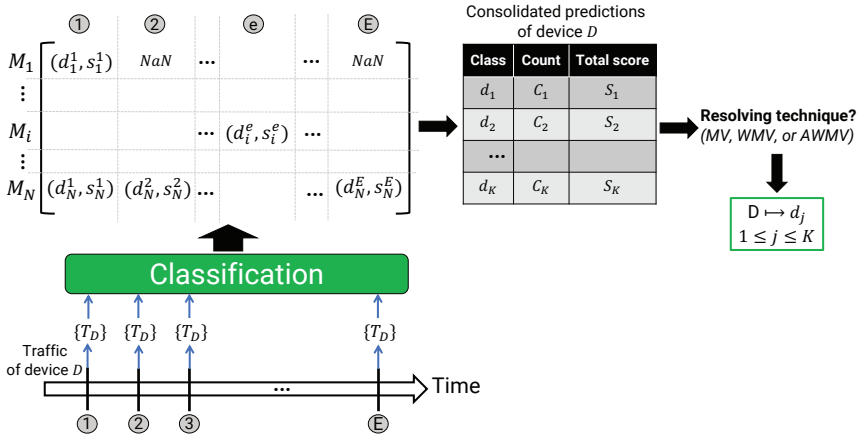


Fig. 4. Resolution method for a multi-view traffic classification of IoT device D.

$x_i = 1$ indicates the specialized-view model M_i, where $i \in [1, N]$, is selected for inference, and $x_i = 0$ otherwise. Therefore, the objective function is given by:

$$\min \sum_{i=1}^{N} x_i \tag{1}$$

In terms of constraints, every IoT class is expected to realize a prediction accuracy of more than a configurable threshold. That way, a minimum prediction quality is secured across individual classes by incorporating necessary specialized models. The threshold value can be set specifically on a per-class basis. In this paper, we choose a global threshold (denoted by A) across all classes. Therefore, each specialized model is considered either feasible (producing the desired accuracy) or infeasible (not delivering the desired accuracy) per each known class, determined from prior validation/testing on labeled data. Let $F = [f_{i,j}]_{N \times K}$ denote the feasibility matrix of N models across K IoT device classes.

$$f_{i,j} = \begin{cases} 1 & \text{if model } M_i \text{ is feasible (accuracy} \geq A) \text{ for class } d_j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where, $i \in [1, N]$ and $j \in [1, K]$. Our constraints, therefore, can be stated by:

$$\forall j \in [1, K], \sum_{i=1}^{N} x_i . f_{i,j} \geq 1 \tag{3}$$

In other words, this constraint essentially checks all elements in the vector of $X \times F$ are non-zero. In §5, we show how this optimization problem can be mapped to a convex problem and solved efficiently using off-the-shelf solvers such as CVXPY tool [14], initialized by data from our `TEST1` dataset. Having our problem defined as a convex optimization problem is attractive as convex problems offer polynomial-time solutions as opposed to non-convex ones that are NP-Hard in general. We will show that our optimally-selected multi-view will incur a sixth of the total cost of all specialized models with almost the same level of prediction accuracy.

3.4 Resolving Multi-View Predictions

With our multi-view approach, specialized models work independently and in parallel. Each model makes its prediction whenever the necessary features become available. For example, in an epoch, a connected device may generate a DNS query but no TCP SYN packet is sent. Given models are often imperfect (may make incorrect and/or less confident classification), a robust strategy is we collect predictions (from various specialized models for a given device D) over a period of time (say, six hours) and resolve discrepancies among models and across predictions for a final prediction the class for D.

Fig. 4 illustrates how multi-view predictions are resolved for a given device D. The device traffic T_D is measured every epoch time and presented to N specialized models to make their inference (the green classification box). In total, device D can receive a maximum of $N \times E$ raw predictions within E epochs, represented by the matrix on the top left of Fig. 4. Note a cell with value "`NaN`" highlights a model (in rows) that is unable to predict at the corresponding epoch (in columns) due to missing features. The prediction (d_i^e, s_i^e) is a two-tuple, consisting of the predicted class d_i^e by model M_i at epoch e with a corresponding confidence score s_i^e.

At the end of epoch E, raw predictions of device D are consolidated into a chart with K rows and three columns. The first column represents unique classes d_j ($j \in [1, K]$) various models may classify the device. The second column shows the count (C_j) of raw predictions for each class d_j. The third column is the total sum of scores in raw predictions for each class d_j, computed by:

$$S_j = \sum_{e=1}^{E} \sum_{i=1}^{N} s_i^e \quad \text{if } d_i^e = d_j \tag{4}$$

We acknowledge that one may want to take a more conservative approach, considering raw predictions with scores higher than their desired level [55]. In this paper, we do not filter predictions.

In ideal situations, we would expect to see a single row of non-zero counts in the consolidated predictions chart, meaning D is consistently mapped to one class by all specialized models. However, finding D mapped to two or more classes (*e.g.,* $D \longmapsto d_1$ and $D \longmapsto d_2$) is a reasonably likely outcome in practice. Therefore, resolving those conflicting predictions becomes important. In this paper, we consider three resolution strategies discussed next.

Majority Voting (MV): This method considers the most popular label as the final result. In other words, regardless of the classification score, the class with the largest count (in the second column of the consolidated chart) is the winner. For example, assume at the end of the resolution period, we obtain $D \longmapsto d_1$ hundred times, $D \longmapsto d_2$ ninety times, and $D \longmapsto d_3$ eighty times from our models. In this case, MV selects d_1 as the final prediction.

Weighted Majority Voting (WMV): This method primarily promotes confident predictions. In other words, regardless of counts, the class with the largest total score (in the third column of the consolidated chart) is the winner. Suppose in the example above, the total score for $D \longmapsto d_1$ is 75, for $D \longmapsto d_2$ is 80, and for $D \longmapsto d_3$ is 75. In this case, WMV infers d_2 as the device type of D.

Average Weighted Majority Voting (AWMV): This method considers popularity and confidence by taking the ratio of the total score to the count. In other words, the class with the largest average score (the third column divided by the second column of the consolidated chart) is the winner. In our examples above, the average weight of $D \longmapsto d_1$ is 0.75, and $D \longmapsto d_2$ is 0.89, and $D \longmapsto d_3$ is 0.93. In this case, AWMV selects d_3 as the final label.

In §5, we will show how these resolving methods affect the accuracy of the classification phase.

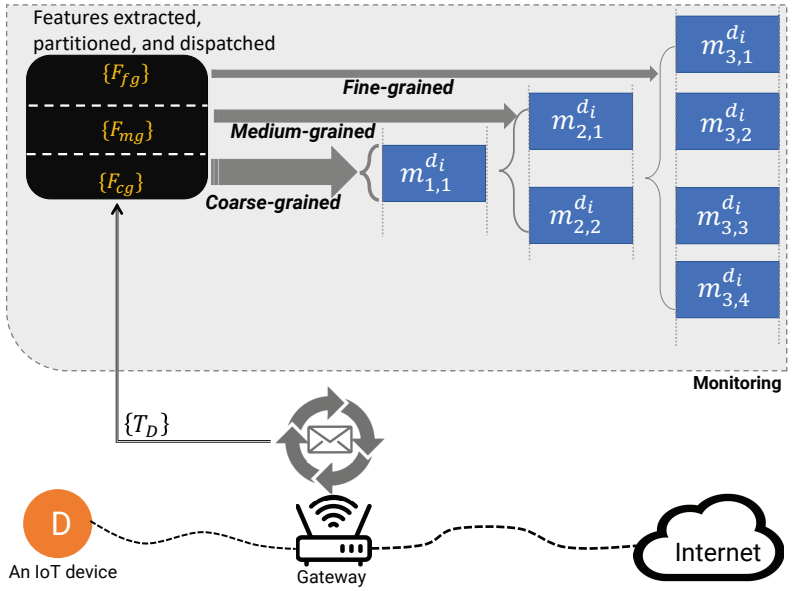


Fig. 5. Architecture of progressively monitoring IoT behaviors with three levels of granularity.

4 PROGRESSIVE MONITORING OF IOT NETWORK BEHAVIORS

We saw earlier in Fig. 1 that the monitoring phase commences once the class of device D is determined. This phase is a continuous process for validating the behavior of classified IoT devices against their benign profile.

Threat Model: The behavior of an IoT device may change [67] or deviate from its expected patterns due to reasons like malware infection [28], volumetric network attacks [25], or even legitimate changes (which may warrant updating the corresponding profile/model) [27]. Volumetric attacks send a large number of packets to target IoT devices for purposes such as draining their battery, disabling their functionality, or utilizing their reflection power [41] to overwhelm a victim server on the Internet. In §5.2, we evaluate our progressive monitoring for attacks such as Ping-of-Death, SYN attack, SYN reflection attack, and UDP DDoS.

With the device class determined, we can employ models trained on expected benign behaviors specific to that class (one-class models) for monitoring. At this stage, our objective is to infer from device traffic continuously and close to real-time; therefore, we incorporate traffic features with the highest measure of availability and frequency. That said, one may choose to employ some additional models that make inferences at slower timescales (catering to less frequent features). We recall from Table 1 that pack-based features are relatively less available and less frequent compared to their flow-based counterparts. Hence, focusing only on those packet-based features can come at the risk of missing real-time short-lived anomalies in the behaviors of connected devices. Flow-based features, instead, are available for real-time monitoring. Therefore, we choose to develop our monitoring inference models based on the flow-based features.

Fig. 5 illustrates the architecture of the monitoring stage of our IoT traffic inference. Similar to what we discussed in the previous section, the monitoring is done passively by a software component running on a general computer fed by the mirror of IoT network traffic from the network gateway. Note that the classification and monitoring modules can collocate on the same computer but operate separately. Given IoT device D mapped to class d_i, a number of device-specific

one-class models denoted by m^{d_i} are trained and employed to ensure D behaves according to its expected patterns; if not, anomalies are flagged by one or more of those models.

It can be seen in Fig. 5 that our architecture advocates an approach whereby monitoring progresses in stages, each with models of specific granularity (from coarse-grained to medium-grained to fine-grained) to manage processing costs at scale. Given IoT class d_i, let $m^{d_i}_{x,y}$ denote model number y at granularity level of x, where $1 \leq x \leq 3$ in this paper. Progression can be triggered in different ways. For example, one may choose to increase the granularity (more expensive monitoring) whenever coarser models flag anomalies for a specific device D (while other devices continue to be monitored by coarse-grained as long as they conform to their expected patterns). Another may want to frequently check the traffic of certain devices against all or just fine-grained models depending upon their available computing resources.

It is important to note that, similar to our classification architecture (§3), our monitoring architecture is flexible and generic in terms of its features, models, and progression. For example, at the finest level, it allows for the choice of application flows (*e.g.,* HTTPS and DNS versus SSH, HTTPS, and DNS). Even in an environment where a device type communicates over a non-standard protocol (*e.g.,* TCP/6000), a one-class model specific to that application flow (service) can be trained and added to this layer. In this paper, we consider and experiment with a three-layer architecture, but one can freely choose more or fewer layers at different granularities.

At each granularity, a set of one-class models (trained on a corresponding set of features) will be applied. Coarse-grained models are trained on expected behaviors at an aggregate level, while behaviors across certain protocols (transport and/or application layers) are analyzed as granularity progresses. In this paper, we use models across the three layers as follows.

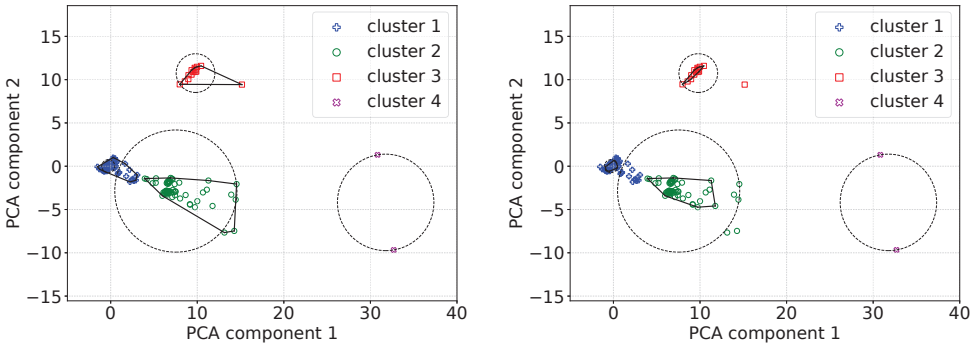
Coarse-grained: For this layer, we use a model $m^{d_i}_{1,1}$ that is trained on an aggregate measure of traffic activities, namely the four features of datalink flows (*i.e.,* $\downarrow\uparrow$ total packet/byte counts of a given MAC address D) discussed in §3.1.

Medium-grained: For this layer, we use two models: $m^{d_i}_{2,1}$ trained on the feature of TCP flows (*i.e.,* $\downarrow\uparrow$ TCP packet/byte counts), and $m^{d_i}_{2,2}$ trained on the feature of UDP flows (*i.e.,* $\downarrow\uparrow$ UDP packet/byte counts).

Fine-grained: For this layer, we use four models: $m^{d_i}_{3,1}$ trained on $\downarrow\uparrow$ HTTP packet/byte counts, $m^{d_i}_{3,2}$ trained on $\downarrow\uparrow$ HTTPS packet/byte counts, $m^{d_i}_{3,3}$ trained on $\downarrow\uparrow$ DNS packet/byte counts, $m^{d_i}_{3,4}$ trained on $\downarrow\uparrow$ NTP packet/byte counts.

Note that some of these models may not be applicable to some IoT device classes. For example, Belkin devices, the LiFX lightbulb, the Nest smoke sensor, the Netatmo weather station, and the TP-Link camera do not use HTTPS traffic (based on the dataset we analyzed for this paper). Therefore, no HTTPS model is trained for those classes.

Employing models with different granularity not only helps with managing the computing resources (when used progressively) but also makes the monitoring phase explainable by flagging anomalies or deviations pertinent to a subset of traffic channels (*e.g.,* ICMP, TCP, UDP, HTTP, NTP, DNS) communicated by the device. This would help network administrators narrow their investigations in verifying and/or determining the exact type of attack. For example, if our TCP model detects an anomaly, the investigation can be focused on TCP-based attacks such as SYN flood and SYN reflection. In another example, if the Ethernet model flags an anomaly while TCP/UDP models produce no alert, the attack could be attributed to protocols like ICMP (*e.g.,* Ping-of-Death). It is important to note that every anomaly does not necessarily indicate malicious activities. The output of the monitoring phase can be consumed by a subsequent inference stage (beyond the



(a) Default convex hulls.

(b) Our refined convex hulls.

Fig. 6. Convex hulls (solid lines) versus spherical boundaries (dashed lines) for training instances of Belkin switch: (a) default convex hulls contain all data instances of corresponding clusters, and (b) our refined version of convex hulls exclude noises from benign boundaries.

scope of this paper and left for future work) that determines specific attacks by checking measured anomalous patterns against known signatures.

One-class models are often trained on data purely from an objective class (*e.g.,* benign instances [25, 67]). However, some algorithms like Isolation Forest [3] require some levels of impurity in training data (*e.g.,* benign instances dominate, but some malicious instances are present). We argue that including malicious instances in the training dataset may limit the model's learning to only "known" malicious behaviors [73]. Hence, models cannot react reliably to growing attacks with unbounded behaviors (zero-day attacks). A better strategy, particularly for IoT devices with bounded normal behaviors, is to train the models on only benign data, so any instance outside the benign boundary would be considered abnormal.

Clustering algorithms have proven to be effective in modeling expected patterns. However, some like K-Means inherently lack the notion of boundary. In other words, a data instance, no matter where it appears in the space and how far it is from expected instances, will be assigned (with a probability) to its nearest cluster. Therefore, to employ these models for anomaly detection applications, we need to augment them with custom boundaries. Work in [67] used spherical boundaries around the centroid of each cluster by covering 97.5% of the closest points to the centroid. Spherical boundaries are easy to form. However, there are many cases where the area they cover is unnecessarily bigger than the actual cluster, which makes the boundaries relatively loose. We will show in §5 that loose boundaries can increase the chance of false negatives and reduce anomaly detection rates.

In this paper, we use the concept of convex hull [60] for developing custom-shaped clusters. Given a set of points P, the convex hull is the smallest boundary that contains all points in P. Note that the complexity of checking whether an instance falls inside convex hull boundaries is higher than spherical boundaries – the time complexity is $O(P)$ versus $O(1)$. There are existing tools like SciPy library [64] that generate convex hull for a given set of points. In this paper, we employ PCA (Principle Component Analysis) to reduce the dimensions of our feature space into a 2D space. This dimension reduction helps with the training/testing time and also enables us to visualize the results. Note that one can also generate convex hulls on a multi-dimensional feature space with less than nine dimensions [64]. With convex hulls, a practical challenge is that training instances may still contain unintended noises that can lead to undesirable boundaries. Collecting purely benign traffic traces of networked assets is almost infeasible as incidents such as device

misconfiguration, network disruption, or even uncaught attacks can poison the benign dataset. Therefore, we need a data purification (refinement) method to avoid including noise instances in our benign boundaries. In this paper, we first use K-Means on raw benign data (including noises) to get the initial set of clusters. Next, we apply two filters (explained below) to remove some odd clusters (with characteristics too dissimilar to their cohort) and then prune the remaining clusters by tightening their boundary.

Fig. 6 illustrates the clusters (formed by K-Means) and their boundaries. In Fig. 6(a), the boundaries computed by the default convex hull are compared with the spherical boundaries. The convex hull algorithm requires at least three data points to form a boundary. Cluster 4, therefore, is excluded by default due to having only two data instances. It can be seen in Fig. 6(a) that default convex hulls include all instances which can skew the benign boundaries (*e.g.,* cluster 3 and, to some extent, cluster 2) toward seemingly noisy instances. Such performance is enhanced in Fig. 6(b), highlighting the refined version of the convex hull we developed for this paper.

4.1 Refined Convex Hull

Noises are inevitable in our training dataset which is intended to be purely benign. Traffic noises are often a few short-lived instances that are caused due to intermittent outages or variations in the network services. Therefore, the common observation is that noisy instances are usually scattered and appear far from most benign instances. Noise can lead to two detrimental impacts on clusters:

(1) Some clusters emerge that do not share the characteristics of other benign clusters. These clusters are often *sparse*, as opposed to most benign clusters that are formed with a significantly larger number of instances (*dense*). An indicator for distinguishing sparse from dense clusters is the mutual distance of the instances. In sparse clusters, instances are far from each other; hence, we expect to see a more prominent measure of mutual distance than the dense clusters.

(2) Although some clusters are dense, they may include noisy instances that can skew/loosen their benign boundary. We refer to these noise instances as *outliers*, which are often far from the centroid of their corresponding cluster compared to other instances in the same cluster.

In what follows, we explain the methods we develop for detecting sparse clusters and outlier instances in dense clusters:

Detecting Sparse Clusters: For this, we use the distribution of mutual distance (denoted as MD) across all clusters. Having a total of C clusters, for a given cluster $c \in [1, C]$ with P instances in it, we compute a mutual distance matrix $\left[D_{a,b}\right]_{P\times P}$ in which $D_{a,b}$ refers to the distance of instances a and b. From the matrix D, we measure MD_c (*i.e.,* the average mutual distance across P instances in the cluster c). Computing MD_c for all C clusters gives a distribution from which we compute an average (μ_{MD}) and standard deviation (σ_{MD}) across all C clusters.

A cluster c is sparse if $MD_c > \mu_{MD} + \alpha.\sigma_{MD}$, where α is a configurable parameter to specify how much deviation from the average is acceptable. Increasing α will allow for the inclusion of sparse clusters, and decreasing it will aim for highly-dense clusters.

Detecting Outliers: We focus on distances of points from the cluster centroid to detect outliers inside a relatively dense cluster. Let CD_a denote the distance of instance a (for all $a \in [1, P]$) from the centroid. The distribution of CD_a is represented by μ_{CD} and σ_{CD}, referring to the average and standard deviation of distances, respectively.

Now, for a given cluster, instance a becomes an outlier if $CD_a > \mu_{CD} + \beta.\sigma_{CD}$, where β is a configurable parameter like α explained above.

We also publicly release our Python source code for generating the convex hull of one-class models [56]. In §5, we will evaluate the refined convex hull model over a range of α and β values and demonstrate how they can affect the performance of anomaly detection.

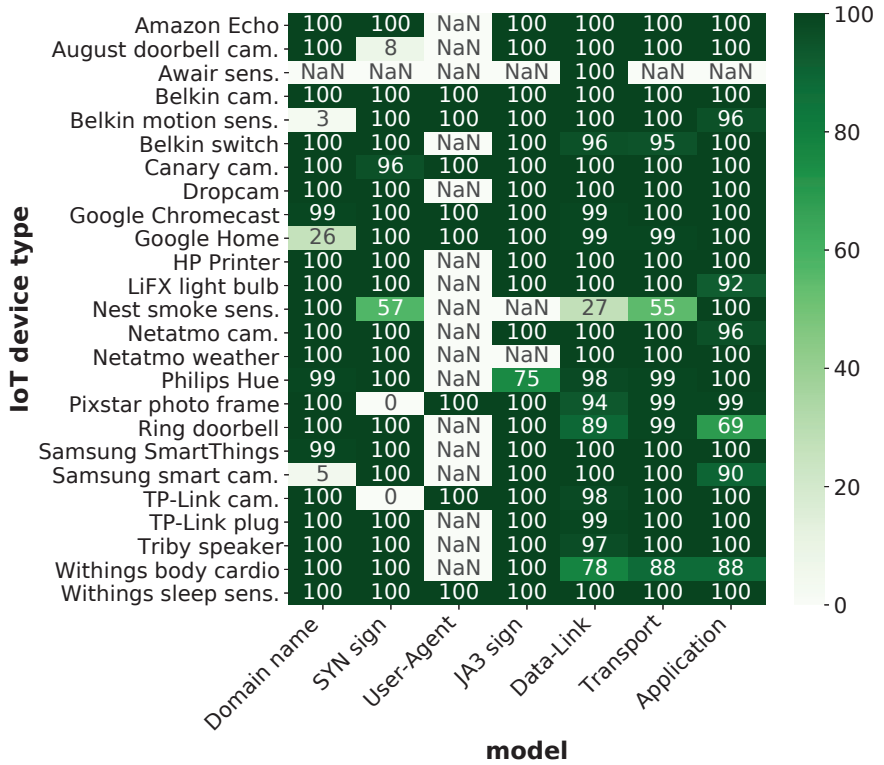


Fig. 7. Testing accuracy of each specialized model for IoT device types with **TEST1** dataset.

5 EVALUATION OF EFFICIENT IOT TRAFFIC INFERENCE

This section starts by evaluating our multi-view classification method on traffic traces we collected from our lab using three resolving techniques discussed in §3.4. We next evaluate our progressive one-class models using a public dataset. Note that the primary focus of this is to highlight the need for, as well as the efficacy of, modular (multi-view and progressive) inference for analyzing network traffic of IoT devices at scale. Hence, extensively evaluating the proposed methods on some of the public datasets that contain full IoT traffic traces [4, 20, 69] is beyond the scope of this paper and is left for future work.

5.1 Classification Phase Results

Before evaluating our multi-view approach, we need to solve the optimization problem formulated in §3.3 to select our optimal set of specialized models. We implement the optimization problem in two different Python libraries, namely Pyomo [10] and CVXPY [14]. CVXPY assures that our optimization problem is convex; otherwise, it would not be able to solve it. The feasibility matrix F of our optimization is initialized according to Eq. 2 using the threshold parameter A and the accuracy values obtained from the **TEST1** dataset (aggregate results were reported in Table 1). We choose the accuracy threshold parameter equal to a fairly high value 99%. The optimal set of models returned by both Pyomo and CVXPY consists of the domain name and datalink models, as they meet the accuracy threshold of 99% for each IoT class in our **TEST1** dataset – such a performance was expected from our optimization problem. We note that the actual accuracy can vary for a new

Table 2. Performance of multi-view classification at different resolution periods: all models versus the optimal set of models.

Resolver technique	All models			Optimal models		
	1h	6h	24h	1h	6h	24h
MV	99%	99%	98%	94%	93%	89%
WMV	100%	100%	100%	99%	99%	98%
AWMV	95%	91%	85%	93%	89%	83%

unseen dataset. We will shortly perform experiments with our optimization problem varying the thresholds and specialized models available to us.

Let us better understand how the optimal set of specialized models is computed. Fig. 7 shows the performance (prediction accuracy) of the seven models we discussed in §3.1 across 25 IoT device classes in the `TEST1` dataset. Cells with a "`NaN`" value highlight device classes d that miss traffic features necessary for model m, in the `TRAIN` dataset – hence, m is unable detect the class d. That said, Amazon Echo is an exception with the User-Agent model. In fact, Amazon Echo has a unique User-Agent header in our `TRAIN` dataset, but it does not exchange any User-Agent header in the `TEST1` dataset. The SYN signature model mispredicts all testing instances of Pixstar photo frame as Google Home and those of TP-Link camera as Belkin switch. This is because of their shared SYN signatures. For the Awair sensor, all datasets (*i.e.,* `TRAIN`, `TEST1`, and `TEST2`) only include EAPol (Extensible Authentication Protocol over LAN) traffic which does not use TCP/UDP protocols. No traffic except Ethernet is seen for this sensor. Therefore, only the datalink model provides predictions for this IoT class.

According to Fig. 7, the only model that is able to provide inference for all IoT classes (device types) is the datalink model. In other words, the datalink model is the only one that offers an availability metric of 100% for our multi-view classification. At the same time, the datalink model falls short of expectations in accurately predicting traffic for several classes, including the Belkin switch, the Nest smoke sensor, the Philips Hue, the Pixstar photo frame, the Ring doorbell, the TP-Link camera, the Triby speaker, and the Withings body cardio. The prediction accuracy for those individual classes is less than our desirable threshold of 99%. Considering alternatives for these classes, we find that the domain name model gives an accuracy greater than 99% for all of them. Hence, having at least two models, namely the domain name model and the datalink model, would satisfy our optimization constraints.

Experimenting with Optimization: We saw in Fig. 7, the seven specialized models we consider in this paper perform fairly well on the majority of IoT classes. Such high performance from individual models may not highlight optimization's benefits to our multi-view inference. Note that our multi-view architecture and optimization readily allow for various specialized models (an extended version or a subset of our list of models) and IoT classes. In order to showcase the dynamics of our optimal model selection, let us experiment with a list of models when the domain name and SYN signature models are assumed to be excluded. We solve our optimization problem with the remaining five specialized models. This time the optimal set consists of JA3, datalink, and application models. Again, this selection is not overly unexpected. Similar to what we saw earlier, the datalink model must be selected as it is the only model that can predict traffic for the Awair sensor. JA3 and application models are needed to be in the mix for the Withings body cardio and the Nest smoke sensor, respectively. Note that we expect an accuracy of at least 99% per class.

Our optimization problem, with no exception, may become infeasible when it is applied to another dataset, set of models, or IoT classes. In that case, relaxing the accuracy threshold (less than 99%) can help make the problem feasible. To showcase an infeasible scenario, let us limit our

Table 3. Cost of multi-view classification: all models versus the optimal set of models.

Cost	All models	Optimal models
CPU time (ns)	866	127
TCAM size (bytes)	94	12

superset of models to User-Agent, datalink, transport, and application with our original accuracy threshold of 99%. In this case, no model can meet the threshold for two classes, namely the Ring doorbell and the Withings body cardio – the problem becomes infeasible to solve. However, if we relax the accuracy threshold to a value like 85%, the datalink and application models satisfy the accuracy constraints for all devices and become our optimal set.

Performance of Optimal Multi-View Classification: We now evaluate the efficacy of our optimal multi-view classification by comparing its performance and cost against those of a classification scenario where all specialized models are employed. For this purpose, we use `TEST2` dataset. Table 2 summarizes the prediction performance (detection rate) of all models versus optimal models selected at three resolution periods: short (one-hour), medium (six-hour), and long (24-hour), with three resolver techniques discussed in §3.4. Given five days' worth of data available in `TEST2`, we can evaluate the performance across 120 short, 20 medium, and 5 long resolution periods. It is important to note that the size of the resolution periods is independent of the classification epoch duration, which is set to a minute throughout this paper. Therefore, longer resolution periods cannot help with the missing values, which is the challenge of the single-combined-view model (§3.2). For the evaluation metric, we measure the detection rate, which is the fraction of total active IoT devices that are correctly classified for a given resolution period. Table 2 shows the average detection rate computed across all resolution periods. Note that the detection rate metric slightly differs from the accuracy metric (per one-minute epoch) we initially used to evaluate the prediction of specialized models in §3. This new metric is computed at the end of each resolution period, where a device is mapped to a class after resolving several predictions.

The first observation from Table 2 is that making classification inferences using all models is slightly more accurate than the optimal models. The lowest gap is 1% when the WMV resolver technique is used for the short and medium resolution periods. It can be as high as 9% for MV over the long resolution period. Another interesting observation is that the inference (using all models and optimal models) performs better when the resolution period is shorter, indicating that mispredictions accumulated over time can be detrimental to the ultimate inference. The impact of such accumulation is more pronounced in MV and AWMV techniques, whereas WMV yields a fairly robust detection rate across the three resolution periods. The advantages of the optimally-selected models become more apparent when computing and processing costs are considered. Table 3 shows the CPU time and TCAM size required for all models versus optimal models. It can be seen that the cost of optimal models is significantly lower than all models by a factor of 6 (by the measure of CPU) and 7 (by the measure of TCAM size).

As discussed throughout this paper, classification is the foundational step for the monitoring phase. Therefore, it is essential to correctly determine the class of all devices before proceeding to the monitoring phase. As we saw in Table 2, this assurance is given by using all models with the WMV technique; however, the optimal set may miss some devices in a few resolution periods. For example, out of the total of 120 short (1hr) resolution periods, the optimal scheme with WMV technique mispredicts the Google Home as the Google Chromecast in 9 periods, the Pixstar photo frame as the Belkin switch in 7 periods, and the Nest smoke sensor as the Google Home in 2 resolution periods. Although the chance of incorrect inference (after resolution) is relatively minimal, it can be avoided by adding another layer of consolidation and/or resolution on top of our current inference. For

example, one can keep obtaining the classification outputs for several resolution periods (*e.g.,* six periods of each one hour) and then resolve conflicts (*i.e.,* different claimed for given MAC address) using similar techniques explained in §3.4. Another technique that could be effective in resolving conflicts is developing a consistency score [67] that helps select the class that is consistently given to a device for a number of consecutive resolution periods. Such an additional resolution is beyond the scope of this paper. In the context of obtaining a perfect classification for all devices, we can compare the performance of all models versus optimal models in terms of response time. It is seen in Table 2 that all models can achieve 100% detection rate in a single 1-hour resolution period; however, the optimal models need extra 1-hour periods for obtaining the same result. Even with some delays, the optimal approach is still attractive in terms of computing costs – we often prefer to spend a few more hours to make a reliable (accurate and confident) inference but keep the computing cost to a minimum.

5.2 Monitoring Phase Results

In order to evaluate the performance of the monitoring phase, we first train seven one-class models per IoT class using their benign traffic data and then quantify their false positive rates (incorrectly predicting benign instances as anomalies) when applied to unseen benign data and their true positive rates (correctly detecting anomalies) when applied to attack (malicious) traffic data. For this purpose, we use public traffic traces [1] that were collected from our own testbed in 2018 and include benign and attack traffic traces of the same set of IoT classes (except for the Withings sleep sensor) we had in classification phase of this paper. We use benign traffic traces collected in May, June, and October 2018 to train the one-class models. We found some devices were inactive during the capture period for several days. Hence, the benign dataset could not be split chronologically into training and testing portions, as it would have resulted in a heavily imbalanced training or testing dataset. To overcome this challenge, we shuffle the benign dataset to distribute data instances across all device classes evenly. Next, we take 70% of the benign dataset and create `BENIGN1` instances used for training the one-class models, and the remaining 30% of the data will create `BENIGN2` instances used for measuring false positive ratio. Note that portions of the public traces [1] we obtain are a mixture of benign and attack traffic. The annotation files released by the same source [1] helped us extract only attack traffic, constructing our `ATTACK` dataset, which we use for measuring the true positives. The CSV files of `BENIGN1`, `BENIGN2`, and `ATTACK` instances are publicly released [56]

The `ATTACK` dataset includes instances of four types of volumetric attacks, including Ping-of-Death, SYN attack, SYN reflection attack, and UDP DoS attack on IoT devices [1]. Each attack was launched at three data rates: 1, 10, and 100 attack-packets-per-second. Attacks were launched using a Python script using the Scapy library. For direct attacks like Ping-of-Death, SYN attack, and UDP DoS, two configurations: (i) local attacker and (ii) remote attacker, were employed to overwhelm target IoT devices connected to the network. For SYN reflection, attacks were launched: (i) from a local attacker machine to the target IoT devices which reflected the incoming traffic to another local machine, and (ii) from a remote attacker machine to the target IoT devices, which reflected the incoming traffic to another remote machine.

For the monitoring phase, we experimented with one-minute and five-minute epochs. We found that an epoch duration of five minutes gives better performance in terms of false/true positives. Hence, we use five-minute epochs for the rest of this subsection. As discussed in §4, some of the intended models cannot be trained for certain IoT device classes due to the unavailability of the features required. We also noticed that some device types generate certain features infrequently. For instance, the Ring doorbell only generates NTP traffic in 20 epochs, in contrast to 823 epochs with DNS traffic. Models trained with insufficient epochs yield poor performance. Therefore, we

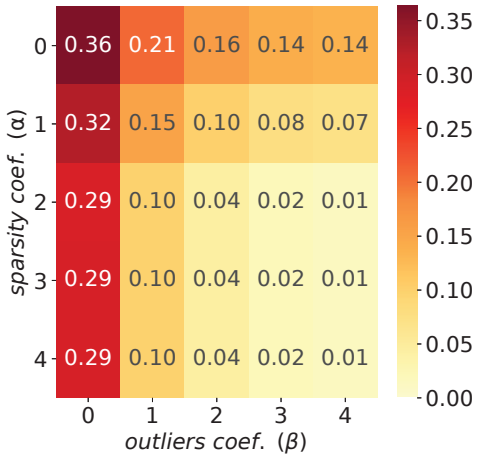


Fig. 8. Average false positive (per IoT class) of the coarse-grained models applied to the benign testing dataset with varying combinations of coefficients α and β.

empirically found that at least 100 epochs (about 8 hours' worth of traffic) are needed for a reliable one-class model.

Let us begin by evaluating our one-class models with varying coefficients α (sparsity) and β (outliers) to analyze their impact on the performance of models. For this evaluation, we only focus on the coarse-grained model (introduced in §4) as Ethernet traffic features are available for all device classes. That said, one can perform the same evaluation on other one-class models and fine-tune parameters α and β for them separately. We use the best combination of α and β obtained for the Ethernet one-class models as default settings for other models (medium-grained and fine-grained).

Fig. 8 illustrates a heat map indicating the average false positive ratio of one-class models (of all IoT classes) when applied to the benign testing dataset for 25 combinations of values for coefficients α and β. Recall smaller α values result in more clusters being considered sparse (noisy) and removed, meaning a reduced number of benign clusters (conservative approach). The coefficient β has a similar effect in restricting benign boundaries (of those clusters considered as dense). Smaller β results in clusters with tighter boundaries, increasing the chance of some truly benign instances falling outside the benign boundaries and contributing to false positives.

Although larger α and β values can improve false positives with the benign dataset, they may loosen boundaries and cause false negatives with the attack dataset. Therefore, balancing these two cases, we choose $\alpha = 2$ and $\beta = 2$ (the middle cell in Fig. 8), which gives a relatively low average false positive ratio equal to 0.04, and an average true negative of 0.96.

With parameters α and β fine-tuned empirically, let us quantify the performance of individual models (specialized and combined) per each IoT class. Fig. 9 shows the false positive ratio of each model across IoT classes on the benign testing dataset. To compare with the performance of our specialized models, we train a combined model similar to what we saw earlier in the classification phase (§3.2). Cells denoted by "NaN" and "Low", respectively, indicate no training instances and a low number (< 100) of training instances. On average, each model offers a reasonably acceptable false positive ratio of less than 0.05 across all device classes. Among eight models, the DNS model performs the worst for packets of the Samsung smart camera with about 0.24 false positive ratio – this could be due to the marginally small number (108) of DNS training instances for this class which is just slightly higher than the minimum requirement of 100 instances discussed above. Two IoT classes, namely the Nest smoke sensor and the Withings body cardio, were only active over 22

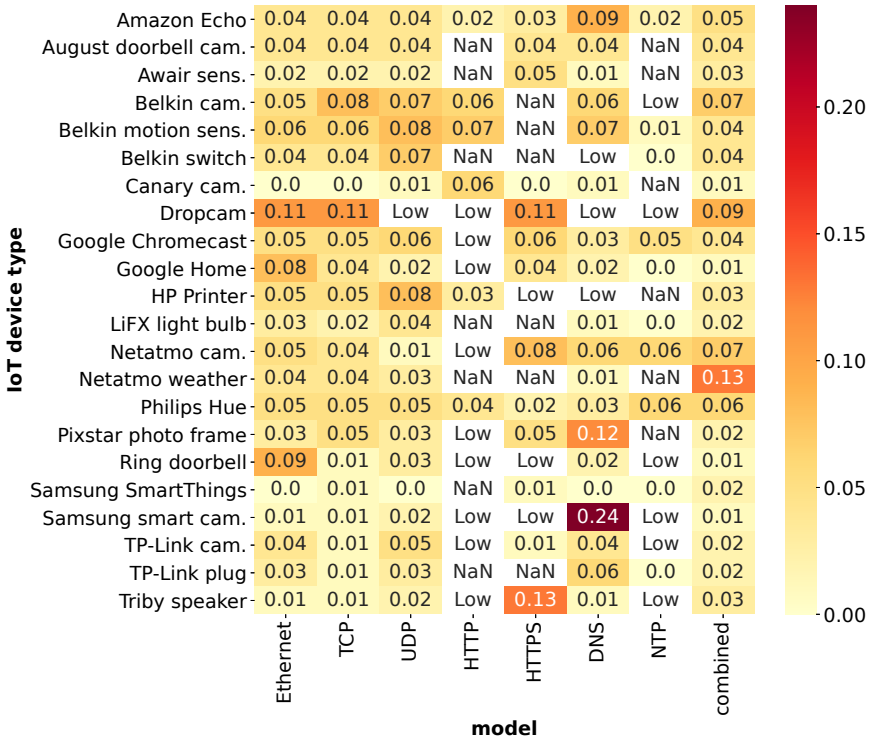


Fig. 9. False positives of individual models (specialized and combined) per IoT class when applied to the benign testing dataset.

and 15 epochs, respectively. Therefore, they are excluded from the evaluation due to insufficient training instances. Looking at the performance of the combined model, it gives an average false positive of 0.03, which is almost the same or slightly better (by a low margin of less than 1%) than that of specialized models. Regarding cost considerations, our computation of flow-based models in §3.2 reveals the following: the cost of the Ethernet model is 12 bytes; for both TCP and UDP models, the cost is 14 bytes each; and, the cost of HTTP, HTTPS, DNS, and NTP models amounts to 18 bytes each. Therefore, the combined model's cost would total 112 bytes, which surpasses each specialized model by a minimum factor of 6. Again, similar to what we found for the single-view-combined model at the classification phase (§3.2), the performance gain of the combined monitoring model is much less than the overall cost of the model.

We now evaluate the true positive rate of our models by applying them to the **ATTACK** dataset. Note that the public traffic traces [1] we used to create our **ATTACK** dataset do not include any attacks via HTTP, HTTPS, DNS, or NTP. However, there are a few instances of attacks via SSDP and SNMP, but we did not train one-class models associated with those specific application-layer protocols. Still, Ethernet and/or UDP models can flag behavioral deviations due to SSDP/SNMP-based attacks. That said, one may train SSDP and SNMP one-class models and utilize our progressive monitoring architecture to detect such attacks.

Table 4 shows the true positives for each attack instance on their respective IoT device type. Some attacks in our dataset were launched on specific device types, leading to blank cells in the table. Each attack was launched at three rates of 1, 10, and 100 packets per second (pps). For each attack vector, we report the true positives of those models whose features could be affected by the

Table 4. True positive rate of the models when applied to attack traces. "E", "T", "U", and "C" denote Ethernet, TCP, UDP, and combined models, respectively.

	Ping-of-Death			**SYN attack**			**SYN reflection**			**UDP DoS**		
	1pps	10pps	100ps	1pps	10pps	100ps	1pps	10pps	100ps	1pps	10pps	100ps
Belkin switch	[E]:1.00 [C]:0.50	[E]:1.00 [C]:0.50	[E]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:0.50	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	–	–	–
Belkin motion sens.	[E]:0.50 [C]:0.00	[E]:1.00 [C]:1.00	[E]:1.00 [C]:1.00	[E]:0.00 [T]:0.17 [C]:0.00	[E]:1.00 [T]:1.00 [C]:0.84	[E]:1.00 [T]:1.00 [C]:1.00	[E]:0.67 [T]:0.50 [C]:0.50	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	[E]:0.50 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00
Samsung smart cam.	[E]:1.00 [C]:0.50	[E]:1.00 [C]:1.00	[E]:1.00 [C]:0.50	[E]:1.00 [T]:1.00 [C]:0.84	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [T]:1.00 [C]:0.84	[E]:1.00 [T]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00
Netatmo cam.	–	–	–	[E]:0.50 [T]:1.00 [C]:0.50	[E]:1.00 [T]:1.00 [C]:0.50	[E]:1.00 [T]:1.00 [C]:1.00	[E]:0.34 [T]:1.00 [C]:0.34	[E]:1.00 [T]:0.84 [C]:0.38	[E]:1.00 [T]:1.00 [C]:0.88	–	–	–
Google Chromecast	–	–	–	[E]:0.25 [T]:1.00 [C]:0.75	[E]:0.25 [T]:1.00 [C]:1.00	[E]:0.75 [T]:1.00 [C]:1.00	[E]:0.00 [T]:1.00 [C]:0.75	[E]:0.25 [T]:1.00 [C]:1.00	[E]:0.50 [T]:1.00 [C]:1.00	–	–	–
LiFX lightbulb	[E]:1.00 [C]:0.00	[E]:0.50 [C]:1.00	[E]:1.00 [C]:1.00	–	–	–	–	–	–	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00
Amazon Echo	–	–	–	–	–	–	–	–	–	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00	[E]:1.00 [U]:1.00 [C]:1.00

attack. For example, the Ping-of-Death attack uses ICMP to send ping requests. It only appears in Ethernet traffic without affecting features of TCP/UDP and above. In most cases, the Ethernet model yields the same true positive rate as TCP/UDP models. For 10 attacks, TCP or UDP models show a higher true positive rate than the Ethernet model. For example, the TCP model can detect all SYN and SYN reflection attacks (the two middle columns in Table 4) on the Google Chromecast, while the Ethernet model can, at best, detect three-quarters of high rates attacks (100pps).

Focusing on the combined model, we see that its true positive rate is lower than or equal to that of the specialized models for most of the attacks. However, there are some exceptions, like the UDP DoS attacks on the Belkin motion sensor, SYN and SYN reflection attacks on the Google Chromecast, and the Ping-of-Death attacks on the LiFX lightbulb in which the combined model outperformed the Ethernet model. For all instances of SYN, SYN reflection, and UDP DoS attacks, the true positive of the TCP and UDP model are higher than the combined model. Overall, the average true positive rate of the Ethernet, TCP, UDP, and combined models (across all device types and attacks available in our dataset) is 0.85, 0.96, 1.00, and 0.83, respectively. Also, the average false negative of Ethernet, TCP, UDP, and combined models is 0.15, 0.04, 0.00, and 0.17, respectively. This highlights that finer-grained models (which are more expensive computationally) can better detect anomalies. Also, in the monitoring phase, similar to what is realized in the classification phase, specialized models are relatively more beneficial than a combined model.

To compare the impact of convex hull boundaries versus spherical boundaries on the performance of models in detecting attacks, we generated benign clusters with spherical boundaries that contain the 97.5th percentile of benign instances. We found that spherical boundaries cause the models to fall short in detecting some of the network attack instances. For example, the TCP model with spherical boundaries gives a true positive of 0.71 for SYN and SYN reflection attacks on the Google Chromecast, while this measure is 1.00 for the TCP model with convex hull boundaries. In fact, the spherical boundaries lead to lower true positives for all four types of attacks, with the Ethernet, TCP, and UDP models having average true positives of 0.73, 0.80, and 0.96.

6 CONCLUSION

In this paper, we developed a generic traffic inference architecture consisting of a multi-view classification stage followed by progressive monitoring phases. We developed a systematic approach for the multi-view classification to quantify the prediction power versus computing costs of the specialized-view models, each trained on specific features (packet or flow). We demonstrated that specialized models could be more accurate and cost-effective than a single combined-view model. We also formulated an optimization problem to select the best views for the multi-view classification that minimizes the computing cost subject to prediction accuracy requirements. For progressive monitoring, we employed one-class models and developed configurable convex hulls for them, tightening benign behavioral boundaries. We evaluated the efficacy of our models (classification and monitoring) by applying them to real traffic traces collected from our IoT testbed. We found that our optimal set of specialized classifier models can reduce the processing cost by a factor of six with insignificant impacts on the prediction accuracy. Also, our monitoring models yielded an average true positive rate of 94% and a false positive rate of 5%. Finally, we publicly released our data (training and testing instances of classification and monitoring tasks) and enhancement code for the convex hull algorithm to the research community.

REFERENCES

[1] A. Hamza. 2019. IoT Benign and Attack Traces. https://iotanalytics.unsw.edu.au/attack-data.html
[2] Yaser Abu-Mostafa et al. 2012. *Learning From Data.* AMLBook.
[3] J. Ahmed et al. 2020. Monitoring Enterprise DNS Queries for Detecting Data Exfiltration From Internal Hosts. *IEEE Transactions on Network and Service Management* 17, 1 (2020), 265–279.
[4] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In *Proc. IEEE Symposium on Security and Privacy (S&P)*. San Francisco, CA, USA.
[5] J. Anand, A. Sivanathan, A. Hamza, and H. Habibi Gharakheili. 2021. PARVP: Passively Assessing Risk of Vulnerable Passwords for HTTP Authentication in Networked Cameras. In *Proc. ACM Workshop on DAI-SNAC*. Virtual Event, Germany, 10–16.
[6] B. Bezawada et al. 2018. Behavioral Fingerprinting of IoT Devices. In *Proc. ACM ASHES*. Toronto, Canada.
[7] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. 2018. Behavioral Fingerprinting of IoT Devices. In *Proc. of the ASHES*. Toronto, Canada.
[8] Bitdefender. 2017. Infected vending machines, lamps, other IoT devices shut down university network. https://bit.ly/3NE6dPu
[9] A. Bremler-Barr et al. 2020. IoT or NoT: Identifying IoT Devices in a Short Time Scale. In *Proc. IEEE/IFIP NOMS*. Budapest, Hungary.
[10] M. Bynum et al. 2021. *Pyomo–Optimization Modeling in Python* (third ed.). Vol. 67. Springer Science & Business Media.
[11] Silvia Cateni et al. 2014. A Method for Resampling Imbalanced Datasets in Binary Classification Tasks for Real-world Problems. *Neurocomputing* 135 (2014), 32–41. https://doi.org/10.1016/j.neucom.2013.05.059
[12] Cisco. 2012. Introduction to Cisco IOS NetFlow - A Technical Overview. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html
[13] Cyber Edge. 2020. Cyberthreat Defense Report. https://cyber-edge.com/wp-content/uploads/2020/03/CyberEdge-2020-CDR-Report-v1.0.pdf
[14] S. Diamond et al. 2016. CVXPY: A Python-embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research* 17, 83 (2016), 1–5.
[15] R. Doshi et al. 2018. Machine Learning DDoS Detection for Consumer Internet of Things Devices. In *Proc. IEEE S&P Workshops*. San Francisco, USA.
[16] Tlamelo E. et al. 2021. A Survey on Missing Data in Machine Learning. *Journal of Big Data* 8 (2021), 1–37.
[17] Wesley Eddy. 2022. Transmission Control Protocol (TCP). RFC 9293. https://doi.org/10.17487/RFC9293
[18] X. Feng, Q. Li, H. Wang, and L. Sun. 2018. Acquisitional Rule-based Engine for Discovering Internet-of-Things Devices. In *USENIX Security*. Baltimore, USA.
[19] Forescout. 2016. Network visibility survey. http://bit.ly/30LBGaf.
[20] S. Garcia, A. Parmisano, and M. J. Erquiaga. 2023. oT-23: A Labeled Dataset with Malicious and Benign IoT Network Traffic. http://doi.org/10.5281/zenodo.4743746
[21] H. Guo et al. 2018. IP-Based IoT Device Detection. In *Proc. of ACM IoT S&P*. Budapest, Hungary.

[22] Hang Guo et al. 2020. *IoTSTEED: Bot-side Defense to IoT-based DDoS Attacks (Extended).* Technical Report ISI-TR-738. USC/Information Sciences Institute. https://bit.ly/3ec9eGS
[23] H. Guo and J. Heidemann. 2020. *IoTSTEED: Bot-side Defense to IoT-based DDoS Attacks (Extended).* Technical Report ISI-TR-738. USC/Information Sciences Institute. https://www.isi.edu/%7ejohnh/PAPERS/Guo20b.html
[24] H. Habibi Gharakheili, M. Lyu, Y. Wang, H. Kumar, and V. Sivaraman. 2019. iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 1071–1085. https://doi.org/10.1109/TNSM.2019.2929511
[25] A. Hamza et al. 2019. Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity. In *Proc. ACM SOSR*. New York, USA.
[26] A. Hamza et al. 2020. Verifying and Monitoring IoTs Network Behavior using MUD Profiles. *IEEE Transactions on Dependable and Secure Computing* 19, 1 (May 2020), 1–18.
[27] A. Hamza et al. 2022. Verifying and Monitoring IoTs Network Behavior using MUD Profiles. *IEEE TDSC* 19, 1 (2022), 1–18.
[28] Ayyoob Hamza, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2018. Combining MUD Policies with SDN for IoT Intrusion Detection. In *Proc. ACM IoT S&P*. Budapest, Hungary.
[29] M. Hasan et al. 2019. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things* 7 (2019), 1–14.
[30] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. M. A. Hashem. 2019. Attack and Anomaly Detection in IoT Sensors in IoT Sites Using Machine Learning Approaches. *Internet of Things* 7 (2019), 100059.
[31] Md Kamrul Hasan et al. 2021. Missing Value Imputation Affects the Performance of Machine Learning: A Review and Analysis of the Literature (2010–2021). *Informatics in Medicine Unlocked* 27 (2021), 100799.
[32] J. Holland, R. Teixeira, P. Schmitt, K. Borgolte, J. Rexford, N. Feamster, and J. Mayer. 2020. Classifying Network Vendors at Internet Scale. https://doi.org/10.48550/ARXIV.2006.13086
[33] D. Yuxing Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster. 2020. IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale. *ACM IMWUT* 4, 2 (2020).
[34] IETF. 2013. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. https://tools.ietf.org/html/rfc7011
[35] IETF. 2019. Manufacturer Usage Description Specification. https://tools.ietf.org/html/rfc8520
[36] Houda Jmila, Gregory Blanc, Mustafizur R. Shahid, and Marwan Lazrag. 2022. A Survey of Smart Home IoT Device Classification Using Machine Learning-Based Network Traffic Analysis. *IEEE Access* 10 (2022), 97117–97141. https://doi.org/10.1109/ACCESS.2022.3205023
[37] D. Kumar et al. 2019. All Things Considered: An Analysis of IoT Devices on Home Networks. In *Proc. USENIX Security*. Santa Clara, USA.
[38] F. Loi et al. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. In *Proc. ACM Workshop on IoT S&P*. Dallas, Texas, USA, 1–6.
[39] G. Lyon. 1997. Nmap. https://nmap.org/
[40] Mi. Lyu et al. 2017. Quantifying the Reflective DDoS Attack Capability of Household IoT Devices. In *Proc. ACM WiSec*. Boston, Massachusetts, USA, 46–51.
[41] M. Lyu, D. Sherratt, A. Sivanathan, H. Habibi Gharakheili, A. Radford, and V. Sivaraman. 2017. Quantifying the Reflective DDoS Attack Capability of Household IoT Devices. In *Proc. ACM WiSec*. Boston, USA.
[42] S. Marchal et al. 2019. AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication. *IEEE JSAC* 37, 6 (June 2019), 1402–1412.
[43] M. Mazhar et al. 2020. Characterizing Smart Home IoT Traffic in the Wild. In *Proc. IEEE/ACM IoTDI*. Los Alamitos, USA.
[44] Y. Meidan et al. 2017. ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis. In *Proc. SAC*. Marrakesh, Morocco.
[45] Y. Meidan et al. 2018. N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Computing* 17, 3 (2018), 12–22.
[46] Y. Meidan et al. 2020. A Novel Approach For Detecting Vulnerable IoT Devices Connected Behind a Home NAT. *Computers & Security* 97 (Oct. 2020), 1–23.
[47] M. Miettinen et al. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *Proc. of IEEE ICDCS*. Atlanta, USA.
[48] D. Mills. 1992. Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305. https://doi.org/10.17487/RFC1305
[49] MITRE. 2020. Common Vulnerabilities and Exposures. https://cve.mitre.org/
[50] N. Msadek et al. 2019. IoT Device Fingerprinting: Machine Learning based Encrypted Traffic Analysis. In *Proc. IEEE WCNC*. Marrakesh, Morocco.

[51] T. D. Nguyen et al. 2019. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *IEEE ICDCS*. Dallas, USA.
[52] T. D. Nguyen et al. 2019. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *Proc IEEE ICDCS*. Dallas, USA.
[53] Paloato. 2020. Unit 42 IoT Threat Report. https://start.paloaltonetworks.com/unit-42-iot-threat-report
[54] A. Pashamokhtari et al. 2020. Progressive Monitoring of IoT Networks Using SDN and Cost-Effective Traffic Signatures. In *Proc. of ETSecIoT*. Sydney, Australia.
[55] A. Pashamokhtari et al. 2021. Inferring Connected IoT Devices from IPFIX Records in Residential ISP Networks. In *Proc. IEEE LCN*. Virtual Event, Canada.
[56] A. Pashamokhtari et al. 2022. IoT Traffic Instances. https://iotanalytics.unsw.edu.au/smartinfer.html
[57] A. Pashamokhtari et al. 2022. PicP-MUD: Profiling Information Content of Payloads in MUD Flows for IoT Devices. In *Proc. IEEE WoWMoM*. Belfast, United Kingdom.
[58] Red-Button. 2016. Dyn (DynDNS) DDoS Attack. https://www.red-button.net/blog/dyn-dyndns-ddos-attack
[59] D. Reis et al. 2018. One-class Quantification. In *Proc. ECML PKDD*. Dublin, Ireland.
[60] R. T. Rockafellar. 1997. *Convex Analysis*. Princeton Mathematical Series.
[61] M. Safi et al. 2022. A Survey on IoT Profiling, Fingerprinting, and Identification. *ACM TIOT* 3, 4, Article 26 (sep 2022), 39 pages.
[62] S. J. Saidi et al. 2020. A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild. In *Proc. of ACM IMC*. New York, USA.
[63] Salesforce. 2019. TLS Fingerprinting with JA3 and JA3S. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967
[64] SciPy. 2021. SciPy Convex Hull. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html
[65] R. A. Sharma et al. 2022. Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment. In *Proc. USENIX Security*. Boston, USA.
[66] A. Sivanathan et al. 2018. Can We Classify an IoT Device using TCP Port Scan?. In *Proc. IEEE ICIAfS*. Colombo, Sri Lanka.
[67] A. Sivanathan et al. 2020. Detecting Behavioral Change of IoT Devices Using Clustering-Based Network Traffic Modeling. *IEEE Internet of Things Journal* 7, 8 (2020), 7295–7309.
[68] A. Sivanathan et al. 2020. Managing IoT Cyber-Security Using Programmable Telemetry and Machine Learning. *IEEE Transactions on Network and Service Management* 17, 1 (2020), 60–74.
[69] A. Sivanathan, H. Habibi Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. 2019. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2019), 1745–1759.
[70] A. Sivanathan, F. Loi, H. Habibi Gharakheili, and V. Sivaraman. 2017. Experimental evaluation of cybersecurity threats to the smart-home. In *Proc. IEEE ANTS*. Bhubaneswar, India, 1–6.
[71] V. Sivaraman, D. Chan, D. Earl, and R. Boreli. 2016. Smart-Phones Attacking Smart-Homes. In *Proc. ACM WiSec*. Darmstadt, Germany, 195–200.
[72] V. Sivaraman, H. Habibi Gharakheili, C. Fernandes, N. Clark, and T. Karliychuk. 2018. Smart IoT Devices in the Home: Security and Privacy Implications. *IEEE Technology and Society Magazine* 37, 2 (2018), 71–79.
[73] R. Sommer and V. Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proc IEEE S&P*. Oakland, CA, USA, 305–316.
[74] H. Sullivan, A. Sivanathan, A. Hamza, and H. Habibi Gharakheili. 2023. Programmable Active Scans Controlled by Passive Traffic Inference for IoT Asset Characterization. In *Proc. IEEE/IFIP NOMS Workshop on Manage-IoT*. Miami, FL, USA.
[75] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy. 2019. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet of Things Journal* 6, 1 (2019), 940–952. https://doi.org/10.1109/JIOT.2018.2865604
[76] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. 2019. PingPong: Packet-Level Signatures for Smart Home Device Events. In *Proc. of NDSS*. San Diego, California.
[77] Y. Wang et al. 2021. Analyzing the Impact of Missing Values and Selection Bias on Fairness. *International Journal of Data Science and Analytics* 12, 2 (2021), 101–119.
[78] K. Yang et al. 2019. Towards automatic fingerprinting of IoT devices in the cyberspace. *Computer Networks* 148 (2019), 318–327.
[79] K. Yang, Q. Li, and L. Sun. 2019. Towards automatic fingerprinting of IoT devices in the cyberspace. *Computer Networks* 148 (2019), 318–327.
[80] J. Zhao et al. 2017. Multi-view Learning Overview: Recent Progress and New Challenges. *Information Fusion* 38 (2017), 43–54.