Progressive Monitoring of IoT Networks Using SDN and Cost-Effective Traffic Signatures

Arman Pashamokhtari, Hassan Habibi Gharakheili and Vijay Sivaraman
School of Electrical Engineering and Telecommunications, UNSW Sydney, Australia
Emails: {armanpasha, h.habibi, vijay}@unsw.edu.au

***Abstract*—IoT networks continue to expand in various domains, from smart homes and campuses to smart cities and critical infrastructures. It has been shown that IoT devices typically lack appropriate security measures embedded, and hence are increasingly becoming the target of sophisticated cyber-attacks. Also, these devices are heterogeneous in their network communications that makes it difficult for operators of smart environments to manage them at scale. Existing monitoring solutions may perform well in certain environments, however, they do not scale cost-effectively and are inflexible to changes due to their static use of models. In this paper[1], we use SDN to dynamically monitor a selected portion of IoT packets or flows, and develop specialized models to learn corresponding traffic signatures. Our first contribution develops a progressive inference pipeline, comprising a number of machine-learning models each is specialized in certain features of IoT traffic. Our inference engine dynamically obtains selected telemetry, including a subset of traffic or flow counters, using SDN techniques. Our second contribution develops three supervised multi-class classifiers, two are protocol specialists trained by packet-based features and one is flow-based model trained by behavioral characteristics of ten unidirectional flows. Our third contribution evaluates the performance of our scheme by replaying real traffic traces of 26 IoT devices on to an SDN switching simulator in conjunction with three trained Random Forest models. Our system yields an overall accuracy of 99.4%. We also integrate our system with an off-the-shelf IDS (Zeek) to flag TCP flood and reflection attacks by inspecting only the suspicious device network traffic.**

I. Introduction

The number of connected IoT devices worldwide is projected [1] to increase to 43 billion in 2023, an almost threefold increase from 2018. Due to ease of use and deployment, about 25% of businesses and enterprises are using IoT technologies today [2]. With this growing demand for IoTs, manufacturers who rush to deliver innovative gadgets, attracting customers, inevitably use third-party software components (*e.g.,* communication libraries, encryption libraries, OS, open-source tools) within the device. The use of deprecated or insecure software components often allows the device to be compromised or used to attack others on the network. The risk of vulnerabilities in embedded components is not easy to mitigate, and thus IoT systems require extra caution when it comes to security. However, nearly 66% of organizations do not have adequate visibility into IoT devices connected to their network [3]. Unlike other computer systems, IoT networks contain thousands of IoT devices, which makes them an attractive target [4] of launchpad for large scale DDoS attacks, *e.g.,* Mirai [5], BASHLITE [6], and Remaiten [7].

Due to escalating security concerns associated with these vulnerable devices, network operators need to obtain real-time insights into the operation of their IoT systems to better maintain and support their assets. A range of classification methods have been developed [8]–[15] to fulfill this need by generating machine-learning models that infer the type of IoT devices from their traffic signatures. These models which are trained by benign activity profile of IoTs can be used to classify devices and possibly detect anomalies.

Most of the existing works [8]–[11], [13] need to inspect every packet going through the network, while some others [12], [14], [15] collect some flow-based statistical information which requires the insertion of rules into a programmable switch for every connected device. Also, existing works often use a single multi-class classifier, which requires a large amount of data and heavy machinery to train, run, and maintain it. Furthermore, we argue that the amount of traffic features required to distinguish an IoT varies across device types. Some devices have unique traffic features, while others have some features in common. Existing works do not consider this fact and treat all IoTs in the same way in terms of traffic features needed for their model.

In this paper, we aim to combat existing challenges by joint use of dynamic telemetry (enabled by SDN) and progressive inferencing strategy. Our contributions are threefold: (1) We develop a scalable and cost-effective architecture that employs multi classification models and dynamically selects a subset of traffic for each device based on the confidence output of the models. It is cost-effective because it judiciously obtains information from a small subset of network traffic for certain devices. Traffic selection is done by SDN, thus the cost of network telemetry can be controlled, (2) We train three models which two of them are packet-based (SYN and DNS) and the other one is flow-based. Of the two packet-based models, our SYN model is novel in fingerprinting IoT traffic. Our models run in a progressive manner on demand; that is, when a model is not confident about a device, our system uses the next model (more complex) for that specific device, and (3) We evaluate our progressive inference scheme with real traffic of 26 IoT devices, and demonstrate its efficacy in classifying IoTs with 99.4% accuracy while detecting reflection and flood attacks.

[1]This project was supported by Google Faculty Research Awards.

II. Related Work

In the context of IoT network monitoring, the common approach for classifying devices is to collect network traffic of IoTs and extract some features from it then use them for training a machine-learning classifier. In general, there are two types of network telemetry (traffic features) that are mainly used in the literature: (1) packet-based attributes that are extracted from each packet (requires packet inspection) *e.g.,* port numbers, domain names, HTTP User-Agent, and occurrence of specific protocols, and (2) flow-based attributes that are statistical information *e.g.,* number of DNS queries, mean traffic rate, and packet count.

Works in [8]–[11] use packet-based features. Examples of packet features used in these works are the occurrence of certain protocols (*e.g.,* TCP, UDP, ARP, NTP, DNS, and ICMP), TCP window-size, IP type-of-service field, TTL, IP do-not-fragment flag, size of packet, payload entropy, payload size, and TCP/UDP port numbers. Although these works yield high accuracy in classifying IoTs, they require a huge amount of computational power to inspect every packet to/from IoT devices. This can become very expensive or even infeasible for real IoT networks that serve thousands of devices.

The other type of network measurement is flow-based, which uses statistical information on behavioral activities for inference models. These attributes can be mean, min, max, standard deviation, or other statistical properties of flow measurements (*e.g.,* packet count). Authors in [12] proposed a two-stage classification which uses six flow-based attributes like DNS interval, sleep time, and flow rate along with three packet-based attributes (port numbers, domain names, and cipher suites). They showed that by using only low-cost attributes like flow volume, duration, and mean rate, a high accuracy (more than 95%) could be achieved. The authors used SDN to capture these features by inserting some OpenFlow rules for each device. This approach may work for networks with hundreds of devices, but for larger networks, it requires inserting thousands of OpenFlow rules into a switch that can affect both computation cost and system performance.

DEFT [13] is another work that applies both packet-based and flow-based attributes. Samples of packet features include HTTP request URI, HTTP method, HTTP response code, and MQTT packet type. Some of the flow-based features are DNS packet count, DNS packet size, SSDP flow duration, and NTP packet size. DEFT is distributed but still requires an all-packet inspection for every device on the network.

Most of the mentioned works need packet inspection, which imposes high computation cost, resulting in scalability issues. Also, they do not consider an important fact that some IoT devices display a uniquely identifiable pattern in specific packets or flows. A study in [16] shows that some IoT devices have a fixed small set of network activity. Overlooking this fact leads to developing a single heavy model that takes tens of features for every device while it is not needed in many cases. Computing these unnecessary features can have a significant impact on computation cost and execution time.

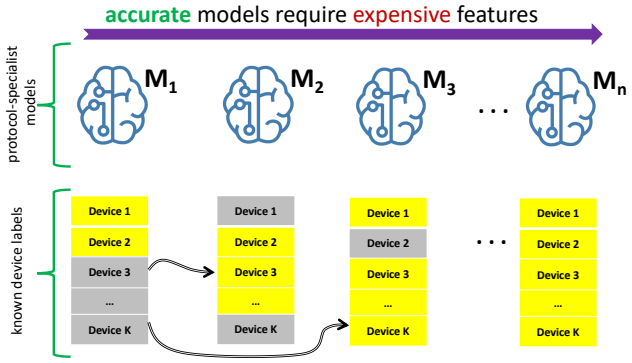


Fig. 1. Progressive inference from device traffic.

III. Dynamic Inference from IoT Network Traffic

Existing works have focused on developing a single model for IoT classification and monitoring. Purely using packet-based features requires **inspection of individual packets** that is **expensive** and provides **limited visibility** into network activities that can lead to missing devices behavior in broader context. Neglecting flow-based characteristics would limit the insights obtained from the behavior of devices at an aggregate level. For instance, in DoS and reflection attacks, the packet signature of a device response to attacks query may not necessarily change, so by only inspecting packets, we might not detect sophisticated attacks. Instead, periodic flow-based statistics like packets count can clearly highlight a volumetric attack. On the other hand, by only using flow-based features, we may lose fine-grained visibility into the traffic by ignoring packet signatures that can be helpful in IoT classification and/or attack detection. We believe a desirable monitoring solution would utilize both packet-based and flow-based features.

One of the main drawbacks of prior works is that most of them demand a large amount of data for each device type to train and test their machine-learning model. However, we note that some IoT devices are very simple and have some unique features in their activity, which can be used to distinguish them among other devices. For example, Nest smoke sensor only uses TCP port 11095 in all of its communications that is unique across all IoT devices we have in our lab. In contrast, Amazon Echo uses 80, 443, 840, 8692 ports. Of these four port numbers 80 and 443 are used by many other devices as well. In this example, we are able to distinguish Nest smoke sensor by seeing the port number 11095, and hence no further traffic inspection is needed. However, for sophisticated and complex devices like Amazon Echo, the port number is insufficient, so we need more features. We believe that some devices can be handled by less expensive models with less amount of information from their network traffic.

Another trouble with a single comprehensive model for a wide range of IoT devices is that its maintenance like retraining (due to legitimate change in traffic pattern) can become quite challenging. It also demands heavy machinery to run on. Also, having a single model increases the chance of failure and makes the model more vulnerable to adversarial

attackers that try to bypass the model with specially-crafted attack data. We believe that combination of multiple models makes the solution more robust against adversarial attacks.

These facts motivated us to develop a pipeline of models where, each model is specialized in a subset of traffic. This pipeline consists of multiple machine-learning classifiers that work with either packet-based or flow-based features. Fig. 1 illustrates an abstract structure of our progressive inference approach. In this pipeline, each model can be specialized in a subset of protocols; for example, the first model may use characteristics of specific TCP packets, the second model may be trained by features of DNS queries, and so on. SDN enables us to provide the required telemetry *e.g.,* a collection of packets and flow counters to infer the IoT devices. SDN helps us to minimize the cost of packet inspection by only mirroring the selected packets also it computes the flow counters at the hardware level without any further computation by our program.

At the beginning of the pipeline (on the left) models are slightly lightweight (processing minimal amount of traffic) but still can distinguish some device types fairly confidently (those that are shown by bright cells Fig. 1) – gray labels indicate low confidence, meaning more data is needed. Moving forward through the pipeline, models become more complex, and hence demand richer network telemetry. Devices that are classified unconfidently by the earlier models are passed on to following models – in certain circumstances (temporal variation of model output) confidently-labeled device may be further checked by later stage models. If the next model is still unable to classify the device confidently, we keep passing it through the pipeline. The last model is the most complex in the pipeline. It is more expensive and requires more network telemetry; therefore, it is confident about most of the devices.

The key challenge in this progressive pipeline is choosing the models. When a model only works with a subset of traffic, there is a chance that some devices do not generate that type of traffic for a considerable time interval. For example, Dropcam does not generate DNS queries for hours; LIFX bulb sends only one TCP SYN packet when it reboots. Thus, for this approach to perform well for any type of IoT device, it should not rely on the occurrence of a specific protocol. Instead, it attempts classifying devices with the lowest amount of data, but if unavailable, then it obtains more data.

For earlier stages of our pipeline, one can use packet signatures extracted from signaling protocols like DHCP, TCP SYN/SYN-ACK, DNS, or NTP. This significantly decreases the number of the inspected packets. In contrast, for later stages, since more data is needed, we can use a collection of flow-based behavioral features without requiring packet inspection.

We will develop three models (§IV) which collectively contain the cost of packets inspection while giving an acceptable accuracy (§V).

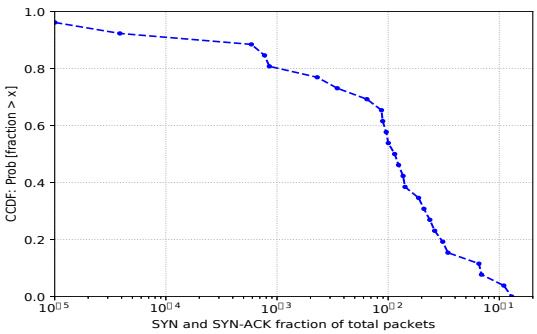


Fig. 2. CCDF of SYN and SYN-ACK fraction of total packets in IoT devices.

IV. Modeling IoT Network Traffic

In this section, we develop three models for our progressive pipeline. Two models are packet specialist (TCP SYN and DNS), and the third model is flow based.

A. TCP SYN model

We begin by analyzing characteristics of TCP SYN and SYN-ACK packets. These packets are sent in the three-way handshaking phase to establish a TCP connection. We extracted TCP handshake packets from PCAP traces of 26 IoT devices in our lab, and found that following header fields display identifiable patterns across various IoT types to some extent. Fig. 2 shows the CCDF of SYN/SYN-ACK fraction of packets generated by our devices over three weeks. It can be seen that this ratio is less than 4% in traffic of more than 80% of IoT devices in our lab, and hence cost-effective in terms of the number of inspected packets.

TCP window size: A 16-bit field in TCP header that is used to specify the number of bytes that sender is ready to receive. We observed 19 distinct window size values across 26 IoT devices. Ten devices each uses a unique window size value (each ranging from 512 to 33580). Some devices like Google home, Belkin camera, Withings sleep sensor and Samsung smart camera each had two different window size values in their SYN/SYN-ACK signatures.

TCP options [17]: A list of pre-defined options, each serves for a specific purpose. "Kind", "length", and "data" are the three fields that define an option entry. We found that distinct devices have different set of options and even the order of the option entries are different across devices. Below, we explain individual options we used for signatures:

- **End of Option List**: This option is used to declare end of the list only when the TCP header is not ended by the options. Out of 26 IoT devices only four (Netatmo weather station, Awair air quality monitor, LIFX light-bulb, and Ring doorbell camera) use this option in their option list.
- **No-Operation**: It is used among the other options to make them a multiplier of 8 to improve performance. 19 devices use this option in their option list which 10 of them use it once and exactly before the window scale option. This option appeared three times in HP printer's signatures which is unique.

TABLE I
PERFORMANCE OF INDIVIDUAL PROTOCOL-SPECIALIST MODELS ACROSS 26 IOT DEVICES.

Model	Google Home	Google Chromecast	Amazon Echo	Canary camera	Triby speaker	Netatmo weather	August doorbell	Belkin motion	Belkin switch	Pixstar photo-frame	iHome plug	Netatmo camera	Samsung camera	NEST smoke	HP printer	Awair air-quality	Withings body	Ring doorbell	TP-Link camera	Hue bulb	LIFX bulb	SmartThings	Dropcam	Bipcare BP	Belkin camera	Withings sleep
SYN SYN-ACK	44% [0.75]	100% [1.00]	100% [1.00]	90% [0.59]	100% [1.00]	100% [1.00]	40% [0.79]	100% [0.99]	100% [1.00]	82% [0.63]	100% [0.99]	100% [0.99]	100% [1.00]	100% [1.00]	100% [0.92]	100% [1.00]	100% [0.99]	100% [0.99]	100% [0.83]	100% [0.99]	100% [1.00]	100% [0.99]	100% [0.97]	100% [0.85]	100% [0.99]	100% [0.99]
DNS	98% [0.99]	99% [0.97]	100% [1.00]	100% [1.00]	100% [1.00]	100% [1.00]	100% [1.00]	47% [0.99]	100% [1.00]	100% [1.00]	100% [1.00]	100% [1.00]	27% [0.88]	100% [0.95]	75% [0.88]	100% [1.00]	100% [1.00]	100% [0.96]	100% [1.00]	100% [0.99]	100% [0.97]	98% [1.00]	100% [0.99]	100% [1.00]	100% [0.82]	100% [0.99]
Flow	92% [0.99]	96% [0.99]	99% [0.99]	99% [0.97]	96% [0.99]	94% [0.99]	99% [0.99]	100% [0.99]	100% [0.99]	99% [0.79]	100% [0.96]	100% [0.99]	100% [0.99]	95% [0.91]	100% [0.99]	100% [0.99]	96% [0.79]	99% [0.98]	100% [0.99]	100% [0.99]	100% [0.99]	100% [0.99]	100% [0.99]	100% [0.90]	100% [0.99]	100% [0.99]

- **Maximum Segment Size**: It defines the maximum size of a TCP segment and must be used in the three-way handshaking phase. Devices usually set this field based on the maximum transmission unit so IP fragmentation does not occur. We found four devices (iHome power plug, Netatmo weather station, Netatmo camera, and Smart Things) each used unique value for this option, three devices shared the value 1152, and 19 devices shared the value 1460.
- **Window Scale**: TCP window scale was introduced to improve the efficacy of TCP by multiplying window size to value of the window scale. 16 IoT devices in our lab use this option in their SYN/SYN-ACK signatures – only seven unique values were observed in the data field of this option across these 16 devices.
- **Selective Acknowledgment**: SACK enables the receiver to send acknowledgment out of order. To use SACK, client and server must agree on it with SACK-permitted option in the three-way handshake. 17 devices send SACK-permitted in their SYN packets' header.
- **Timestamp**: Timestamp was defined in the standard because of RTTM (Round Trip Time Measurement) and PAWS (Protect Against Wrapped Sequences) algorithms. 12 devices send timestamp option in their SYN/SYN-ACK header.

IPv4 Don't Fragment flag: This one-bit flag in IPv4 header, indicates that the packet should not be fragmented along the route. 17 devices set this flag to avoid IP fragmentation.

Inspired by p0f [18], we develop our colon-separated signatures of TCP SYN as follows:

$$\{syn/synack\} : \{f/df\} : [windowsize] : [option\#1kind]\_[option\#1datavalue] : ... : [option\#nkind]\_[option\#ndatavalue]$$

The fields by their order indicate: the packet is SYN or SYN-ACK, fragmented (f) or not (df), the integer value of the TCP window size, and the list of options with the same order that was sent in the TCP SYN header. Some options only have kind, and some others have a data value (*e.g.,* maximum segment size, and window scale). We do not add the data value for some options like selective acknowledgment and timestamp because they change in each packet. In addition to the signature, we add port numbers to increase the accuracy and uniqueness. We use destination port number of SYN packets and source port number of SYN-ACK packets.

We extract the mentioned fields from TCP headers and create the signatures periodically. This means that we consider aggregate traffic over an epoch (say, 5 minutes) and create our instances accordingly. Thus, each instance in our dataset is a list of signatures along with a list of corresponding port numbers a device communicated with over an epoch.

1) SYN model training: Our testbed contains 26 IoT devices connected to a router with OpenWRT installed on it. On the router there is *tcpdump* which captures all incoming/outgoing packets and dumps them into a daily PCAP file. Our traffic traces span six weeks.

For this model, we only analyze SYN/SYN-ACK packets that individual IoT devices send. We use Bag-of-Words to vectorize the string values of the SYN/SYN-ACK signatures. Using Bag-of-Words, we implicitly count the number of packets in each epoch as well, which makes our model more accurate. We add three words, namely "unknown syn", "unknown syn-ack", and "unknown port" to our vocabulary and map any new value to one of these words to avoid ignoring any unseen values. With this method we capture new values and the model will give a low confidence upon seeing an instance with unknown features.

Following dataset pre-processing (vectorizing), we split it into two sets of training (70%) and testing (30%). We train a Random Forest multi-class classifier. Note that we also tried other models like Naive Bayes, Logistic Regression, and KNN, and found that Random Forest outperforms all of them by accuracy. Table I shows the accuracy of each model across 26 IoT devices. Accuracy is presented in percentage and confidence level is shown in brackets – red color cells highlight device types which receive low accuracy/confidence from respective models.

2) Problems with the SYN model: Although this model is very lightweight, there are some devices that cannot be classified confidently. For example, August Doorbell and Canary camera use the same TCP options list and remote TCP port numbers. Additionally, the number of SYN packets they send to their cloud server in each epoch is very similar. As another example, we noticed that some devices like LIFX lightbulb and Dropcam, only generate one SYN packet on reboot and then maintain that connection for hours (even days). This becomes a problem because these devices are active, generating traffic

but with only the SYN model we cannot identify them. These issues motivated us to develop other models that can address these problems.

B. DNS model

DNS is a commonly used protocol used by IoT devices which frequently communicate with cloud servers. In the DNS query, a device indicates the domain name that it wants its IP address. IoT devices communicate with a limited number of endpoint cloud servers [12] which their domain names could be very indicative for detecting device type and manufacturer.

For the DNS model, we use the domain name extracted from the DNS queries. Then similar to the SYN model, by using Bag-of-Words and Random Forest, we train the DNS model classifier. The accuracy of this model is 98.2%. DNS model is very cost-effective in terms of the number of inspected packets (*i.e.,* no more than 10% of traffic).

Similar to the SYN model, there are two cases where DNS model loses accuracy/confidence. (1) Some devices have similar DNS queries, specially when they are from the same manufacturer (*e.g.,* Belkin motion sensor and Belkin camera). (2) Some devices may not send any DNS query for hours (*e.g.,* Dropcam) though they are active.

C. Flow model

For our system to be able to work for every device, we develop a final model based on flow statistical information. Inspired by [15], we use ten flow rules to measure flow counters (packet and byte) of specific protocols during every epoch. Our rules are inserted with three levels of priority to avoid ambiguity between the overlapping rules. We assign higher priority to the selected rules because they use TCP/UDP as the underlying transaction protocols. ARP and TCP/UDP have medium priority because they have overlap with total↑.

- High priority: SYN↑, SYN-ACK↑, DNS↑, NTP↑, SSDP↑
- Medium priority: TCP↑, UDP↑, ARP↑
- Low priority: remote↓, and total↑

Note that ↑ and ↓ respectively indicate outgoing (from device) and incoming (to device) direction of traffic. For our features, we use packet count and byte count of three flows including DNS↑, remote↓, and total↑, and only packet count for the remaining seven flows – in total, 13 features.

We train a Random Forest model based on above features that gives an average accuracy of 98.7%. Because the features used in this model are ordinal (in contrast to Bag-of-Words in the previous models), it required three weeks of additional training data to achieve a reasonable accuracy/confidence.

V. System Evaluation

We now evaluate the efficacy of our proposed scheme. Fig. 3 shows the architecture our SDN-based system empowered by trained models on top – for brevity we do not show the SDN controller. We replay PCAP traces of 26 IoT devices (collected over a day in our lab).

Having the three models trained, we record the mean (μ) and standard deviation (σ) of models confidence per each

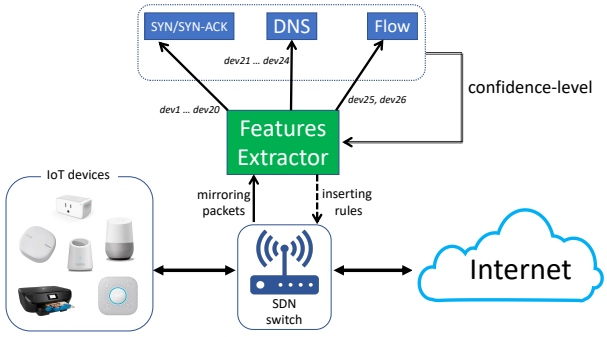


Fig. 3. System architecture of our prototype.

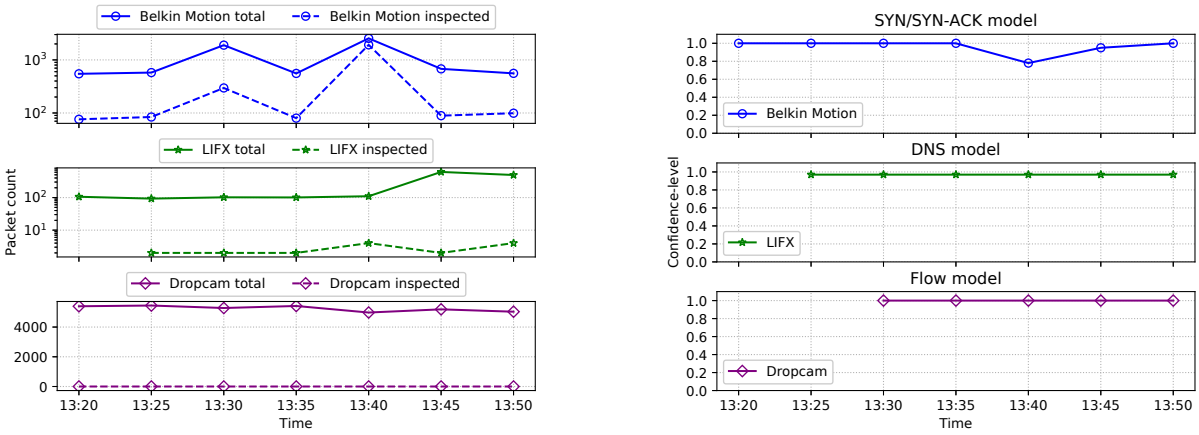
device label. Given a model, if its average confidence (μ) for training instances of a device label is less than a threshold (say, 90%) then we mark that device as a gray label for the model. This happens when several devices share traffic signatures or the model is not trained by sufficient amount of data from that class. Considering μ values, we associate each device with a model that has the lowest cost. For example, due to low accuracy/confidence of the SYN model for Google home device, we assign it to the DNS model. During testing phase, we check in real-time (every epoch) whether the resulted confidence for each subject device is greater than an expected value (we consider $\mu - 0.5\sigma$), if not, the device gets flagged as anomaly, and all of its traffic is mirrored to an IDS.

The SYN model is default for every IoT device. This means that the SDN switch is first inserted by two rules that mirrors only SYN and SYN-ACK packets of every device to a "feature extractor" engine. At the end of each epoch, if no SYN/SYN-ACK packet is captured for a device on the mirror port, then feature extractor inserts a new mirroring rule (DNS packets) only for that device. Similarly, in case of no DNS query from the expected devices, the system enters into the last stage whereby ten device-specific rules (§IV-C) are inserted to generate required telemetry for the flow model.

We plot a time trace (30 minutes) of our system evolution for three representative devices in Fig. 4. Note that one device (Belkin motion sensor shown by top row plots) was under SYN flood attack for a short duration (13:35 - 13:40)

Fig. 4(a) shows the total number of packets these devices generated versus the inspected packets. We observe that the number of inspected packets in SYN/SYN-ACK and DNS models is one order of magnitude less than the total packets. Also, in case of Dropcam (bottom row) for which the Flow model is used, no packet is inspected. Fig. 4(b) illustrates the confidence level of each model for the assigned device. SYN model has high confidence level ($>90\%$) for these devices (Table I), thus they are initially assigned to this model.

At the end of the first epoch (13:20) the engine noticed that though LIFX was assigned to the SYN model, it did not send any SYN or SYN-ACK packet. Thus, the engine decided to assign LIFX to the next model (*i.e.,* DNS). Also, at the same time, similar situation happened to Dropcam. Dropcam was assigned to the SYN model, but it did not generate any of these packets; so, it was assigned to the DNS model for the next epoch. At the end of the next epoch (13:25), LIFX is



(a) Total vs. inspected packets

(b) Confidence of protocol-specialist models.

Fig. 4. Time trace of: (a) total versus inspected packets count, and (b) confidence-level of protocol-specialist models, for three representative IoT devices.

classified correctly by the DNS model with high confidence. However, Dropcam did not send any DNS queries. Therefore, the engine assigned Dropcam to the flow model so from the next epoch (13:30), all of them are assigned to the appropriate model and are classified with high confidence.

Accuracy of the individual models are 89%, 98.2%, and 98.7% for SYN, DNS, and flow model respectively. However, the overall accuracy of the system when devices are assigned dynamically to the models, is above 99.4%. This means that for an accurate and cost-effective solution, a single perfect model is not necessary, instead, multiple models that are fairly accurate for certain devices can achieve this result.

A. Verifying Attack Detection

In general, any vulnerability or attack, that forces a device to generate SYN, SYN-ACK packet that is not benign (*e.g.,* SYN flood and reflection attack), can be detected by the model due to a significant confidence level decrease. Our SYN model is able to accomplish this, because by using Bag-of-Words we are implicitly counting the number of SYN/SYN-ACK packets that devices usually send in each epoch, which significantly increases during the attack.

Fig. 4 shows the result for attack on Belkin motion sensor. From 13:35 to 13:40, we launched a SYN flood attack on Belkin motion sensor. It is shown in Fig. 4(b) that during this attack, the model confidence drops by about 25%. By capturing this drop, we can forward the device traffic to Zeek, an off-the-shelf IDS. If the attack continues till next epoch, Zeek will detect and flag it, and the device needs to be quarantined (by automatic insertion of a reactive SDN rule).

It is important to mention that attack detection was not the main scope of this work. Instead, we considered a cost-effective subset of IoT traffic for inference, and hence some attacks may go undetected. Our future work will focus more on detecting attacks and anomalies.

VI. Conclusion

In this paper we developed progressive inference engine for classifying and monitoring network traffic of IoT devices. We addressed the shortcoming of existing inference models which are static, expensive, and inflexible. We developed a progressive inference engine consisting of multiple models each with different accuracy and cost that can be used for specific devices. We employed SDN to dynamically acquire different types of network telemetry. We evaluated the efficacy of our system using three Random Forest models that collectively achieve a high accuracy 99.4%.

References

[1] A. Gupta et al. Forecast: Internet of Things — Endpoints and Associated Services, Worldwide, 2017. Technical report, Gartner Research, Dec 2017.

[2] McKinsey. Growing opportunities in the Internet of Things, Jul 2019.

[3] Forescout. Network visibility survey. http://bit.ly/30LBGaf, 2016.

[4] Minzhao Lyu et al. Quantifying the Reflective DDoS Attack Capability of Household IoT Devices. In *Proc. ACM WiSec*, Boston, Massachusetts, USA, Jul 2017.

[5] NJ cybersecurity & communications integration cell. Mirai. http://bit.ly/2RgwVlI, 2016.

[6] Black Lotus Labs. Attack of things! http://bit.ly/36hP07n, 2016.

[7] Pierluigi Paganini. The Linux Remaiten malware is building a Botnet of IoT devices. http://bit.ly/30PNwQK, 2016.

[8] B. Bezawada et al. Behavioral fingerprinting of IoT devices. In *Proc. of the ACM CCS*, Toronto, Canada, October 2018.

[9] D. Kumar et al. All Things Considered: An Analysis of IoT Devices on Home Networks. In *28th USENIX Security Symposium*, Santa Clara, CA, USA, August 2019.

[10] K. Yang et al. Towards automatic fingerprinting of IoT devices in the cyberspace. *Computer Networks Journal*, 148:318–327, 2019.

[11] M. Miettinen et al. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *Proc. of ICDCS*, Atlanta, CA, USA, June 2017.

[12] A. Sivanathan et al. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2019.

[13] V. Thangavelu et al. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet of Things Journal*, 6(1), 2019.

[14] A. Sivanathan et al. Characterizing and classifying IoT traffic in smart cities and campuses. In *IEEE INFOCOM WKSHPS*, Atlanta, GA, USA, May 2017.

[15] A Sivanathan et al. Managing IoT Cyber-Security using Programmable Telemetry and Machine Learning. *IEEE TNSM*, 2020.

[16] A. Hamza et al. Clear as MUD: Generating, validating and applying IoT behavioral profiles. In *ACM SIGCOMM IoT S&P*, Budapest, Hungary, 2018.

[17] IANA. Transmission Control Protocol (TCP) Parameters. http://bit.ly/2RgSsea, 2019.

[18] p0f. http://lcamtuf.coredump.cx/p0f3/, 2016.