Computers & Security 97 (2020) 101968

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

# A novel approach for detecting vulnerable IoT devices connected behind a home NAT☆

Yair Meidan[a,*], Vinay Sachidananda[b], Hongyi Peng[b], Racheli Sagron[a], Yuval Elovici[a], Asaf Shabtai[a]

[a] Ben-Gurion University of the Negev, Ben-Gurion Blvd. 1, Beer-Sheva, 84105, Israel
[b] National University of Singapore, 21 Lower Kent Ridge Rd, 119077, Singapore

ARTICLE INFO

*Article history:*
Received 1 April 2020
Revised 14 July 2020
Accepted 16 July 2020
Available online 17 July 2020

*Keywords:*
Internet of things (IoT)
Device identification
Network address translation (NAT)
Machine learning
DeNAT

ABSTRACT

Telecommunication service providers (telcos) are exposed to cyber-attacks executed by compromised IoT devices connected to their customers' networks. Such attacks might have severe effects on the attack target, as well as the telcos themselves. To mitigate those risks, we propose a machine learning-based method that can detect specific vulnerable IoT device models connected behind a domestic NAT, thereby identifying home networks that pose a risk to the telcos infrastructure and service availability. To evaluate our method, we collected a large quantity of network traffic data from various commercial IoT devices in our lab and compared several classification algorithms. We found that (a) the LGBM algorithm produces excellent detection results, and (b) our flow-based method is robust and can handle situations for which existing methods used to identify devices behind a NAT are unable to fully address, e.g., encrypted, non-TCP or non-DNS traffic. To promote future research in this domain we share our novel labeled benchmark dataset.



## 1. Introduction

The ability to launch massive distributed denial-of-service (DDoS) attacks via a botnet of compromised devices is an exponentially growing risk in the Internet of Things (IoT) (Antonakakis et al., 2017; Bertino and Islam, 2017; Hallman et al., 2017; Kit, 2016a; Kolias et al., 2017; Rayome, 2017). Such massive attacks, possibly emerging from IoT devices in home networks (Kambourakis et al., 2017), impact not only the target of the attacks, but also the infrastructure of telecommunication service providers (telcos) along the attack path as well. Given the greater bandwidth available to customers today, the combined traffic surge from a botnet of infected IoT devices might hit the telco's infrastructure and eventually overload it. This could cause episodic downtime and serious backlashes in the form of widespread customer dissatisfaction.

Two recent massive IoT-based cyber-attacks exemplify the propagation and real-world impact of such attacks on telcos. In one of them (Kirk, 2016; Krebs, 2016), over 900,000 customers of the German ISP, Deutsche Telekom (DT), were knocked offline for several days because of an IoT-based DDoS attack caused by a Mirai variant. This attack disrupted domestic Internet connections, as well as telephony and television services. According to Krebs (2016), this attack relied on exploiting a security flaw in *specific models* of vulnerable and poorly-secured Internet-connected cameras and other IoT devices. In another attack (Haran, 2016; Kit, 2016b; Than, 2016), the home broadband network service of StarHub, a Singaporean telco, suffered from lengthy disruptions due to massive DDoS attacks which similarly took advantage of vulnerabilities found on specific webcam and router models.

In cases where instances of vulnerable IoT models (i.e., IoT devices) are connected to home (or non-security oriented SMEs) networks, most customers don't have the awareness, knowledge, or means to prevent or handle ongoing attacks, and so the burden of preventing them falls on the telco. To effectively defend against IoT-based attacks launched from customers' premises and avoid a

☆ This work was done when the author Vinay Sachidananda was at Singapore University of Technology and Design, Singapore.

* Corresponding author.
*E-mail addresses:* yairme@post.bgu.ac.il (Y. Meidan), comvs@nus.edu.sg (V. Sachidananda), hongyi_peng@u.nus.edu (H. Peng), elovici@bgu.ac.il (Y. Elovici), shabtaia@bgu.ac.il (A. Shabtai).

https://doi.org/10.1016/j.cose.2020.101968

disruption of services to their subscribers, measures to quickly detect and respond to such attacks must be put in place by telcos (Kit, 2016b). This can be accomplished by issuing a software update, the way DT has done (Kirk, 2016). Another possible measure is to continuously monitor the network traffic to detect exploitation attempts, infections, or attacks which are targeted at third parties and then block these activities. However, this approach requires substantial overhead, e.g., overhead related to planning, deploying, and maintaining the monitoring system(s). In addition, although continuous monitoring is able (to some extent) to detect attacks when they are executed, it might prove to be too late for the parties involved. That is, the target of the attack might already be hit, the domestic subscribers might suffer from service malfunctions, and the telco's infrastructure and reputation might be damaged by that time.

In this research, motivated by the real-world examples above as well as substantial long- and short-term DDoS-related impact on telcos (see Section 5.1), we consider an additional measure for telcos to use in their defense against IoT-based attacks launched from their customers premises and propose a method for detecting connected vulnerable IoT models *before* they are compromised. This method, which takes a preemptive approach, can assist telcos in identifying potential threats to their networks and enable them to take preventive actions in a timely manner. Our method monitors the traffic of each smart home separately to **verify whether specific IoT models, known to be vulnerable to exploitation by malware for the purpose of executing cyber-attacks, are connected to the home network**.

Home networks typically use gateway routers with NAT (network address translation (Egevang et al., 1994)) functionality, which, in order to decrease the number of public IP addresses, replaces the local (private) source IP address of each outbound packet with the globally unique public IP address of the router. Consequently, since the outbound traffic which emerges from all the various hosts that are connected behind the NAT are identified with the same source IP address (i.e., the router's address), detecting connected IoT models from outside the network is a challenging task (see Section 3.1). By using our proposed method, a telco can (1) detect vulnerable IoT devices connected behind a NAT, and (2) use this information to take action. In the case of a potential DDoS attack, our method would enable the telco to take steps to spare the company and its customers harm in advance, such as offloading the large volume of traffic generated by an abundance of infected domestic IoT devices; in turn, this could prevent the combined traffic surge from hitting the telco's infrastructure, reduce the likelihood of service disruption, and ensure continued service availability.

In essence, for every vulnerable IoT model we propose to (a) collect flow level traffic metadata, (b) train a machine learning-based classifier in a centralized manner, and then (c) deploy the classifier using local detectors, each of which monitors a smart home and can take steps to mitigate the risk once it detects the existence of a vulnerable IoT device.

We summarize our paper's contributions as follows:

1) Our method addresses a challenging real-world problem that has already caused adverse effects in Germany (Krebs, 2016) and Singapore (Kit, 2016b), and poses a risk to the infrastructure of telcos and their customers worldwide.
2) Our method does not rely on packet level analysis which is relatively intrusive and inefficient in terms of computation and storage (Sivanathan et al., 2016). We use compact flow level statistics (without analyzing the payload at all) as opposed to deep packet inspection (DPI). Moreover, our method operates at the *individual* flow level and still provides excellent detection performance.
3) Although several methods have been proposed for the analysis of NATed traffic, only a few of them (Guo and Heidemann, 2018; Li et al., 2019) address the challenge of IoT device identification. Moreover, we empirically show that our method is robust and performs well in situations where existing methods are only partially applicable, e.g., encrypted, non-TCP or non-DNS traffic.
4) We empirically evaluate our method using genuine traffic data collected from various commercial home IoT devices over a substantial period of time, and demonstrate that we can quickly and accurately detect IoT models behind a NAT. Unlike some past studies which evaluated their methods using partially, questionably, or completely unlabeled datasets, or just one type of device (Li et al., 2019), our data is versatile and explicitly labeled with the device model. We share [1] our experimental data with the scientific community as a novel benchmark to promote future reproducible research in this domain.

## 2. Use case, scope, and assumptions

### 2.1. Fundamental use case

Throughout this paper, we assume a telco which strives to proactively protect its infrastructure and services from harm. In the fundamental use case we address, the telco wishes to mitigate cyber risks originating from vulnerable IoT models connected to the networks of its home customers (B2C), typically behind a NAT. In this kind of setting, multiple hosts (e.g., laptops, smartphones, and IoT devices) can be connected behind the NAT (Guo and Heidemann, 2018), and the telco does not know how many of them are actually represented by the same (public) IP address. IoT devices could be exploited on a large scale to execute a DDoS attack (Antonakakis et al., 2017; Bertino and Islam, 2017), a scenario which could seriously harm the telco. In order to avoid such scenarios, the telco tries to detect vulnerable IoT models *before* they are exploited by adversaries, thus enabling the company to take preventive actions.

### 2.2. Scope

Within the vast domain of IoT security (Bertino and Islam, 2017; Khan and Salah, 2018), which pertains to both Wi-Fi and non-Wi-Fi based communication protocols (Ray and Bagwari, 2017), in this research, we focus on detecting the connection of vulnerable IoT models to domestic networks behind a NAT. In practice, we propose training a classifier for each vulnerable IoT model separately and using the trained classifiers for detection in smart homes. Note that we focus only on the *detection* of potentially harmful connections, as opposed to the *complete classification* or *mapping* of all of the connected devices. That is, for risk mitigation purposes we only wish to tell if the vulnerable IoT model is connected behind the NAT or not. If insufficient evidence is found for the connection of a vulnerable IoT model, it does not matter to us which devices are connected, whether IoT or not.

Also, although our method may be extendable to other use cases (see Section 7.1) and IoT application domains, in this research we concentrate on smart *homes*, a market which continues to grow (Technology Council, 2018), and more specifically, we limit the scope of this research to IoT models that connect to the router directly, via either a Wi-Fi or Ethernet connection. The network traffic of these IoT models is easier to (a) capture and monitor from outside the domestic network on a large scale (as opposed to Bluetooth or ZigBee), and (b) directly profile, since these IoT models

[1] http://doi.org/10.5281/zenodo.3924770.

don't connect via a smartphone or a multipurpose controller. Further, the IoT models we focus on have already been used in cyber-attacks in the real world (e.g., webcams Kit, 2016b; Krebs, 2016) or in research (e.g., smart light bulbs Ronen and Shamir, 2016).

### 2.3. Assumptions

#### 2.3.1. Definition of an IoT model

As previously mentioned, typically (Kambourakis et al., 2017) IoT-based DDoS attacks rely on exploiting the vulnerabilities of *specific* IoT models. In this context, we use a combination of four elements to define an *IoT model*:

1) Type: webcam, light bulb, socket, etc.
2) Manufacturer: D-Link, TP-Link, Efergy, etc.
3) Model number: DCS-942L, LB130, eGo, etc.
4) Firmware version

For example, TV.Samsung.UE55D7000.(undisclosed firmware version) is an IoT model which has been found to be vulnerable to a Mirai botnet variant (Spring, 2018) that utilizes a DNS amplification technique. The same malicious technique can be used to exploit another vulnerability found on router.FON.2601E-SE/RE/FSW-S/FSW-B.(firmware versions 1.1.7 and earlier), as listed under CVE-2019-6015 (Mitre, 2019). Like many other cases, the solution proposed (JVN, 2019) for this CVE was to update the firmware. This example supports the manner in which we define an IoT model, since a given combination of IoT type.manufacturer.model number no longer endangers the telco when it is patched with a new firmware version, and thus it no longer needs to be detected.

#### 2.3.2. Setup of the monitored networks.

For any domestic network monitored we assume a typical setup: The IoT devices are connected to a gateway router, mostly via Wi-Fi. This router provides an interface for connecting the IP-enabled devices to the Internet, and it has NAT functionality.

#### 2.3.3. Role and limitations of the telco

The telco is a passive listener that observes the traffic emerging from the customers' premises which are NATed. In addition, we assume that the telco does not know which applications are running when the traffic is being monitored.

#### 2.3.4. Exploitability of IoT devices

Various cyber-security aspects of IoT devices have been broadly discussed in the past (201, 2017; Khan and Salah, 2018; Zhang et al., 2014), including exploitation mechanisms (Andy et al., 2017; Shwartz et al., 2018) and real-world attack examples (Haran, 2016; Kambourakis et al., 2017; Krebs, 2016). Specific to our use case where IoT devices might be used to execute DDoS attacks, we note that an IoT device can be exploited either *before connecting* to the NATed network (e.g., via a supply chain attack (Omitola and Wills, 2018; Siboni et al., 2018)) or *while connected* behind a NAT, as in Hamilton (2016) where it was noted that Mirai's scanner is able to scan NATed devices which are configured with IP addresses if they have port forwarding enabled for ports 22/23. *Hole punching* (Shaked, 2019) is another method by which an external entity can reach a desired client on a network using a NAT. Accordingly, in this research we assume that some IoT models can be exploited even when they are connected to a home network behind a NAT. Then, based on this assumption, we focus on developing a method to detect the connection of such vulnerable IoT models using network traffic analysis.

## 3. Background and related work

### 3.1. NATing, deNATing, and IoT identification behind a NAT

The IoT device identification/detection methods proposed in the literature typically require the analysis of *chronological* sequences of packets (Lopez-Martin et al., 2017; Miettinen et al., 2017) or sessions (Meidan et al., 2017) for device (type) fingerprinting or even for the basic calculation of interarrival times as a predictive feature (Li et al., 2019; Mahalle, 2013; Radhakrishnan et al., 2015). However, in home networks (assumed in our fundamental use case) it is common to use NAT-enabled Wi-Fi routers (Gokcen et al., 2014; Maier et al., 2011). This makes IoT device identification difficult, since as part of *NATing* the outbound traffic, a NAT router effectively 'hides' the internal IP addresses of all of the individual connected devices by replacing them with the router's external IP address. Once NATed, it becomes a real challenge to correlate (from the outside) each packet to its packet stream, thus undermining the validity of existing IoT identification methods. In order to correlate subsequent (NATed) outbound packets to one another while maintaining their chronological order, one could attempt to use the destination IP and TCP sequence number. However, relying on these features might not be sufficient for IoT model detection when multiple hosts behind the NAT (Guo and Heidemann, 2018) communicate with the same destination IP, e.g., because they are produced by the same manufacturer (Apthorpe et al., 2017) or when UDP is utilized rather than TCP. The UDP protocol does not use a sequence number, and as demonstrated in Section 6.6.4, in many cases the share of UDP-based traffic is greater than 50%.

To overcome this challenge, several studies have proposed methods for *deNATing* (Orevi et al., 2017), which is the reverse action of *NATing*. DeNATing essentially aims at re-identifying the communication flowing through a NAT. Once performed, deNATing facilitates subsequent flow-based analyses, including IoT model detection, which we address in this study. In Sections 3.3 and 3.4, we survey existing deNATing methods and illustrate their shortcomings with regard to our use case. As opposed to most of the existing methods, we propose using the popular NetFlow protocol for deNATing, as it already aggregates related packets internally into flows. Although NetFlow practically performs deNATing internally to some extent, the detection of specific IoT models based on just a single flow remains a definitive challenge. We address this challenge using supervised machine learning algorithms.

### 3.2. NetFlow as a basis for IoT traffic analysis

In the domain of network traffic analysis several levels of data granularity are typically used to define an entity, such as a packet, transaction, session, flow, or conversation window (Bekerman et al., 2015; Callado et al., 2009). Among them, packet level traffic analysis is a common approach, wherein DPI is an accepted technique (El-Maghraby et al., 2017). This approach requires the capturing and analysis of highly detailed network traffic data, including the payload of each packet. Although potentially informative for IoT model detection, packet level analysis is relatively disadvantageous (Li et al., 2013) in terms of storage and computation efficiency. That is, the collection and analysis of all of the raw traffic (including the payload) in high traffic networks is technically challenging. For a telco, the traffic volume can reach up to multiple gigabits per second, and it is far from trivial to capture and analyze such a tremendous amount of data. In addition, although DPI technologies are in fact progressively gaining legal legitimacy (Stalla-Bourdillon et al., 2014), DPI puts user privacy at risk, as it allows the telco access to personal information. Of even greater concern, in cases where this data is leaked, there is a privacy risk for the communicating parties, whose private data might be exposed (e.g., video captures that

**Table 1**
The scope and orientation of prior deNAT and IoT identification related studies.

| Ref. | Context | | Objective | Subject of Interest | | | Assumed Environment | | |
|---|---|---|---|---|---|---|---|---|---|
| | DeNAT | IoT | | Person | Device | OS | Home, SOHO | Smart City | ICS, SCADA |
| Gokcen et al. (2014) | | | Identifying the presence of NAT and routing devices | | √ | | √ | | √ |
| Khatouni et al. (2019) | | | Detecting NAT devices | | √ | | √ | | |
| Savage et al. (2000) | | | Tracing back attackers (especially of DoS/flood attacks) | √ | | | √ | √ | |
| Verde et al. (2014) | √ | | Classifying NATed traffic as generated by a specific user or not | √ | | | √ | √ | |
| Tekeoglu et al. (2011) | √ | | Approximating the number of active nodes behind a NAT | | √ | | √ | | |
| Maier et al. (2011) | √ | | Counting NAT devices and active hosts behind each NAT | | √ | | √ | | |
| Ori et al. (2008) | √ | | Locating potential attackers hidden behind a NAT | √ | | | √ | | |
| Orevi et al. (2017) | √ | | DeNATing the traffic of devices which use the same OS (Win. 8 & 10, Android) | | √ | √ | √ | | |
| Ercolani et al. (2016) | | √ | Classifying a device type as originating from SCADA or not | | √ | | | | √ |
| Patton et al. (2014) | | √ | Assessing the threats originating from the usage of IoT devices | | √ | | √ | √ | √ |
| Hamad et al. (2019) | | √ | Identifying IoT devices connected to an organizational network using one IP flow | | √ | | | | |
| Apthorpe et al. (2017) | √ | √ | Inferring user behavior from encrypted NATed traffic | √ | | | √ | | |
| Guo and Heidemann (2018) | √ | √ | Identifying IoT devices potentially vulnerable to DDoS attacks such as Mirai | | √ | | √ | √ | √ |
| Li et al. (2019) | √ | √ | Discovering smart cameras behind a NAT | | √ | | √ | | |
| This paper | √ | √ | Identifying the presence of a vulnerable IoT model behind a NAT | | √ | | √ | | |

are transmitted over HTTP). Finally, the DPI approach is in general not feasible as many devices use encrypted communications, e.g., HTTPS which defeats DPI.

In this paper, we propose using NetFlow (cisco.com, 2017) as the basis for IoT traffic analysis. Collecting and analyzing only NetFlow's statistical aggregations (i.e., *meta*data) instead of the *raw* data requires significantly less computation and storage, thus making the analysis more efficient. Generally, this protocol was integrated as a feature in Cisco's routers and provides the capability of summarizing IP network traffic passing through an interface. This has become a common solution, natively supported by most routers (Verde et al., 2014), and it is acknowledged as the de facto standard for compact representation of large traffic volumes. In the academic literature, NetFlow has been used for security related tasks, like botnet detection and deNATing (Abt and Baier, 2012; Guo and Heidemann, 2018; Verde et al., 2014). Although the NetFlow protocol was introduced by Cisco, it is important to note that the features we use in our work are vendor-neutral, and they can also be acquired by other protocols from vendors other than Cisco, such as J-Flow (Networks, 2011) from Juniper and sFlow (AG, 2019) from HP.

### 3.3. Scope and orientation of previous studies

The action of *deNATing* is defined in Orevi et al. (2017) as re-identifying communications which flow in and out of a NAT. Table 1 summarizes past studies which are related to deNATing and/or IoT identification in terms of research scope and orientation. Papers in the table are indicated as *deNAT* (e.g., Guo and Heidemann, 2018; Li et al., 2019; Ori et al., 2008; Verde et al., 2014) if they specifically address the challenge of deNATing, as defined above (Orevi et al., 2017). In contrast, past studies, such as Gokcen et al. (2014) and Khatouni et al. (2019), involve the analysis of NATed traffic, however they are not considered here as deNAT papers, because they don't perform deNATing per se. That is, instead of dividing NATed traffic (all with the same public source IP address of the NAT router) into distinct packet streams for further analysis, they focus on identifying the *presence* of a NAT device in a network.

Regarding research objectives, most of the deNAT papers listed in Table 1 aimed to uncover the identity and/or quantity of the devices connected behind NAT routers, or the people who use them. Some motivations for doing so include the following:

- **Security**: Detecting attackers (Ori et al., 2008) or devices that are vulnerable (Guo and Heidemann, 2018; Li et al., 2019; Patton et al., 2014) or infected by a malware (Orevi et al., 2017)
- **Traffic management**: Applying policies such as parental browsing control or per device communication limits (Orevi et al., 2017), as well as traffic interception
- **Commerce**: Traffic profiling for targeted advertising (Orevi et al., 2017)
- **User tracking**: Inferring user connectivity (Verde et al., 2014) or behavior (Apthorpe et al., 2017)

Our motivation is also security-oriented; it reflects the perspective of a telco wishing to defend against IoT-based DDoS attacks by detecting and handling vulnerable devices connected to domestic NATed networks.

Unlike prior deNAT studies whose subject of interest was operating systems (Orevi et al., 2017) or people and their behav-

ior (Apthorpe et al., 2017; Ori et al., 2008; Verde et al., 2014), our method is designed to detect connected devices (instances of vulnerable IoT models). In contrast to other prior deNAT related studies which addressed large-scale environments like smart cities (Guo and Heidemann, 2018; Verde et al., 2014) and smart manufacturing (Guo and Heidemann, 2018), we tailored our method to smart homes. Of the deNAT papers listed in the table, most were designed for or evaluated using traffic generated by devices which run the Windows or Android operating system (OS), e.g., (Orevi et al., 2017; Verde et al., 2014). These OSs typically characterize PCs and smartphones, respectively, and they are uncommon in the IoT domain. The only papers we found that perform deNATing in the IoT domain (which is typically Linux-based (Taivalsaari and Mikkonen, 2018)) are Apthorpe et al. (2017), Guo and Heidemann (2018) and Li et al. (2019). However, Apthorpe et al. (2017) does not deal with device identification, and Guo and Heidemann (2018) and Li et al. (2019) have substantial shortcomings for our use case, as discussed in the next subsection.

### 3.4. Methods and evaluation in previous studies

Table 2 summarizes the primary features and related methods and algorithms for the studies listed in Table 1. Apparently, most of the methods rely on features extracted directly from the TCP/IP model. The problem is that in some cases those methods might fail to address our use case. For instance, features from the application layer like the HTTP user agent (Gokcen et al., 2014), MSN (Microsoft Network Messenger) transaction ID (Ori et al., 2008), and requested DNS domains (Apthorpe et al., 2017; Li et al., 2019) might not be available to the telco when the traffic is encrypted. In contrast, we use NetFlow which extracts data from different layers (such as the network and transport layers which are usually not encrypted), as well as *meta*data like traffic rates and volumes. The TCP timestamp is another commonly used feature in deNATing (Orevi et al., 2017; Tekeoglu et al., 2011), however its analysis requires a minimal number of packets, and it might even be disabled. Moreover, many IoTs use the UDP transport protocol at least to a certain extent, so the robustness of this feature becomes questionable. Our NetFlow-based method can handle both TCP and UDP, so it covers a wider range of IoT models for longer periods of time. Other approaches rely heavily on the server(s) the IoT devices communicate with (Apthorpe et al., 2017) and related DNS domains (Guo and Heidemann, 2018). With NetFlow, we can analyze all kinds of traffic and not just specific protocols such as DNS. Some prior studies make use of the IP TTL (Maier et al., 2011) based on the assumption that a NAT device iteratively decrements its value by one. However, the value at which it decrements might differ between various NAT devices, so generalization may be weak. Other research proposed using open ports (Ercolani et al., 2016) to deduce the traffic's origin, however those studies aimed at roughly distinguishing between SCADA and non-SCADA traffic. Our method is better suited for the fundamental use case presented (Section 2.1), as it proposes (and empirically delivers) fine-grained detection of IoT models of different manufacturers.

For empirical evaluation, most prior studies used tcpdump log files (Guo and Heidemann, 2018; Khatouni et al., 2019; Li et al., 2019; Orevi et al., 2017), Shodan scans (Ercolani et al., 2016; Patton et al., 2014), or simulated data (Savage et al., 2000). In some cases, the dataset is not available for research reproduction (Gokcen et al., 2014); in others, it is questionably labeled (Guo and Heidemann, 2018) or not at all (Maier et al., 2011); and in most cases, it does not represent smart homes. Moreover, even if labeling is present, the dataset either only comes from non-IoT hosts (Ori et al., 2008), captured for only a small amount of time (Li et al., 2019), or the class does not reflect the device model (Orevi et al., 2017). As opposed to them, in our research we use NetFlow records, collected in a controlled home-like lab for a period of 37 days from various IoT and non-IoT devices. Moreover, in contrast to Li et al. (2019) which focused only on (smart) webcams, we evaluated our method empirically using a variety of popular home (consumer) IoT devices, such as a smart light bulb, socket, doorbell, streamer, speaker, and webcams, as well as laptops and smartphones which served as a representative background traffic of a smart home (see Section 5.2). Most importantly, we explicitly labeled our data with the ground truth regarding the device models. Because of this, we believe that the novelty, authenticity, diversity, scope, and reliability of our publicly available dataset (Meidan et al., 2020) will facilitate future research and may serve as a benchmark for deNATing algorithms, specifically in the context of smart homes.

Among the past studies which are both deNAT and IoT device related, only Li et al. (2019) and Guo and Heidemann (2018) propose mechanisms for device identification, as we do. In Li et al. (2019), Li et al. proposed "CamHunter", a three-stage method for identifying smart cameras behind NATs. This method (1) screens for IP addresses which may contain smart cameras, (2) extracts heartbeat vectors, and (3) generates fingerprints from the vectors as a basis for classification. The first stage of CamHunter requires non-negligible manual efforts in exploring and characterizing protocol banners and DNS requests for each camera model separately. Moreover, as noted by the authors, data leakage via protocol banners raises privacy issues. So, in order to avoid such issues, manufacturers have paid more attention to privacy concerns and decreased their usage of protocol banners. As a result, this data source has become unreliable. DNS requests are also used in this stage, however they might be similar for dissimilar IoT models (possibly made by the same manufacturer) and thus lead to FPs. Another disadvantage of the method proposed in Li et al. (2019) to take into account is that both data sources (protocol banners and DNS requests) are easy to forge so compromised IoTs might evade detection. A key drawback in CamHunter lies in its second stage, where the heartbeat vectors are extracted from sequences of consecutive packets. Identifying subsequent packets of multiple connected devices can be easily done when all of the source IP and/or MAC addresses are visible (not NATed). This is how Li et al. (2019) trained their models. However, as today the number of hosts behind a NAT has increased (Guo and Heidemann, 2018; Maier et al., 2011), the separation of packets into distinct streams from individual devices becomes a true challenge for the passive listener on the other side of the NAT (i.e., the telco). Actually, this is the very challenge in deNATing (Orevi et al., 2017), and addressing it would require the use of additional methods which might not be accurate enough, and thus lead to incorrect packet grouping and ordering, and subsequently to incorrect feature calculation.

In another study whose objectives closely resemble ours, Guo and Heidemann (2018) suggested identifying NATed IoT device types based on per-type lists of communicated device-facing manufacturer server names (not human-facing or third parties). This idea resembles the underlying logic of Li et al. (2019), who noted that cloud-based devices require that DNS requests be sent to get the IP addresses of the devices' servers. Since different vendors provide different cloud platforms, those smart devices request distinct domain names, which can be utilized for network-based detection. Like Li et al. (2019), however, substantial semi-manual overhead is required to (1) identify server candidate names; (2) exclude third party and human-facing servers based on domain substring rules and HTML response size analysis using a manually defined threshold, respectively; (3) handle shared server names; and (4) track server IP changes over time and across networks. In their experiments, 158 distinct server names were identified for 10 IoT models, but significant IP changes were observed in the mapping

**Table 2**
The methods used in prior deNAT and IoT device identification related studies.

| Ref. | Primary feature(s) | | | | | | | | Methods and algorithms |
|---|---|---|---|---|---|---|---|---|---|
| | HTTP user agent | TCP time-stamp | Comm-unicated servers | DNS dom- ain | DNS IP- ID | IP TTL | Open ports | Other | |
| Gokcen et al. (2014) | ✓ | | | | | ✓ | | TTL range | C4.5 and Naive Bayes classifiers using traffic data processed by NetMate, plus a comparison with a passive fingerprinting approach based on the TTL range, IP TTL, and HTTP user agent string |
| Khatouni et al. (2019) | ✓ | ✓ | ✓ | | | | | Browser and OS information | Ten ML classifiers applied to flow level data extracted from four flow exporters (Argus, Netmate, Tranalyzer, and Tstat) |
| Savage et al. (2000) | | | | | | | | Packet marking | Employing edge marking to trace back to the origin of a packet, especially in cases of DoS attacks |
| Verde et al. (2014) | | | | | | | | IP addresses, ports, protocol, packets, bytes, TCP flags, start/end time, ToS | Per user HMM (hidden Markov model) classifier using NetFlow data |
| Tekeoglu et al. (2011) | | ✓ | | | | | | | Clustering TCP header timestamps of received packets using least-squares line fit, plus a convex hull for performance improvements |
| Maier et al. (2011) | ✓ | | | | | ✓ | | | Plain decision rules using IP TTL values for NAT detection and estimation of the number of hosts behind it, as well as the hosts' OS. In addition, employing HTTP user agent string plus OS and browser fields to distinguish between hosts |
| Ori et al. (2008) | ✓ | ✓ | | | ✓ | | | MSN Messenger TID, SMTP | Cluster-based similarity partitioning algorithm for creating similarity criteria between packets. After the initial clustering of each dedicated analyzer, a final clustering was performed to make a decision |
| Orevi et al. (2017) | | ✓ | | | ✓ | | | | Association of DNS requests using linear regression or clustering |
| Ercolani et al. (2016) | | | | | | | ✓ | | Segregation between SCADA and non-SCADA devices using Shodan scans and clustering of open ports, plus Gephi to visualize them |
| Patton et al. (2014) | | | | | | | | Header messages | Employing the Shodan search engine to access different IoT devices, identify them, and assess their security level |
| Hamad et al. (2019) | | ✓ | ✓ | | | | | Flow-based features | Extraction of behavioral and flow-based features from sequences of 20–21 packets, followed by multiclass classification (RF, kNN, SVM, etc.) |
| Apthorpe et al. (2017) | | | ✓ | ✓ | | | ✓ | | Division into traffic streams according to the communicated IP addresses and possibly also TCP ports, followed by device type identification and behavior inference using domains and traffic rates, respectively |
| Guo and Heide-mann (2018) | | | ✓ | ✓ | | | | | Identification of the communicated manufacturer's servers to infer the IoT device type |
| Li et al. (2019) | ✓ | ✓ | | ✓ | | | | Protocol banner, packet length and count | (1) Preliminary screening of IP addresses based on DNS requests and protocol banners; (2) Extraction of heartbeat flows and generation of time-series vectors (sequences of packet size, direction, and delivery intervals); (3) Classification of fingerprints using RF, kNN, and SVM |
| this paper | | | ✓ | | | | | Flow duration, size, destination, protocol, ports, etc. | Training a machine learning-based classifier for each (vulnerable) IoT model separately using NetFlow data, in order to detect its connection to domestic networks behind a NAT |

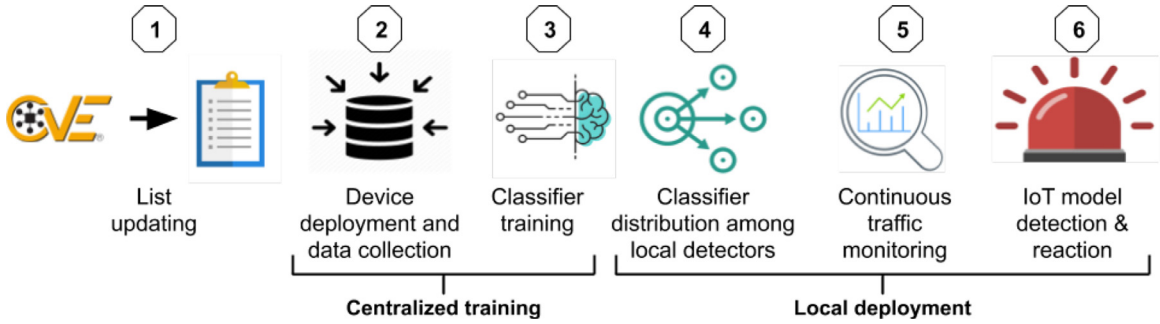


**Fig. 1.** An overview of the key steps in our proposed method.

of server name to IP. Additional shortcomings in Guo and Heidemann (2018) are (1) its inapplicability for IoT models with only a few servers, (2) its limitations in distinguishing between IoT models with shared servers, and (3) very long detection times (in the order of days). In contrast, our method is indifferent to the number of (shared) communicated servers and can detect IoT models instantly.

## 4. Proposed method for IoT model detection

For each item on a list of vulnerable IoT models to detect, we propose that the assumed telco train and deploy a machine learning-based classifier in order to detect connections of the respective IoT models to customers' (NATed) domestic networks. In order to maximize efficiency and control of the detection process, we advise the telco to frequently update the detection list (step 1 in Fig. 1), train the classifiers centrally (steps 2–3), and deploy them locally (steps 4–6).

A prerequisite for IoT device detection is to maintain a *detection list* which consists of the vulnerable IoT models of interest to the telco. Two well-established online resources which can be used as input to this list are the *Common Vulnerabilities and Exposures* (CVE) system (Mitre, 1999) and the *National Vulnerability Database* (NVD) (NIST, 2000).

### 4.1. Centralized training

For data collection, IoT devices of various models, as well as non-IoT devices (mostly computers and mobile devices), are to be deployed in the central lab, i.e., connected to its internal network behind a NAT (step 2). This central lab could be operated by the telco itself, possibly within cyber-security units that telcos operate to offer services besides Internet connectivity (e.g., Trustwave, 2015). Another option is that a third party could offer classifier training as a service or that the IoT device manufacturers could train classifiers for their products and share these classifiers with the telcos. Then, for an IoT model which has been identified as potentially vulnerable and thus is part of the detection list, respective devices (i.e., instances of that IoT model) are to be added to the central lab's network. Upon normal usage of the connected devices, network traffic data is generated and subsequently processed by NetFlow, which is installed on the NAT router. The resultant flows are continuously collected using nProbe and stored on a designated server, thus accumulating the raw flow level dataset. Then, a classifier for this IoT model is trained (step 3). This includes data preprocessing (encoding, normalization, feature engineering, etc.), flow labeling (*true* if the flow was generated by the given IoT model, and *false* if otherwise), data partitioning, and the application of machine learning techniques for training and fine-tuning the classifier.

Since the telco's objective is to *detect* specific IoT models rather than *map* the entire domestic network, we propose using binary (one-vs-rest) classification algorithms. In order to determine the classification threshold of a classifier, we recommend following the approach of *near-zero false positives* (NZFP) (Camiña et al., 2019). The reason to optimize for FPs is that a telco may prefer coping with a small amount of missed detections rather than handling the adverse effects of reacting to false alarms (Barapatre et al., 2008; Kashef and Barakat, 2018; Victor et al., 2010) (i.e., unjustified detection of NATed IoT models). Such adverse effects include, e.g., bearing excess costs of scrubbing large amounts of benign traffic, performing redundant or inappropriate virtual patching, and needlessly sending notifications to Internet subscribers, which might lead to customer dissatisfaction (see Section 4.3 for a list of possible reactions to IoT model detections). In practice, we propose setting the classification threshold as the value of the classification score which results in NZFP using a validation set. The exact value of "near-zero" (e.g., 0.00001) can be determined by the telco, such that it takes into account considerations such as the amount of FPs and FNs it can handle, the install base of the IoT model, and the likelihood of an upcoming cyber-attack.

### 4.2. Local deployment

Similar to Doshi et al. (2018) and Hadar et al. (2017), we propose using local agents to deploy the IoT model classifiers although typically a telco can already centrally monitor all of the packet contents from its clients' home networks. The main reason for doing so is that a central monitoring solution would probably result in a table that contains information on the IoT devices owned by specific customers. This table could become a valuable target for attackers (to steal information from numerous households at one time) or become a single point of failure. Another reason for preferring local agents over centralized monitoring stems from concerns regarding customers' privacy. That is, the classification results of the (locally deployed) IoT model classifiers would not be sent elsewhere but rather be used only locally as triggers to automated reactions (see Section 4.3).

Due to privacy preservation considerations, a telco should deploy the local traffic monitoring solution only from *outside* its customers' premises. Additionally, this solution cannot be implemented as a software agent on the home routers (location 1 in Fig. 2), because in many cases (e.g., M1, 2020), end users are not obligated to using any specific gateway router model, and the router is not necessarily part of the deal with the telco. Instead, in order for our method to be generic and supported by any home network setup, we propose situating hardware agents outside the customers' premises (location 2), between the home router and the ONT (location 3). These *local detectors* monitor the (NATed) traffic data which emerges from the home networks (step 5), and apply the pre-trained IoT model classifier in order to detect connected IoT devices of this model and respond accordingly (step 6). Each local detector can be implemented using a low-cost thin com-

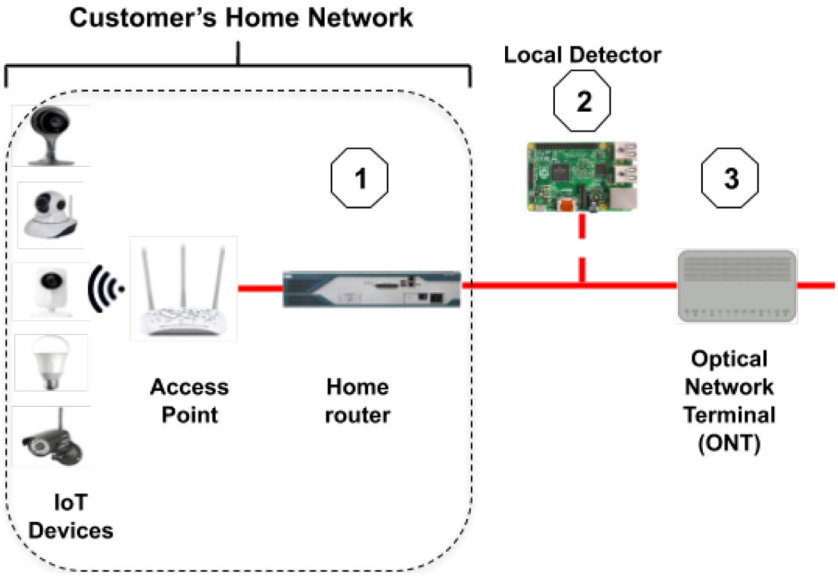


**Fig. 2.** Possible locations in the network where the IoT model detection method could be deployed.

puter (such as a Raspberry Pi), and should have the following software components: (1) nProbe for collecting flows from the monitored monitored home network, (2) a software environment (e.g., Python) for preprocessing the flows, (3) the trained classifier, and (4) a software component capable of making decisions and executing actions (described in Section 4.3) based on the classification results. As part of continuous monitoring, each flow collected by a local detector from a monitored home network is preprocessed exactly the way it was in the central lab, in order to achieve the same data structure and scale. Then, each flow is assigned a score by the IoT model classifier. To determine whether this flow was generated by a given IoT model or not, the classification score is compared with the classification threshold, whose value is optimized using a validation set such that it minimizes false detections (see Section 6.4 for an empirical evaluation of the NZFP approach).

### 4.3. Possible reactions to IoT model detection

As mentioned in Section 4.2, the local detectors are conceived to monitor outbound flows and react based on the classification results obtained by the IoT model classifiers. Following are examples of such reactions:

#### 4.3.1. Performing virtual patching

As proposed in Hadar et al. (2017), in order to prevent exploitation of vulnerable IoT devices, a telco can use (1) a cloud service, or (2) a designated IoT security appliance (such as the local detector we propose) to perform virtual patching. The security appliance monitors the vulnerable device's network traffic and verifies that it does not violate a set of predefined rules.

#### 4.3.2. Rerouting the traffic

Once a vulnerable IoT model is detected, the telco might want to tunnel the outbound traffic through a scrubbing center (Smithperrone and Sims, 2017) for deeper monitoring and inspection. If a malicious activity is recognized (i.e., the vulnerable device has actually been exploited and is being used to execute DDoS), the offending traffic can be dropped.

#### 4.3.3. Notifying the Internet subscriber

Doshi et al. (2018) via email or text message is another possible reaction to IoT model detection. As part of this notification, the customer would be informed that a vulnerable IoT device might be connected to their home network, and it is recommended that an appropriate action be taken by the customer. Changing a default password or updating the firmware are common solutions that today's IoT device users are likely able to perform. However, although technically feasible, this kind of reaction could be considered a violation of the customer's privacy, so we advise telcos to use it only if they receive customer consent for such notifications.

## 5. Evaluation method

### 5.1. Lab setup

In order to collect data, we set up a dedicated network representing a real-world scenario of various IoT and non-IoT devices connected behind a NAT, as illustrated in Fig. 3. First, we partitioned a switch (Cisco Catalyst 2960-X, 1G ports (Cisco, 2019)) into $VLAN_{in}$ and $VLAN_{out}$, representing the home network side and the telco side, respectively. Then, we connected various commercial IoT devices, as well as laptops and smartphones, to $VLAN_{in}$ via a wireless access point. We also connected $VLAN_{in}$ to a NAT router (Cisco 3825 (Cisco, 2006)) where NetFlow is installed. In turn, we connected the router to $VLAN_{out}$, which was connected to the Internet. To simulate the stages of *centralized training* and *local deployment*, we installed nProbe on a server and Raspberry Pi, to collect the NetFlow records from the router for further analysis. In addition, to enable the evaluation of previously conducted deNAT related studies (most of which rely on packet level data rather than NetFlow records), we also performed port mirroring from $VLAN_{in}$ and $VLAN_{out}$ and then captured pcap files using Wireshark.

With regard to the cost of the lab's setup described above, it is important to note that the total cost of the setup was only in the order of several thousands of dollars, including all of the hardware and software components described above. More importantly, the total cost of the setup is considerably lower than the consequences of a DDoS attack. Such an attack is likely to cause Internet service downtimes, which might translate into customer churn as well as long-term damage to the telco's reputation and its ability to compete with other telcos in terms of the increasingly important QoE (Quality of Experience) measure (Ahmad et al., 2017), i.e., the quality of the telco's service, as perceived by the users. Additional (short-term) costs for the telco, which are associated with DDoS

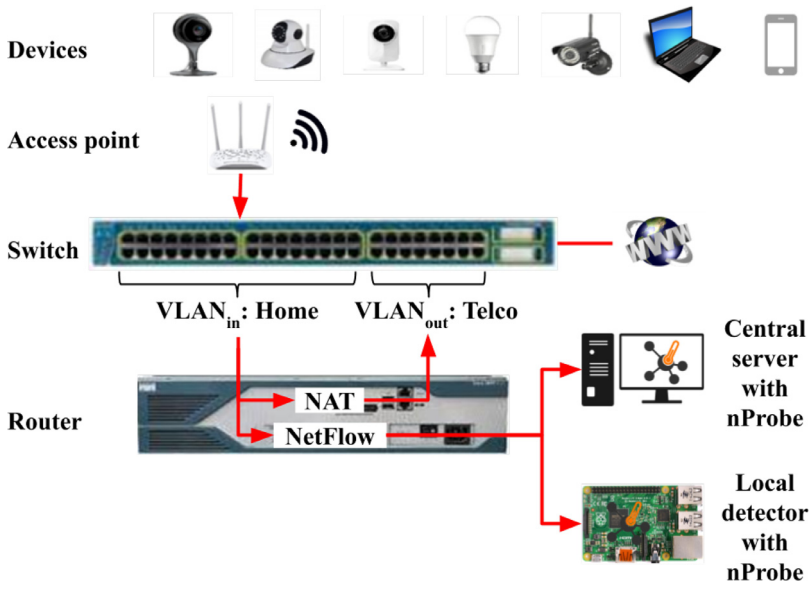


**Fig. 3.** Our evaluation setup, simulating a customer's smart home (denoted as *in*) and the telco's viewpoint (denoted as *out*).

attacks, are discussed in, e.g., Ismail (2018) and Patton (2018), including the time and money spent on mitigating the attack, costs of scrubbing huge amounts of malicious traffic, and legal liability.

### 5.2. Data acquisition

#### 5.2.1. Data collection

For empirical evaluation of our method, we purchased a variety of commercial IoT devices and deployed them in our lab together with non-IoT devices (see Table 3). The IoT devices we deployed represent seven manufacturers (including Amazon, Samsung, and WEMO) as well as three of the most popular types of home IoTs in many regions of the world (Kumar et al., 2019), namely 'Media/TV' (streamer, speaker), 'Surveillance' (webcam) and 'Home Automation' (socket, doorbell, light bulb).

In terms of the IoT devices, we note the distinction made by Li et al. (2019) and Guo and Heidemann (2018) between three common operational phases:

1) **Initial**: Device setup immediately after booting
2) **Active**: User interaction (e.g., webcam viewing)
3) **Idle**: Network connectivity without user-driven traffic

Similar to Li et al. (2019), we chose to focus on the *idle* phase, in which no human interaction is present, however the IoT devices could be very active, e.g., streaming video or sending measurements. The reason to focus on the idle phase is that IoT devices are in this state most of the time (i.e., high coverage). Moreover, during this phase they also tend to send regular messages to their servers, so their traffic patterns are expected to be relatively stable and characteristic, and thus promote machine learning-based classification. Note that a benefit of the relative stability of the network traffic of IoT devices in their idle state (i.e., when they are not interacted with) is that for each IoT model, one centrally-trained classifier should suffice. That is, when idle, IoT devices of the same model are likely to demonstrate very similar (if not nearly identical) traffic behavior, in terms of the features we use, no matter where the devices are deployed. Hence, there is no need to train a classifier for each home separately.

Altogether, we collected NetFlow data for a period of 37 days, during which the non-IoT devices were rather active (i.e., interacted with), and the IoT devices were mostly idle. We presume that this is largely representative of smart homes, where laptops and smartphones are extensively used while smart devices such as sockets or webcams are connected to the network however rarely interacted with.

#### 5.2.2. Feature extraction

We configured nProbe to output a *.flows file once an hour while recording a feature set that is minimal yet potentially informative for IoT model detection. Table 4 outlines these features, explains their content, and provides references to past deNAT and/or IoT identification related studies which employed these (or equivalent) features. We used most of these features as is (indicated by check marks) and added a couple of derived features to each flow:

1) DURATION: Time between first/last packet switching
2) IP: Instead of using the entire destination IPv4 address as an informative feature for classification, we utilized only its prefix which states the network number (as in Liu et al., 2020 and Smith, 2020), without the local address of the specific destination host (i.e., the suffix, also called the "rest" field). We did this to reduce sparsity and avoid overfitting.

#### 5.2.3. Ground truth labeling

In our lab we recorded flows both behind and in front of the NAT. This way we were able to match each external outbound flow (labeled with the public source IP address of the router) with its internal twin (labeled with the private source IP address of the device), based on all of the features except the following:

- IPV4_SRC_ADDR (explicitly changed by the NAT)
- L4_SRC_PORT (might be changed by the NAT)
- FIRST_SWITCHED and LAST_SWITCHED (there might be some minor delays)

We also made sure to configure static IPs on the router, so after finding the private source IP address of each flow, we labeled it with the correct source MAC address and IoT model. Since the firmware of the IoT devices did not change during the data collection period, the IoT model on the label includes only the first three components: type, manufacturer, and model number. We have shared this labeled dataset publicly as part of this research (Meidan et al., 2020).

**Table 3**
The devices used in this research, the amount of captured flows, and the flows' interarrival time and duration.

| Device identifier | | Device | Model | | | | Outbound | Interarrival time (Sec.) | | Flow duration (Sec.) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MAC address | IP address | category | Ref. | Type | Manufacturer | Model number | flows | Mean | St. Dev. | Mean | St. Dev. |
| 90:70:65:03:55:E9 | 192.168.1.201, 2, 3 | IoT | Ring (0000) | doorbell | Amazon | Ring | 17,179 | 184.45 | 31.52 | 4,217.01 | 9.36 |
| 50:C7:BF:A1:02:67 | 192.168.1.113 | IoT | TP-LINK (a) | light_bulb | TP_Link | LB130 | 102,767 | 31.1 | 27.28 | 4,223.54 | 9.24 |
| C0:56:27:B4:84:0F | 192.168.1.212 | IoT | WEMO (0000) | socket | Wemo | Insight | 172,796 | 18.5 | 21.64 | 4,227.05 | 9.28 |
| C0:56:27:B4:B3:45 | 192.168.1.211 | IoT | WEMO (0000) | socket | Wemo | Insight | 166,804 | 19.16 | 22.66 | 4,227.05 | 9.28 |
| 78:28:CA:AA:70:52 | 192.168.1.195 | IoT | SONOS (0000) | speaker | Sonos | One | 78,480 | 40.73 | 20.91 | 4,226.78 | 9.29 |
| F0:81:73:C9:BD:56 | 192.168.1.208 | IoT | Amazon (2019) | streamer | Amazon | Fire_TV_Stick | 162,287 | 19.69 | 12.08 | 4,225.92 | 9.84 |
| 9C:8E:CD:0E:B5:24 | 192.168.1.225 | IoT | Amcrest (0000) | webcam | Amcrest | IPM-721W | 479,492 | 6.67 | 4.06 | 4,218.54 | 15.89 |
| 00:16:6C:7F:82:21 | 192.168.1.227 | IoT | CNET (0000) | webcam | Samsung | SNH-1011N | 217,899 | 14.67 | 9.37 | 4,220.62 | 10.63 |
| 98:DE:D0:D0:B0:F0 | 192.168.1.217 | Non-IoT | TP-LINK (b) | access_point | TP_Link | TL-WA901ND | 13,213 | 241.88 | 898.09 | 4,226.54 | 9.26 |
| 4C:1D:96:A9:AE:66 | 192.168.1.210 | Non-IoT | DELL (0000) | laptop | Dell | Latitude_7400 | 4864 | 573.56 | 9515.2 | 4,227.86 | 14.79 |
| 60:67:20:7A:E7:D8 | 192.168.1.215 | Non-IoT | del (0000) | laptop | Dell | Latitude_E6430 | 23,254 | 55.94 | 4051.06 | 4,211.06 | 10.42 |
| AC:5F:3E:48:15:C0 | 192.168.1.238 | Non-IoT | Samsung (0000) | smartphone | Samsung | Galaxy_Note_5 | 15,001 | 184.09 | 5771.09 | 4,225.97 | 10.9 |
| E8:50:8B:1B:A3:89 | 192.168.1.231 | Non-IoT | Samsung (0000) | smartphone | Samsung | Galaxy_Note_4 | 111,250 | 28.72 | 89.75 | 4,221.01 | 8.95 |

#### 5.2.4. Data preprocessing

After gathering all the hourly *.flows files, adding the above-mentioned two derived features, and labeling each flow with the model of the device that produced the traffic, we performed only a few additional steps of data preprocessing, as follows:

- Keeping only the external outbound flows, i.e., the ones which originated from the devices in our lab and were sent to external servers (outside our network).
- Removing all of the zero variance features (e.g., SRC_AS, DST_AS, INPUT_SNMP, OUTPUT_SNMP, see the related feature descriptions in cisco.com, 2011), since they would have no contribution to classification.
- Keeping only the features that could be relevant for IoT model detection/classification, as mentioned in Table 4. For instance, we removed the source IP addresses of the external outbound flows, because due to the NAT functionality they were all the same, i.e., the router's public address. The remaining feature set was rather small (only nine features) and uncorrelated, so we did not perform additional feature selection.
- Encoding the categorical features (PROTOCOL, SRC_TOS, TCP_FLAGS and IP) and scaling the numerical features, such that all of the input features would be in the range of [0, 1].

### 5.3. Experimentation

Table 3 contains thirteen rows, one for each device that we connected to our controlled network, and from which we captured NetFlows. Of the 13 physical devices, eight are IoTs, representing seven unique IoT models (we used two identical devices which represent the same IoT model, namely socket.Wemo.Insight). A large portion of the section that follows describes how we trained, tuned, and evaluated the classifiers for those seven IoT models, while the five remaining non-IoT devices were used as background noise that is typical to home networks.

#### 5.3.1. Data partitioning

In our experiments, we utilized the first 30 days (approximately 81% of each device, on average) for classifier training, and the remaining 7 days (approximately 19%) to test the classification performance. Instead of using the popular random (sometimes stratified) sampling, we opted for chronological partitioning, because it better represents our use case: An IoT model classifier is trained in the central lab based on the accumulated traffic flows and only then applied to ongoing flows. To facilitate reproducibility and comparison of results in future research, for each flow in our benchmark dataset, we indicate whether it belongs to the training or the test set.

#### 5.3.2. Performance metrics

We deem the following performance metrics as essential for algorithm selection:

- **Training time**: The number of seconds it takes to train the classifier (using the Google Colab platform). Obviously, the quicker it takes to train a classifier the better it is for the telco, e.g., in terms of scalability.
- **Classifier size**: The disk size (KB) required by the IoT model classifier. Depending on the resources of the local detector, space-consuming classifiers might limit the number of IoT models which can be detected.
- **Flow classification time**: The number of seconds it takes to classify a single flow in the test set. This affects the system's latency and its ability to apply several classifiers on a flow, each classifier corresponding to a different IoT model to detect.

**Table 4**
The features used in this research (ordered alphabetically), plus explanations, notes, example values, and prior deNAT and IoT identification related studies which used them.

| Features | | | Example references | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | Explanation (cisco.com, 2011) | Notes and examples values | Guo and Heidemann (2018) | Li et al. (2019) | Meidan et al. (2017) | Apthorpe et al. (2017) | Patton et al. (2014) | van der Elzen and van Heugten (2017) | Nobakht et al. (2016) |
| FIRST_SWITCHED | System uptime at which the first packet of this flow was switched | Used here to calculate flow duration | | √ | √ | | | | √ |
| IN_BYTES | Incoming counter for the number of bytes associated with an IP Flow | Length N x 8 bits | √ | √ | √ | √ | | | √ |
| IN_PKTS | Incoming counter for the number of packets associated with an IP Flow | Length N x 8 bits | | | √ | | | | √ |
| IPV4_DST_ADDR | IPv4 destination address | To reduce sparsity we remove the host portion and use the network portion only | √ | | √ | √ | √ | √ | √ |
| L4_DST_PORT | TCP/UDP destination port number | Indicative (to a certain extent) as to the communication protocol being used | | | √ | | √ | √ | |
| L4_SRC_PORT | TCP/UDP source port number | | | | | | | √ | |
| LAST_SWITCHED | System uptime at which the last packet of this flow was switched | Used here to calculate flow duration | | √ | √ | | | | √ |
| PROTOCOL | IP protocol byte | 6: TCP, 17: UDP | | | √ | √ | √ | √ | √ |
| SRC_TOS | Type of Service byte setting when there is an incoming interface | 0: unused, 1: lowcost, 2: reliability, 3: throughput, 4: low delay, 5–7: IP precedence | | | | | | √ | √ |
| TCP_FLAGS | Cumulative of all the TCP flags seen for this flow | SYN, ACK, FIN, etc. | | | √ | | | | |

- **Area under the precision-recall curve (AUPRC)**: As can be seen in Table 3, there is a substantial class-imbalance (outbound flow counts) in the data we captured, which means that some classes (i.e., IoT models) are represented much more than others in our dataset. This class imbalance pertains to both the training and test sets (see Table 5), which we use to evaluate each IoT model classifier with a binary (one-vs-rest) approach. Thus, as suggested in Davis and Goadrich (2006), we use the AUPRC as a key metric for summarizing the classification performance of each classifier.

Section 6 uses the abovementioned performance metrics to quantitatively evaluate our proposed method for each classifier separately. To summarize those metrics over a multitude of classifiers, we use the form of (*Mean* $\pm$ *St.Dev.*), which are calculated as unweighted means and standard deviations, respectively, to reflect equal contribution.

## 6. Experimental results

### 6.1. Selection of the classification algorithm

In our first experiment (summarized in Table 5) we evaluated three supervised learning algorithms, which are considered state-of-the-art (Rai and Mandoria, 2019). Below, we provide an introduction to these algorithms as well as the values we set for their key hyper-parameters (based on preliminary experimentation and tuning).

1) **Light Gradient Boosting Machine** (LGBM): This is a highly-efficient (Ke et al., 2017) gradient boosting framework which uses tree-based learning algorithms. Among its benefits are its ability to handle large amounts of data, and attain accuracy levels superior to any other boosting algorithm with low memory requirements. We evaluated this algorithm using Microsoft's implementation (Microsoft, 0000). To lower the chances of overfitting, we limited the number of leaves in a tree (*num_leaves* hyperparameter) to 32. To enable an effective boosting process with short training times, we set the number of boosting iterations (*max_iterations*) to 30.
2) **Deep Neural Network** (DNN): Over the past decade, this neurally-inspired machine learning algorithm has been shown to be powerful and effective in tackling various real-world tasks (Cichy and Kaiser, 2019). We evaluated it using the Keras library (Chollet, 2015), in which we defined the following architecture: Two hidden layers (six neurons in each) that use the *ReLU* activation function, each followed by a dropout layer with a rate of 0.1 to prevent overfitting. The output layer had one neuron which used the *Sigmoid* activation function. We defined the *binary cross-entropy* loss and compiled the model using the *RMSprop* optimizer.
3) **Support Vector Machine** (SVM): Kernel SVMs (Cortes and Vapnik, 1995) generally have $O(N^3)$ complexity, where $N$ is the total number of training instances. In our case, to reduce the computation and expedite the training process, we used a much faster implementation called LinearSVC instead, which is provided by scikit-learn (Pedregosa et al., 2011). This implementation aims at solving the optimization problem with a complexity of approximately $O(N)$.

Table 5 presents our experimental results in terms of the performance metrics defined in Section 5.3.2. We obtained these results by training the abovementioned classification algorithms using all of the training set (all of the 30 days), and testing using all of the test set (the remaining seven days). A summary of those results, combined with insights and conclusions we draw from them, is presented below.

- **Training time**: LGBM required the shortest amount of time to train a classifier (2.19 $\pm$ 1.68), with most devices requiring less than 1.5 seconds. SVM took slightly longer to train (8.34 $\pm$ 6.01), and DNN was considerably slower than both LGBM and SVM (135.1 $\pm$ 48.58). For the telco, the (very) short amount of training time required by LGBM allows thorough experimentation and/or classifier re-training to fine-tune multiple hyperparameters of the algorithm.
  A possible explanation for the difference in the training time (as well as the classifier size, classification time, and AUPRC) is the number of unique values of the categorical features of each IoT model. Particularly, as can be seen in Table 6, different IoT models communicate with different numbers of unique IP addresses, using different numbers of unique source and destination ports, and so on. In turn, as part of the preprocessing step, the categorical features are one-hot encoded and result in sparse matrices. These matrices are then used for training the classifiers, which become more complex. For instance, the input layer of a DNN IoT model classifier becomes larger, and requires many more parameters (i.e., weights) to train. Thus, the training phase takes more time, and also the required disk space becomes larger. In our experiments, streamer.Amazon.Fire_TV_Stick and speaker.Sonos.One had the second and third largest amount of total unique categorical values, and considering the four performance metrics, they performed not as good as the other IoT models for most algorithms.
- **Classifier size**: Both DNN and SVM resulted in classifiers that required the same disk size, no matter what the IoT model is. In our experiments, the SVM classifiers were the smallest (0.86 KB), as each of them is mostly comprised of the optimized feature weights (a coefficient array of size [$1 \times$ *numberoffeatures*]). The DNN classifiers were slightly larger (6.55 KB for each classifier), as each of them contains more information than the SVM classifier, e.g., the DNN's architecture, its optimized weights, the training configuration (loss, optimizer, etc.), and the state of the optimizer. In comparison, LGBM resulted in classifiers of various sizes (42.54 $\pm$ 55.23), which are larger than the DNN and SVM classifiers. Another cause for the classifier size differences is that the weights of DNN are stored in HDF5 (The HDF Group, 2000-2010) format which is highly organised and can save storage space. In comparison, the parameters and weights of LGBM are stored in the less sophisticated text format, which suffers from redundancy, and also includes the overhead of converting numeric data to ASCII text.
- **Flow classification time**: The SVM classifiers required only ($1.7E-08 \pm 6.5E-10$) seconds to classify a single flow in the test set, on average. The LGBM classifiers were moderately slower ($5.9E-07 \pm 5.5E-07$), and the DNN classifiers were the slowest ($2.8E-05 \pm 5.5E-07$). Still, relative to the interarrival time of the flows (Table 3), as well as the flow durations, it is clear that both the classification times and the related differences between the evaluated algorithms are negligible.
- **Area under the precision-recall curve (AUPRC)**: To compute the per IoT model AUPRC, we used the *average_precision_score()* function in scikit-learn (Pedregosa et al., 2011). The results in Table 5 demonstrate how LGBM (0.99 $\pm$ 0.017) outperforms DNN (0.967 $\pm$ 0.044) and SVM (0.89 $\pm$ 0.139) in terms of AUPRC. Although the differences might seem to be minor, in our use case they could have major effects in practice. That is, on the one hand, a telco wishing to detect vulnerable IoT devices connected to its customers' home networks would aim not to miss too many vulnerable IoT models (i.e., maximize the recall), so as to minimize botnet related risks. On the other hand, the telco would also struggle not to take actions against false detections (i.e., maximize the precision), since they could

**Table 5**
Comparison of the classification algorithms evaluated in this research.

| IoT | Outbound flows | | Training time (Sec.) | | | Classifier size (KB) | | | Flow classification time (Sec.) | | | AUPRC using the test set | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| model | Training set | Test set | LGBM | DNN | SVM | LGBM | DNN | SVM | LGBM | DNN | SVM | LGBM | DNN | SVM |
| webcam. Amcrest. IPM-721W | 388,783 | 90,709 | **1.23** | 103.41 | 20.54 | 9.47 | 6.55 | **0.86** | 2.7E-07 | 2.9E-05 | **1.7E-08** | **1** | **1** | 0.884 |
| socket. Wemo. Insight | 276,733 | 62,867 | **1.47** | 199.79 | 6.01 | 17.24 | 6.55 | **0.86** | 3.4E-07 | 2.9E-05 | **1.7E-08** | **1** | 0.999 | 0.979 |
| webcam. Samsung. SNH-1011N | 171,845 | 46,054 | **1.37** | 99.74 | 4.87 | 15.13 | 6.55 | **0.86** | 3.0E-07 | 2.8E-05 | **1.6E-08** | **0.999** | 0.985 | 0.97 |
| streamer. Amazon. Fire_TV _Stick | 131,668 | 30,619 | **5.23** | 201.63 | 8.9 | 148.37 | 6.55 | **0.86** | 1.6E-06 | 2.9E-05 | **1.8E-08** | **0.992** | 0.961 | 0.934 |
| light_bulb. TP_Link. LB130 | 83,247 | 19,520 | **1.04** | 80.71 | 2.54 | 7.69 | 6.55 | **0.86** | 2.5E-07 | 2.8E-05 | **1.7E-08** | **1** | **1** | 1 |
| speaker. Sonos. One | 63,648 | 14,832 | **3.94** | 118.22 | 10.59 | 90.54 | 6.55 | **0.86** | 1.1E-06 | 2.8E-05 | **1.7E-08** | **0.952** | 0.879 | 0.594 |
| doorbell. Amazon. Ring | 14,023 | 3156 | **1.07** | 142.17 | 4.94 | 9.36 | 6.55 | **0.86** | 2.5E-07 | 2.8E-05 | **1.6E-08** | **0.989** | 0.945 | 0.867 |
| | **Unweighted column-wise Mean** | | **2.19** | 135.10 | 8.34 | 42.54 | 6.55 | **0.86** | 5.9E-07 | 2.8E-05 | **1.7E-08** | **0.99** | 0.967 | 0.890 |
| | **Unweighted column-wise St.Dev.** | | **1.68** | 48.58 | 6.01 | 55.23 | 0.00 | **0.00** | 5.5E-07 | 5.5E-07 | **6.5E-10** | **0.017** | 0.044 | 0.139 |

**Table 6**
Number of unique values of the categorical features of the training set.

| IoT model | Feature | | | | | | | | | | Total values |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | L4_DST _PORT | L4_SRC _PORT | SRC_TOS | SRC_AS | DST_AS | INPUT _SNMP | OUTPUT _SNMP | TCP _FLAGS | PROTOCOL | IPV4 _DST _ADDR | |
| webcam. Samsung. SNH-1011N | 2 | 4031 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1439 | 5481 |
| light_bulb. TP_Link. LB130 | 3 | 16,593 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 9 | 16,614 |
| speaker. Sonos. One | 3 | 21,952 | 2 | 1 | 1 | 1 | 1 | 9 | 2 | 49 | 22,021 |
| doorbell. Amazon. Ring | 3 | 9475 | 1 | 1 | 1 | 1 | 1 | 4 | 2 | 112 | 9601 |
| webcam. Amcrest. IPM-721W | 4 | 28,539 | 2 | 1 | 1 | 1 | 1 | 7 | 2 | 6 | 28,564 |
| socket. Wemo. Insight | 4 | 2044 | 2 | 1 | 1 | 1 | 1 | 6 | 3 | 15 | 2078 |
| streamer. Amazon. Fire_TV _Stick | 5 | 14,781 | 3 | 1 | 1 | 1 | 1 | 15 | 3 | 3856 | 18,667 |

**Table 7**
Feature importance for the LGBM classifiers.

| IoT model | Feature | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | DURATION | L4_SRC _PORT | L4_DST_PORT | IP | IN_BYTES | PROTOCOL | IN_PKTS | TCP_FLAGS | SRC_TOS |
| webcam. Samsung. SNH-1011N | **36** | **34** | 23 | 11 | 9 | 5 | 4 | 2 | 0 |
| light_bulb. TP_Link. LB130 | **8** | **14** | 5 | 0 | 5 | 4 | 3 | 0 | 0 |
| speaker. Sonos. One | 92 | **151** | 23 | 71 | **456** | 24 | 118 | 67 | 21 |
| doorbell. Amazon. Ring | **18** | 11 | 6 | 0 | 13 | 2 | **12** | 0 | 0 |
| webcam. Amcrest. IPM-721W | 7 | 10 | 6 | 2 | **13** | 4 | **16** | 2 | 2 |
| socket. Wemo. Insight | **40** | **77** | 9 | 5 | 5 | 2 | 7 | 5 | 5 |
| streamer. Amazon. Fire_TV_Stick | 46 | 131 | 125 | **266** | **562** | 4 | 195 | 194 | 27 |
| **Column-wise sum** | 247 | 428 | 197 | 355 | 1063 | 45 | 355 | 270 | 55 |

harm the quality of service provided to the related Internet customer. As LGBM demonstrates the largest AUPRC, it provides the best trade-off between precision and recall, thus is likely to be preferred by a telco.

Altogether, the SVM classifiers were found to be superior to the others in terms of the classifier size and flow classification time. However, since the inferior algorithms (DNN and LGBM) led to results that are feasible and acceptable for our use case, we based our algorithm selection decision using the other two performance metrics (training time and AUPRC). Actually, for our use case those metrics have a greater effect, as a shorter training time facilitates the scaling of our IoT model detection method as well as fine-tuning the classifiers, and a higher AUPRC indicates a classifier that is more able to differentiate between flows emanating from behind a NAT (i.e., to tell if they were generated by a certain IoT model or not).

Regarding the classifier size, note that assuming reasonable free storage space of 1 GB on a local detector, the number of LGBM classifiers (which attained the highest AUPRC) that can possibly be saved is approximately $\frac{1,000,000KB}{42.54KB} \approx 23,507$. Hence, based on the assumption that the list of vulnerable IoT models to detect would be much smaller (see Section 7.2.1), the classifier size does not pose a constraint. Moreover, in this research we consider the classifiers' ability to correctly determine whether a flow was generated by a certain IoT model or not (measured here in terms of AUPRC) as a key concern to the telco (see Section 4.1). Consequently, we

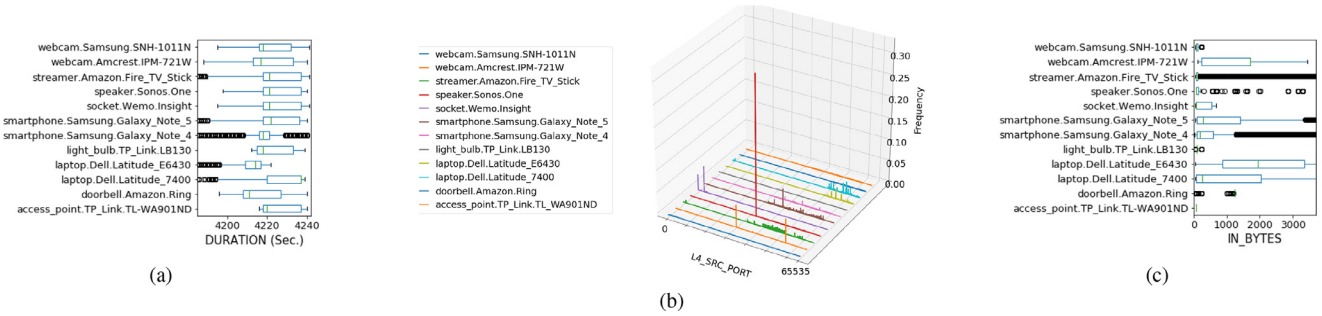


**Fig. 4.** The distribution of features found important by LGBM: (a) DURATION, (b) L4_SRC_PORT, and (c) IN_BYTES.

decided to continue only with the LGBM IoT model classifiers in the rest of our experimentation.

### 6.2. Feature importance

The information presented in Table 7 shows the importance of the features in LGBM classifiers, calculated as the number of times a feature is used in a classifier (Microsoft, 0000). In this table we highlighted (bold entries) the two most important features for each IoT model. Based on this ranking, the duration of a flow (in seconds) and its source port are quite consistently the most important features (i.e., characteristic and informative for detection), followed by the number of incoming bytes. As can be seen in Fig. 4, those features, which were found the most important by LGBM, indeed distribute differently among the examined IoT and non-IoT device models: Fig. 4a and Fig. 4c demonstrate different medians, interquartile ranges, and number of outliers for the features DURATION and IN_BYTES, respectively. Fig. 4b demonstrates that, as one would have expected, for some device models the distribution of L4_SRC_PORT is uniform, however for others the distribution is not entirely so. That is, the source ports of, e.g., speaker.Sonos.One and socket.Wemo.Insight, weren't chosen entirely randomly, but rather certain source ports were used much more frequently than others, such that they became informative for classification. The frequency at which each device model uses each port is expressed by the height of the related bar on the histogram. As easily observed, indicative peaks do exist for the examined device models, and thus L4_SRC_PORT was found to be an important feature by LGBM. The remaining features were important for some IoT models and unimportant for others.

### 6.3. Hyperparameter tuning

Since the LGBM IoT model classifiers outperformed the DNN and SVM classifiers (see Table 5), we further calibrated LGBM by fine-tuning the values of *num_leaves* and *max_iterations*. For that, we performed a grid search using combinations of these two hyperparameters over the following ranges:

- $num\_leaves \in [2^3, 2^4, \ldots, 2^8]$
- $max\_iterations \in [5, 10, 20, 30, 50, 60, 70, 80, 100, 200]$

As can be seen in Fig. 5a, in the lower range of *num_leaves*, the AUPRC peaks at approximately $2^7 = 128$ leaves. Note that although increasing values of *num_leaves* can (slightly) improve the AUPRC, they might also lead to overfitting. Also note that beyond the AUPRC peak, where approximately $max\_iterations = 60$, increasing values of *max_iterations* leads to excess training (Fig. 5b) and classification (Fig. 5c) times, without improving the AUPRC.

Table 8 presents the results of the same performance metrics presented in Table 5, this time only for the LGBM with the fine-tuned hyperparameters, i.e., $num\_leaves = 128$ and $max\_iterations = 60$. As evident in Table 8, the AUPRC improved ($0.992 \pm 0.014$), compared with the basic LGBM implementation ($0.99 \pm 0.017$), which was obtained using $num\_leaves = 32$ and $max\_iterations = 30$. As is also evident in Table 8, the training time, classifier size, and flow classification time metrics deteriorated, however their values are still acceptable and their impact is of lesser importance to a telco than the classifiers' ability to accurately detect NATed IoT models.

The experimental results that are presented in the following subsections are based on LGBM IoT model classifiers that were trained using these fine-tuned hyperparameter values.

### 6.4. Optimization of the classification threshold

AUPRC provides an indication of an algorithm's performance for varying classification thresholds. However, eventually a telco needs to decide only one threshold for a classifier, and use this threshold as part of the local deployment phase. To decide this threshold we suggest to take the NZFP approach (described in Section 4), since reactions to false detections are likely to be followed by customer dissatisfaction and cause harm to the telco's reputation. In our experiments, to implement the NZFP approach for each IoT model we

1) Trained a LGBM classifier using the training set
2) Applied the classifier to the validation set
3. Used scikit-learn's *roc_curve()* function to produce TPR (true positive rate) and FPR (false positive rate) values for various classification thresholds
3) Determined a near-zero false positives target value (e.g., $NZFP = 0.0001$)
4) Set the threshold as the lowest classification threshold (produced by *roc_curve()*) which led to an FPR equal to or lower than the target NZFP
4) Applied the classifier to the test set, and for each flow, if its score was below the threshold we classified it as false, or true otherwise
5) Calculated TPR and FPR using the test set

Table 9 summarizes the threshold optimization experiments with four near-zero target values: $NZFP \in [0.01, 0.001, 0.0001, 0.00001]$. This table shows that $NZFP = 0.01$ results in very high TPR ($0.998 \pm 0.004$), however with FPR that is not practical in 4/7 of the IoT models. Better results were achieved with $NZFP = 0.001$, however it seems that if only one (global) *NZFP* is to be set, then 0.00001 is the best choice among the tested values. It provides virtually zero FPR and only a small negative effect on TPR, except for speaker.Sonos.One (whose target *NZFP* can be fine-tuned and set between 0.001 and 0.0001)

### 6.5. Sensitivity analysis

The experimental results presented in Tables 5 and 8 pertain to IoT model classifiers that were trained using the complete training

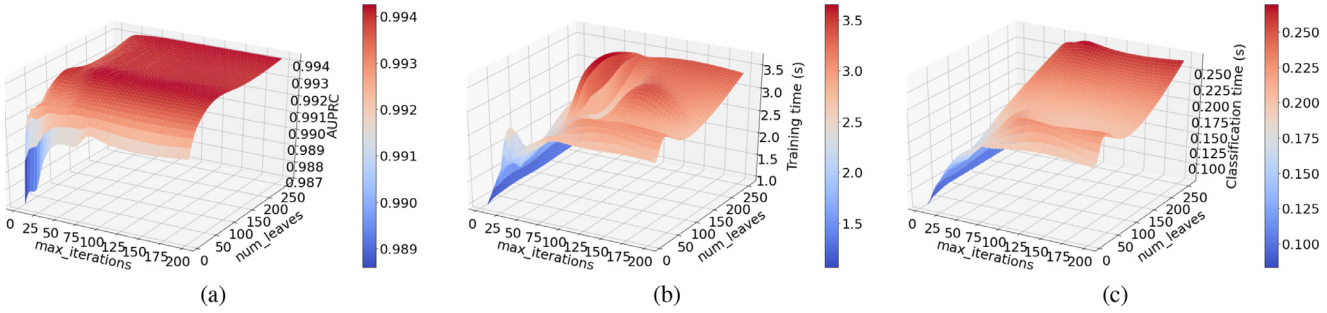


**Fig. 5.** The effect of tuning LGBM's *max_iterations* and *num_leaves* hyperparameters on (a) the average AUPRC, (b) the training time, and (c) the classification time.

**Table 8**
Performance of LGBM IoT model classifiers following hyperparameter tuning.

| IoT Model | Training time (Sec.) | Classifier size (KB) | Flow classification time (Sec.) | AUPRC using the test set |
|---|---|---|---|---|
| webcam.Samsung.SNH-1011N | 3.098 | 192.790 | 5.89E-07 | 1.000 |
| light_bulb.TP_Link.LB130 | 1.071 | 7.710 | 2.18E-07 | 1.000 |
| speaker.Sonos.One | 1.419 | 33.240 | 2.75E-07 | 0.961 |
| doorbell.Amazon.Ring | 1.985 | 121.300 | 4.26E-07 | 0.989 |
| webcam.Amcrest.IPM-721W | 1.296 | 12.170 | 9.23E-07 | 1.000 |
| socket.Wemo.Insight | 1.746 | 58.370 | 3.59E-07 | 1.000 |
| streamer.Amazon.Fire_TV_Stick | 6.059 | 471.550 | 1.67E-06 | 0.994 |
| **Unweighted column-wise Mean** | 2.382 | 128.161 | 6.37E-07 | 0.992 |
| **Unweighted column-wise St.Dev.** | 1.752 | 165.319 | 5.13E-07 | 0.014 |

**Table 9**
TPR and FPR obtained by LGBM IoT model classifiers applied to the test set using various NZFP target values on the validation set.

| IoT | NZFP=0.01 | | NZFP=0.001 | | NZFP=0.0001 | | NZFP=0.00001 | |
|---|---|---|---|---|---|---|---|---|
| model | TPR | FPR | TPR | FPR | TPR | FPR | TPR | FPR |
| webcam.Samsung.SNH-1011N | 1.000 | 0.085 | 0.999 | 0.011 | 0.999 | 0.011 | 0.992 | 0.000 |
| light_bulb.TP_Link.LB130 | 1.000 | 0.000 | 1.000 | 0.000 | 1.000 | 0.000 | 1.000 | 0.000 |
| speaker.Sonos.One | 1.000 | 0.046 | 0.998 | 0.006 | 0.498 | 0.001 | 0.370 | 0.000 |
| doorbell.Amazon.Ring | 0.989 | 0.006 | 0.989 | 0.006 | 0.988 | 0.000 | 0.987 | 0.000 |
| webcam.Amcrest.IPM-721W | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 |
| socket.Wemo.Insight | 1.000 | 0.000 | 1.000 | 0.000 | 1.000 | 0.000 | 1.000 | 0.000 |
| streamer.Amazon.Fire_TV_Stick | 1.000 | 0.079 | 0.975 | 0.005 | 0.971 | 0.001 | 0.944 | 0.000 |
| **Unweighted column-wise Mean** | 0.998 | 0.174 | 0.995 | 0.147 | 0.922 | 0.145 | 0.899 | 0.000 |
| **Unweighted column-wise St.Dev.** | 0.004 | 0.366 | 0.010 | 0.376 | 0.188 | 0.377 | 0.234 | 0.000 |

set, i.e., the flow records that were (1) captured during the entire period of 30 days, and (2) generated by all of the IoT models. Below, we provide an evaluation of the selected algorithm (LGBM) in terms of AUPRC when applied to the same test set but trained using datasets which consist of flow records that were (1) captured during a smaller number of consecutive days, or (2) generated using one less IoT model, respectively.

*6.5.1. Size of the training set*

In a prior deNAT related paper (Khatouni et al., 2019), the authors noted that the size of the training set strongly affects the performance of the classifier. To evaluate the sensitivity of the LGBM classifiers to the size of the training set, we conducted several experiments. In each experiment we (1) decided on a number of capturing days $n$ to evaluate, (2) assigned the most recent $n$ consecutive days to the training set, (3) trained the classifiers, and (4) always applied the classifiers to the same test set to calculate the AUPRC.

Table 10 summarizes these experiments. It shows that for light_bulb.TP_Link.LB130 and socket.Wemo.Insight, a single day of data capturing is enough to reach $AUPRC = 1$, while doorbell.Amazon.Ring required six days to reach its maximal AUPRC. The best AUPRC values ($0.993 \pm 0.013$) were attained using nine consecutive days for training, which is 30% of the total size of the training set we captured and reported thus far (30 days altogether).

*6.5.2. Variety of device models in the training set*

Especially during the early stages of the central lab's setup, it is likely that the training data would be captured from a limited variety of device models, i.e., the ones purchased and deployed in the lab. This lack of variety is associated with the vulnerable IoT models on the detection list, as well as popular "background" IoT (off the detection list) and non-IoT devices. This would result in IoT model classifiers that are trained using a certain limited variety of device models, and then tested on another set (i.e., the device models connected to the home networks of the telco's customers). To analyze the sensitivity of LGBM classifiers to this situation, in each iteration of another experiment performed, we (1) removed one IoT model from the training set, (2) trained IoT model classifiers for all of the *remaining* IoT models, and (3) calculated the AUPRC using the same test set as before to enable a fair comparison of the classification performance. Table 11 summarizes this experiment and demonstrates that the expected performance deterioration is minor. Specifically, the AUPRC only declined slightly from 0.992 to 0.961, on average. The only IoT model which led to an average AUPRC below 0.9 is streamer.Amazon.Fire_TV_Stick. A

**Table 10**
AUPRC values obtained by LGBM classifiers using the test set and increasing number of traffic capturing days in the training set.

| IoT | Number of traffic capturing days in the training set | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| model | 1 | 2 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 (all) |
| webcam. Samsung. SNH-1011N | 0.994 | 0.995 | 0.994 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | **1.000** | 1.000 | 1.000 |
| light_bulb. TP_Link. LB130 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| speaker. Sonos. One | 0.953 | 0.951 | 0.959 | 0.959 | 0.964 | 0.963 | 0.963 | 0.963 | **0.965** | 0.963 | 0.963 | 0.961 |
| doorbell. Amazon. Ring | 0.985 | 0.987 | 0.987 | **0.993** | 0.992 | 0.986 | 0.990 | 0.990 | 0.989 | 0.984 | 0.984 | 0.989 |
| webcam. Amcrest. IPM-721W | 0.998 | 0.998 | 0.998 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| socket. Wemo. Insight | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| streamer. Amazon. Fire_TV_Stick | 0.970 | 0.994 | 0.994 | 0.990 | **0.995** | 0.995 | 0.994 | 0.995 | 0.995 | 0.995 | 0.995 | 0.994 |
| **Unweighted column-wise Mean** | 0.985 | 0.989 | 0.990 | 0.991 | **0.993** | 0.992 | 0.992 | 0.993 | 0.993 | 0.992 | 0.992 | 0.992 |
| **Unweighted column-wise St. Dev.** | 0.018 | 0.017 | 0.014 | 0.015 | **0.013** | 0.014 | 0.013 | 0.013 | 0.013 | 0.014 | 0.014 | 0.014 |

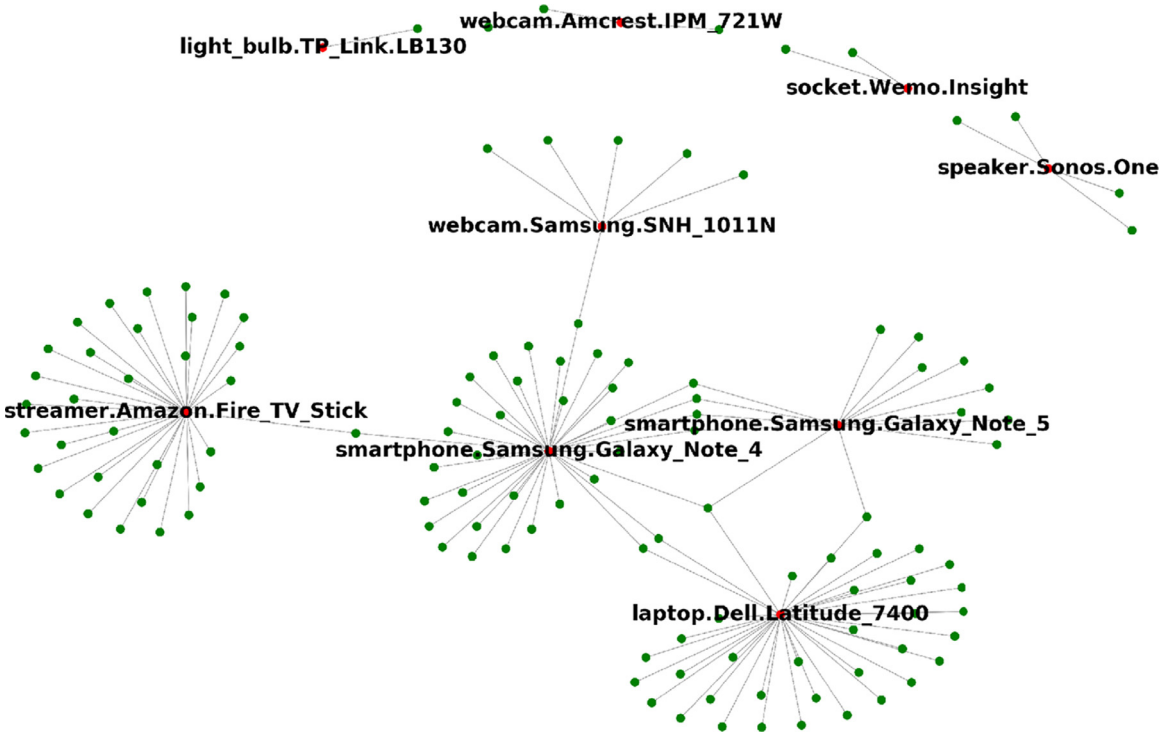


**Fig. 6.** Clusters of domain names (green nodes) requested by the devices (red nodes) in our experiment. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

**Table 11**
AUPRC values obtained by LGBM IoT model classifiers using the test set upon removing one IoT model at a time from the training set.

| IoT model removed from the training set | AUPRC using the test set | |
|---|---|---|
| | Mean | St. Dev |
| webcam.Samsung.SNH-1011N | 0.942 | 0.092 |
| light_bulb.TP_Link.LB130 | 0.986 | 0.016 |
| speaker.Sonos.One | 0.997 | 0.004 |
| doorbell.Amazon.Ring | 0.991 | 0.015 |
| webcam.Amcrest.IPM-721W | 0.973 | 0.040 |
| socket.Wemo.Insight | 0.961 | 0.064 |
| streamer.Amazon.Fire_TV_Stick | 0.873 | 0.278 |
| **Unweighted column-wise Mean** | 0.961 | |

possible explanation is that unlike, e.g., light_bulb.TP_link.LB130 or webcam.Amcrest.IPM-721W, streamer.Amazon.Fire_TV_Stick communicates with servers that are shared with other device models (see Section 6.6.1).

### 6.6. Applicability of the primary existing deNATing methods to our use case

Table 2 outlines the methods used in prior deNAT and IoT device identification related studies. In this subsection, we provide a quantitative evaluation of their applicability to our use case. Most of these methods rely on features that are not represented in NetFlow records, so we conducted the applicability evaluation using packet level network traces. These traces were captured from the devices connected to $VLAN_{in}$ (see Fig. 3) by $Wireshark_{in}$ for a period of 24 hours when the IoT devices were in their idle state.

#### 6.6.1. Domain names in DNS requests

All three papers that we found on deNATing in the IoT (i.e., the particular context of our use case) make use of the domain names in DNS requests (Apthorpe et al., 2017; Guo and Heidemann, 2018; Li et al., 2019). These papers rely on the notion that IoT devices, unlike PCs and smartphones, exchange traffic regularly with unique servers, typically run by their manufacturers. At the same time, these papers also admit of considerable drawbacks associated with relying on the requested domains, e.g., substantial overhead in generating and maintaining the device specific lists of domains, overlapping communicated domains, etc. (see Section 3). In addition, there is concern regarding the availability of the requested domain names in the first place due to traffic encryption.

Empirically, based on the traffic we captured in our lab, most of the IoT devices request a rather small and unique set of domain names, with a limited extent of domain name overlapping among different IoTs. This is expressed in Fig. 6 as relatively distinct clusters of domain names and the respective IoT models which requested them. In fact, four IoT models requested completely distinctive domain names during the traffic capturing period. This finding confirms the notion of servers being regularly communicated with by IoT devices, thus supporting the use of this method for IoT model detection. On the other hand, the remaining IoTs demonstrate a slight overlap in requested domain names, and in addition, it can be seen in Fig. 7 that the concern regarding the unavailability of this method's fundamental data source (i.e., the contents of the DNS request) due to encryption is realistic. That is, the encryption issue is pertinent to most of the non-IoTs we evaluated, as well as two IoT devices (streamer.Amazon.Fire_TV_Stick and speaker.Sonos.One). Furthermore, the network traces we collected show that light_bulb.TP_Link.LB130 actually communicates using the DNS protocol only twice in 24 hours, so any DNS-based method would take long or even fail to detect this IoT model. In contrast, our flow-based method does not rely solely

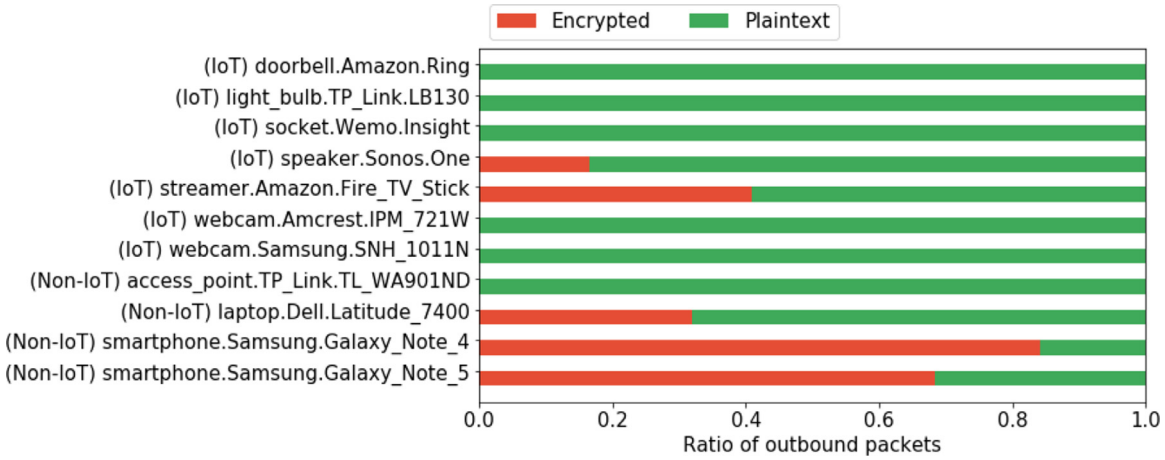


**Fig. 7.** Share of encrypted and plaintext packets among outbound traffic.

on any one protocol, and it detects this IoT model with an AUPRC of 1. Note that during the same data capturing time, both doorbell.Amazon.Ring (IoT) and access_point.TP_Link.TL-WA901ND (non-IoT) did not produce any DNS traffic, so DNS-based methods cannot detect these device models. This is also the reason why they are not shown in Fig. 6.

The use of encryption in IoT traffic is increasing (Li et al., 2019; Valdez et al., 2019), and some IoT models might not use DNS at all; as DNS, and even DNS over HTTPS, can reveal sensitive user information (Bushart and Rossow, 2019), alternative data sources should be considered for IoT deNATing to increase effectiveness and coverage while reducing privacy violations.

*6.6.2. Additional application layer features*

A number of such features have been utilized in the past in deNAT related studies. Of them, MSN transaction ID (Ori et al., 2008) is virtually irrelevant to the IoT, while HTTP user agent (Gokcen et al., 2014; Maier et al., 2011) is not universal and is expected to be used even less frequently due to privacy issues (Li et al., 2019). Furthermore, similar to the above methods which rely on domain names in DNS requests, the main drawback of the application layer features is that in cases where the traffic is encrypted they become unavailable, thus compromising the IoT model's detection coverage.

*6.6.3. IP-ID in DNS requests*

As noted by Orevi et al. (2017), the IP ID field in subsequent packets from a host to a fixed destination IP (e.g., to its DNS resolver) is typically incremented by 1 in some OSs, including Windows 8 and 10 and Android. This behavior allows passive listeners on the other side of the NAT to identify distinct clusters of related packets, use additional data sources such as the TCP timestamp to create packet sequences, and then perform further analyses (possibly IoT identification). However, Windows and Android are relatively uncommon in the IoT, so the steady increment has to be validated in order to ascertain the feasibility of DNS IP ID-based IoT deNATing.

As can be seen in Fig. 8, in our experiment the non-IoTs, as well as several IoT devices, appear to follow the abovementioned steady and low incrementation of DNS IP ID. However, for the socket.Wemo.Insight and speaker.Sonos.One IoT models (as well as streamer.Amazon.Fire_TV_Stick) the increment is highly variable, and thus these IoT models cannot be deNATed using this method. Note that light_bulb.TP_Link.LB130 uses DNS only twice a day, so this is another IoT model that can hardly be detect using the IP ID in DNS requests. Also note that since no DNS traffic was captured for doorbell.Amazon.Ring and access_point.TP_Link.TL-WA901ND, the DNS IP-ID was irrelevant to these device models, so they are not shown in Fig. 8. In contrast, our method demonstrates high overall coverage in terms of the variety of IoT models it can detect; it also exhibits encouraging detection results for each IoT model separately (see Tables 5 and 8).

*6.6.4. TCP Timestamp*

This timestamp, which can be found on the header of TCP packets, has been used in several prior studies in the context of deNATing, e.g., Orevi et al. (2017); Tekeoglu et al. (2011). In Linux, which is a common OS in consumer IoT devices, the TCP timestamp is set to zero on boot and incremented regularly every ten milliseconds. The line of time versus TCP timestamp has been proposed by the authors of Tekeoglu et al. (2011) as a means of distinguishing between traffic streams of NATed hosts, however even they noted that this line is not really linear and is highly error prone. In addition, as can be seen in Fig. 9, in 3/6 of the IoT devices in our experiment, the TCP protocol was used in less than 30% of the outbound packets. This means that in the remaining packets the TCP timestamp was not available, and thus these IoT devices could not be deNATed using this feature. In contrast, our method can cope well with any flow, regardless of whether it uses TCP, UDP, or ICMP.

## 7. Discussion

### *7.1. Additional use cases*

In addition to the fundamental use case of precautionary DDoS mitigation by detecting and handling vulnerable IoT models, our approach is well suited for several other use cases. That is, subject to experimental proofs of concept, centralized training and local detection based on (NATed) traffic flows can potentially be utilized for purposes such as the following:

*7.1.1. IoT threat intelligence*

Once a telco successfully deploys our method, it can further utilize the detected IP addresses of (NATed) networks to which potentially harmful IoT devices are connected. This information can provide context to traffic-based *indications of compromise* (IOCs) and enhance anomaly detection, improving the ability to distinguish between malicious and benign activity.

*7.1.2. Classifier sharing*

Once trained, the classifiers can be shared with other telcos. This way, the networks of other telcos are better defended as well. Moreover, if many telcos successfully deploy the trained classifiers

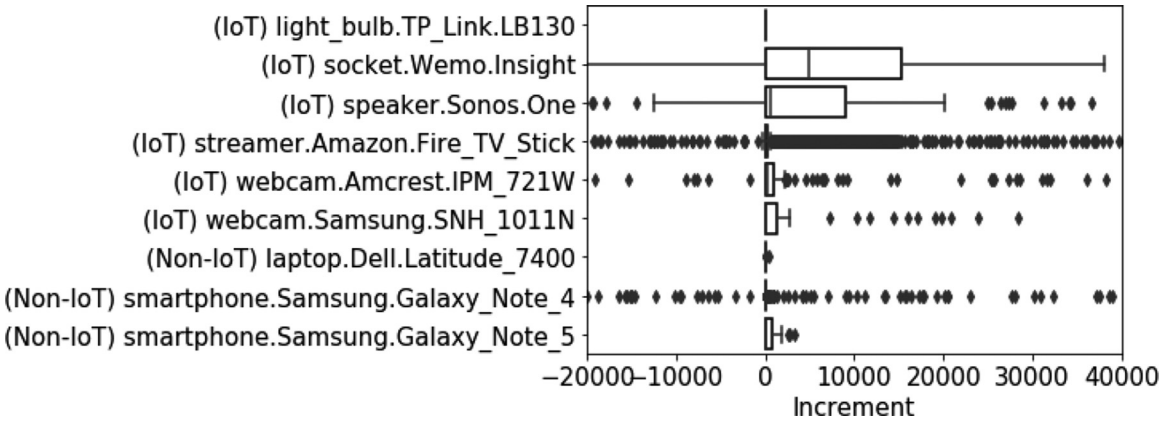


**Fig. 8.** Distribution of the IP-ID's increment in subsequent DNS requests.

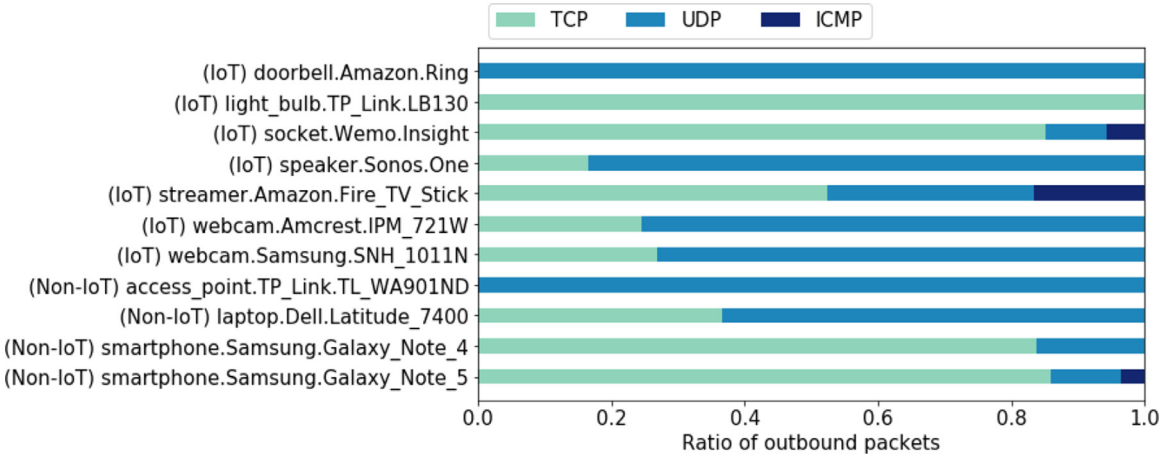


**Fig. 9.** Share of communication protocols among outbound packets.

simultaneously and take appropriate preventive measures, a large-scale DDoS campaign can be dealt with more effectively. In this way, both the telcos' networks and the *target* of the attack (e.g., servers of a DNS provider (Antonakakis et al., 2017)) will be better safeguarded.

#### 7.1.3. Coarse detection

Instead of detecting a specific IoT model, our method can be generalized to "families" of IoT devices. For example, devices with the same manufacturer, or same model number but different firmware (Mitre, 2019) might share the same core vulnerability. For detection, the classifier learns the common denominator in behavior across different firmware releases.

#### 7.1.4. Detection of malicious activity

Bitcoin mining has reportedly (Lee, 2019) sprung up in several locations in Iran, leading to substantial spikes in electricity consumption (and likely bandwidth utilization as well). Our approach is general enough to profile this malicious activity and detect it.

### 7.2. Research limitations

#### 7.2.1. Scalability

The central lab can train IoT model classifiers only after it purchases devices of the related IoT models, configures them, and collects a sufficient amount of NetFlow records. This process might take a few days to complete. However, we are not aware of any other traffic-based detection method that can skip this time-consuming data acquisition stage. To shorten it, the central lab is advised to connect multiple devices of the same model and collect their traffic data simultaneously. Another option, assuming that telcos are willing (or forced, due to regulations) to collaborate on DDoS mitigation, is that telcos would train the classifiers separately in a synchronized manner (each telco addresses another IoT model) and then share the trained classifiers as open source, thus saving training overhead and broadening the variety of detectable IoT models.

It is important to note that a telco does not have to address all IoT models. Instead, we encourage a telco to focus on a small, yet effective subset of IoT models, i.e., the specific IoT models that (1) are found vulnerable to botnet infections, and (2) have the largest install base. For example, as noted by Bertino and Islam (2017) and Kambourakis et al. (2017), the Mirai botnet did not target all IoT devices worldwide but rather only specific IoT models. Also, as noted by Torabi et al. (2018), a group of no more than four IoT device types accounts for about 99.4% of all of the compromised consumer IoT devices.

#### 7.2.2. Patches and firmware updates

These might make the trained classifiers obsolete, however any other data driven classifier is prone to face this challenge as well. We propose dealing with this challenge by first checking the release notes of the updated IoT models to see whether the upgrade has fixed the vulnerability, and then make a more informed decision, particularly if the updated (patched) IoT model is not as vulnerable as the original (unpatched) IoT model the telco could remain with the "old" classifier and stop using it once all of the devices which belong to the unpatched IoT model have been up-

graded. Otherwise, training and dispatching of patched IoT models' classifiers is advised, based on patched IoT devices (the same devices in the central lab however with the updated software).

*7.2.3. Handling multiple contradicting detections*

As part of our proposed detection method, each monitored flow is processed by multiple classifiers (one for each vulnerable IoT model of interest). Occasionally, it might be the case that a flow is classified as *true* by more than one classifier (i.e., mistakenly identified as belonging to multiple IoT models at the same time). In such cases it is likely that at least one of the classifiers is correct, so, from the business perspective, the telco should pay better attention to that domestic network (monitor it more closely to prevent attacks from a vulnerable device). From the algorithmic perspective, an option we have already begun exploring is to classify based on majority voting on several flows, instead of just one. In that ways, a "temporary label" would be assigned for each flow separately, and the final decision would be made later, e.g., once an hour (more or less). This way, there should be less chance of multiple contradicting detections; in addition, majority voting-based detection would be more robust. Another simpler yet more risky approach for cases when multiple detections contradict each other, is to favor the one with the maximal classification score.

*7.2.4. Detecting IoT devices that are already compromised*

Our method focuses on IoT devices that are at a high risk of being infected by a botnet for subsequent use in launching DDoS attacks. We train the related classifiers on benign data so they will excel at detecting those devices *before* they are compromised, as a means of enabling and permitting risk mitigation actions. However, our method is inclined to detect devices which are *already* compromised as well, since in botnet infections the basic functionality of the device remains unchanged. For instance, newly recruited webcams and light bulbs would continue to transmit video and be remotely turned on and off, respectively. In accordance, their traffic data would reflect normal communication patterns with the ordinary servers. The difference is that *in addition* to the normal communication patterns, command & control related communications would typically arise (Kolias et al., 2017), e.g., resolving the hard-coded domain name, and receiving attack commands. The new patterns are likely to be missed by any IoT model classifier, however they should have no effect on the typical pre-infection patterns which accompany the (unchanged) basic functionality.

*7.2.5. Physical security of the local detectors*

In our solution we propose using hardware-based local detectors, deployed outside the customers' premises. We are aware that physical theft of the local detectors or attacks on them are not impossible. This problem is actually also faced by multiple sectors, e.g., IP cameras and smart traffic lights which are spread across cities in public places, as well as WSNs (wireless sensor networks). However, in our use case there are two aspects which limit the impact of such risks: First, we assume that physical theft or attacks are relatively rare. That is, assuming that a typical telco has at least tens or hundreds of thousands of home Internet subscribers, even if dozens of local detectors are stolen or physically tampered with, the impact on telcos is not critical. Second, depending on individual nations' laws, a local detector can be placed (and hidden) within a home's vicinity or close to it in secured places, thus local detectors will be less susceptible to physical attacks than, e.g., IP cameras in smart cities.

## 8. Conclusion and future research

In this paper, we address a real-world problem that has already caused harm in many countries, namely: the recruitment of IoT devices to serve in botnets, later to be used for executing DDoS attacks. Compared with past research that approached this problem with an attack's target in mind, we address this problem from the point of view of a telco, whose infrastructure is put at risk by such attacks. More specifically, we propose a means of detecting exploitable IoT devices that are connected to home networks, a situation which challenges existing IoT identification techniques due to the typical usage of NAT-enabled router gateways. NATed traffic makes flow-based analysis much harder, as packets generated by the same device are difficult to associate to one another. To mitigate the abovementioned risk and efficiently perform IoT deNATing, we propose using the popular NetFlow protocol, because NetFlow (1) already internally aggregates related packets into flows, (2) saves computing and storage, (3) refrains from inspecting payloads, and (4) enables detection, even for non-TCP, non-DNS, and encrypted traffic. We demonstrate that by using no more than one flow, our LGBM-based method enables detection of vulnerable IoT models with an average AUPRC of ($0.992 \pm 0.014$).

This research is a first step towards dramatically mitigating the risk posed to telcos' infrastructure by domestic (NATed) IoT devices. The following is a list of related challenges we consider addressing in the future:

- We empirically tested our method using commercial IoT devices which represent seven manufacturers and three of the most popular types of IoT devices. In order to further validate the scalability of our method, we plan to re-evaluate it using additional IoT devices which represent an even broader range of IoT models, types, and manufacturers.
- It may be beneficial to complement IoT model detection with tools that are capable of performing tasks such as local vulnerability scanning (to check if the vulnerability has been patched) or virtual patching (Hadar et al., 2017).
- Further tuning of the IoT model classifiers could be achieved using two extensions: When available, IETF MUD (the Internet Engineering Task Force's Manufacturer Usage Descriptions) profiles (Lear et al., 2019) could facilitate rule-based classification in a step that could precede the implementation of our method. In addition, a packet level classifier (e.g., DNS domain name-based) could follow our flow-based method and improve the detection performance in ambiguous cases.
- Forging DNS request records and banners (Li et al., 2019) are potential adversarial attacks by which an attacker would make a vulnerable IoT model behave differently and thus avoid detection. Similarly, a spoofing attack, in which an infected device performs many dummy requests to IP addresses and ports that are different from the default ones, could result in missed detection. Although our method is designed to detect vulnerable IoT devices *before* they are exploited, we plan to evaluate the resilience of our method to such adversarial attacks in future research.

In any case, we encourage researchers and practitioners to further explore and utilize our publicly available data.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRediT authorship contribution statement

**Yair Meidan:** Conceptualization, Methodology, Software, Writing - original draft, Project administration, Data curation. **Vinay Sachidananda:** Conceptualization, Writing - review & editing. **Hongyi Peng:** Software, Investigation. **Racheli Sagron:** Software,

Investigation, Visualization. **Yuval Elovici:** Conceptualization, Supervision, Funding acquisition. **Asaf Shabtai:** Conceptualization, Supervision, Funding acquisition.

## Acknowledgments

This project received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 830927. The authors would also like to thank Luis Barriga (Ericsson Research), Dean Eckert, Daniel Benatar, Hanan Libhaber, Tar Wolfson, Liam Habani and Ron Kogas for their valuable contributions.

## References

Abt, S., Baier, H., 2012. Towards efficient and privacy-preserving network-based botnet detection using netflow data.. In: Ninth International Network Conference (INC), pp. 37–50.
AG, P., 2019. Optimize high traffic networks with PRTG sflow monitoring.
Ahmad, A., Floris, A., Atzori, L., 2017. OTT-ISP joint service management: a customer lifetime value based approach. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1017–1022.
Amazon, 2019. Fire stick.
Amcrest,. Amcrest IPM-721W.
Andy, S., Rahardjo, B., Hanindhito, B., 2017. Attack scenarios and security analysis of MQTT communication protocol in IoT system. In: 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI). IEEE, pp. 1–6.
Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al., 2017. Understanding the Mirai Botnet. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, Canada.
Apthorpe, N., Reisman, D., Feamster, N., 2017. A smart home is no castle: privacy vulnerabilities of encrypted IoT traffic. Workshop on Data and Algorithmic Transparency.
Barapatre, P., Tarapore, N.Z., Pukale, S.G., Dhore, M.L., 2008. Training MLP neural network to reduce false alerts in IDS. In: 2008 International Conference on Computing, Communication and Networking, pp. 1–7.
Bekerman, D., Shapira, B., Rokach, L., Bar, A., 2015. Unknown malware detection using network traffic classification. In: Communications and Network Security (CNS), 2015 IEEE Conference on, pp. 134–142.
Bertino, E., Islam, N., 2017. Botnets and internet of things security. Computer doi:10.1109/MC.2017.62.
Bushart, J., Rossow, C., 2019. Padding ain't enough: assessing the privacy guarantees of encrypted dns. arXiv:1907.01317.
Callado, A.C., Kamienski, C.A., Szabó, G., Gero, B.P., Kelner, J., Fernandes, S.F., Sadok, D.F.H., 2009. A survey on internet traffic identification.. IEEE Commun. Surv. Tut. 11 (3), 37–52.
Camiña, J.B., Medina-Pérez, M.A., Monroy, R., Loyola-González, O., Villanueva, L.A.P., Gurrola, L.C.G., 2019. Bagging-randomminer: a one-class classifier for file access-based masquerade detection. Mach. Vis. Appl. 30 (5), 959–974.
Chollet, F., 2015. keras. https://github.com/fchollet/keras.
Cichy, R.M., Kaiser, D., 2019. Deep neural networks as scientific models. Trends Cogn. Sci. 23 (4), 305–317. doi:10.1016/j.tics.2019.01.009.
Cisco, 2006. Cisco 3800 series integrated services routers. Data Sheet.
Cisco, 2019. Cisco catalyst 2960-x and 2960-xr series switches data sheet. Data Sheet.
Cisco.com, 2011. NetFlow Version 9 Flow-Record Format.
Cisco.com, 2017. Cisco - NetFlow.
CNET,. Samsung snh-1011n.
Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297.
Davis, J., Goadrich, M., 2006. The relationship between precision-recall and RoC curves. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 233–240.
DELL,. Latitude 7400.
Dell Latitude e6430.
Doshi, R., Apthorpe, N., Feamster, N., 2018. Machine learning DDoS detection for consumer internet of things devices. 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA. DOI: 10.1109/SPW.2018.00013
Egevang, K., Francis, P., et al., 1994. The IP network address translator (NAT). Technical Report. RFC 1631, may.
El-Maghraby, R.T., Elazim, N.M.A., Bahaa-Eldin, A.M., 2017. A survey on deep packet inspection. In: 2017 12th International Conference on Computer Engineering and Systems (ICCES). IEEE, pp. 188–197.
Ercolani, V.J., Patton, M.W., Chen, H., 2016. Shodan visualized. In: 2016 IEEE Conference on Intelligence and Security Informatics (ISI). IEEE, pp. 193–195. doi:10.1109/ISI.2016.7745467.
Gokcen, Y., Foroushani, V.A., Heywood, A.N.Z., 2014. Can we identify NAT behavior by analyzing traffic flows? In: 2014 IEEE Security and Privacy Workshops. IEEE, pp. 132–139. doi:10.1109/SPW.2014.28.
Guo, H., Heidemann, J., 2018. IP-based IoT device detection. In: Proceedings of the 2018 Workshop on IoT Security and Privacy. ACM, Budapest, Hungary, pp. 36–42. doi:10.1145/3229565.3229572.
Hadar, N., Siboni, S., Elovici, Y., 2017. A lightweight vulnerability mitigation framework for IoT devices. In: Proceedings of the 2017 Workshop on Internet of Things Security and Privacy, pp. 71–75.
Hallman, R., Bryan, J., Palavicini, G., Divita, J., Romero-Mariona, J., 2017. IoDDoS the internet of distributed denial of service attacks - a case study of the Mirai malware and IoT-based botnets. In: Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS. SciTePress, pp. 47–58. doi:10.5220/0006246600470058.
Hamad, S.A., Zhang, W.E., Sheng, Q.Z., Nepal, S., 2019. Iot device identification via network-flow based fingerprinting and learning. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, pp. 103–111.
Hamilton, R., 2016. Mirai Scanner: Are You an Unwitting Mirai Botnet Recruit?Imperva.
Haran, V., 2016. StarHub Attack Raises IoT Security Questions.
Ismail, N., 2018. Telcos struggling to mitigate the threats of cyber attacks. Information Age.
JVN, 2019. Jvnvu#94678942 fon routers may behave as an open resolver.
Industrial Enterprise and IoT Security Threats: Forecast for 2018 | Kaspersky Lab ICS CERT. 2017.
Kambourakis, G., Kolias, C., Stavrou, A., 2017. The Mirai botnet and the IoT Zombie Armies. In: Proceedings - IEEE Military Communications Conference MILCOM, 2017-October doi:10.1109/MILCOM.2017.8170867.
Kashef, A.E., Barakat, N., 2018. Intelligent alarm system to protect small, valuable items. In: 2018 International Conference on Computer and Applications (ICCA), pp. 326–330.
Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y., 2017. Lightgbm: a highly efficient gradient boosting decision tree. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 3146–3154.
Khan, M.A., Salah, K., 2018. Iot security: review, blockchain solutions, and open challenges. Fut. Gener. Comput. Syst. 82, 395–411.
Khatouni, A.S., Zhang, L., Aziz, K., Zincir, I., Zincir-Heywood, N., 2019. Exploring nat detection and host identification using machine learning. In: 2019 15th International Conference on Network and Service Management (CNSM). IEEE, pp. 1–8.
Kirk, J., 2016. Mirai Botnet Knocks Out Deutsche Telekom Routers.
Kit, T. S., 2016a. DDoS attack on StarHub first of its kind on Singapore's telco infrastructure: CSA, IMDA.
Kit, T. S., 2016b. DDoS attack on StarHub first of its kind on Singapore's telco infrastructure: CSA, IMDA.
Kolias, C., Kambourakis, G., Stavrou, A., Voas, J., 2017. DDoS in the IoT: Mirai and other botnets. Computer 50 (7), 80–84. doi:10.1109/MC.2017.201.
Krebs, B., 2016. New Mirai Worm Knocks 900K Germans Offline.
Kumar, D., Shen, K., Case, B., Garg, D., Alperovich, G., Kuznetsov, D., Gupta, R., Durumeric, Z., 2019. All things considered: an analysis of IoT devices on home networks. In: 28th USENIX Security Symposium (USENIX Security'19), pp. 1169–1185.
Lear, E., Romascanu, D., Droms, R., 2019. Manufacturer usage description specification. Internet Engineering Task Force (IETF) doi:10.17487/RFC8520.
Lee, D., 2019. Iran seizes 1,000 Bitcoin mining machines after power spike.
Li, B., Springer, J., Bebis, G., Gunes, M.H., 2013. A survey of network flow applications. J. Netw. Comput. Appl. 36 (2), 567–581.
Li, B., Zhu, Y., Liu, Q., Zhou, Z., Guo, L., 2019. Hunting for invisible smartcam: characterizing and detecting smart camera based on netflow analysis. In: ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE, pp. 1–7.
Liu, Y., Song, T., Liao, L., 2020. Tpii: tracking personally identifiable information via user behaviors in http traffic. Front. Comput. Sci. 14 (3), 1–14.
Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., Lloret, J., 2017. Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access 5, 18042–18050. doi:10.1109/ACCESS.2017.2747560.
M1, 2020. Terms and Conditions: Fibre Broadband.
Mahalle, P.N., 2013. Object classification based context management for identity management in internet of things. Int. J. Comput. Appl. 63 (12), 1–6. doi:10.5120/10515-5486.
Maier, G., Schneider, F., Feldmann, A., 2011. NAT usage in residential broadband networks. In: International Conference on Passive and Active Network Measurement. Springer, Berlin, Heidelberg, Berlin, pp. 32–41. doi:10.1007/978-3-642-19260-9-_4.
Meidan, Y., Bohadana, M., Shabtai, A., Guarnizo, J.D., Ochoa, M., Tippenhauer, N.O., Elovici, Y., 2017. ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In: Proceedings of the Symposium on Applied Computing - SAC '17. ACM Press, Marrakech, Morocco, pp. 506–509. doi:10.1145/3019612.3019878.
Meidan, Y., Sachidananda, V., Peng, H., Sagron, R., Elovici, Y., Shabtai, A., 2020. IoT-deNAT: outbound flow-based network traffic data of IoT and non-IoT devices behind a home NAT. https://doi.org/10.5281/zenodo.3924770. 10.5281/zenodo.3924770.
Microsoft,. Welcome to lightgbm's documentation.
Miettinen, M., Marchal, S., Hafeez, I., Frassetto, T., Asokan, N., Sadeghi, A.-R., Tarkoma, S., 2017. IoT sentinel demo: automated device-type identification for security enforcement in IoT. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, pp. 2511–2514. doi:10.1109/ICDCS.2017.284.

Mitre, 1999. Common vulnerabilities and exposures.
Mitre, 2019. Cve-2019-6015.
Networks, J., 2011. Juniper flow monitoring.
NIST, 2000. National vulnerability database.
Nobakht, M., Sivaraman, V., Boreli, R., 2016. A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow. In: Proceedings - 2016 11th International Conference on Availability, Reliability and Security, ARES 2016 doi:10.1109/ARES.2016.64.
Omitola, T., Wills, G., 2018. Towards mapping the security challenges of the internet of things (IoT) supply chain. Procedia Comput. Sci. 126, 441–450.
Orevi, L., Herzberg, A., Zlatokrilov, H., Sigron, D., 2017. DNS-DNS: DNS-based De-NAT Scheme. NDSS DNS Privacy Workshop.
Ori, Z., Levi, M., Elovici, Y., Rockach, L., Nir Shafrir, Sinter, G., Pen, O., 2008. Identifying computers hidden behind a nat using machine learning techniques. In: ECIW2008-7th European Conference on Information Warfare and Security: ECIW2008. Academic Conferences Limited, p. 335.
Patton, M., Gross, E., Chinn, R., Forbis, S., Walker, L., Chen, H., 2014. Uninvited connections: a study of vulnerable devices on the internet of things (IoT). In: Proceedings - 2014 IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014 doi:10.1109/JISIC.2014.43.
Patton, S., 2018. What is the Real Cost of a DDoS Attack?IoT Tech Expo.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.
Radhakrishnan, S.V., Uluagac, A.S., Beyah, R., 2015. Gtid: a technique for physical device and device type fingerprinting. IEEE Trans. Depend. Secure Comput. 12 (5), 519–532.
Rai, M., Mandoria, H., 2019. Network intrusion detection: a comparative study using state-of-the-art machine learning methods. In: 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 1. IEEE, pp. 1–5.
Ray, A.K., Bagwari, A., 2017. Study of smart home communication protocol's and security privacy aspects. In: 2017 7th International Conference on Communication Systems and Network Technologies (CSNT), pp. 240–245.
Rayome, A. D., 2017. DDoS attacks increased 91% in 2017 thanks to IoT.
Ring,. Video Doorbell.
Ronen, E., Shamir, A., 2016. Extended functionality attacks on IoT devices: the case of smart lights. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 3–12.
Samsung, a. Samsung Galaxy Note 4.
Samsung, b. Samsung Galaxy Note5.
Savage, S., Wetherall, D., Karlin, A., Anderson, T., 2000. Practical network support for IP traceback. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication - SIGCOMM '00. ACM Press, New York, New York, USA, pp. 295–306. doi:10.1145/347059.347560.
Shaked, I., 2019. P2P Hole punching in Home IoT - A best UX or a built in security vulnerability?Firedome.
Shwartz, O., Mathov, Y., Bohadana, M., Elovici, Y., Oren, Y., 2018. Opening pandora's box: effective techniques for reverse engineering IoT devices. In: Eisenbarth, T., Teglia, Y. (Eds.), Smart Card Research and Advanced Applications. Springer International Publishing, Cham, pp. 1–21.
Siboni, S., Shabtai, A., Elovici, Y., 2018. An attack scenario and mitigation mechanism for enterprise byod environments. ACM SIGAPP Appl. Comput. Rev. 18 (2), 5–21.
Sivanathan, A., Sherratt, D., Gharakheili, H.H., Vishwanath, A., 2016. Low-cost flow-based security solutions for smart-home IoT devices. IEEE Advanced Networks and Telecommunications Systems (ANTS), Bangalore, India.
Smith, B. W., 2020. Systems and Methods for Blocking Spoofed Traffic. US Patent App. 16/592,544.
Smith-perrone, J., Sims, J., 2017. Securing cloud, SDN and large data network environments from emerging ddos attacks. In: 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence. IEEE, pp. 466–469.
SONOS,. One: The smart speaker.
Spring, T., 2018. Mirai variant targets financial sector with IoT DDoS attacks.
Stalla-Bourdillon, S., Papadaki, E., Chown, T., 2014. From porn to cybersecurity passing by copyright: how mass surveillance technologies are gaining legitimacy the case of deep packet inspection technologies. Comput. Law Secur. Rev. 30 (6), 670–686.
Taivalsaari, A., Mikkonen, T., 2018. A taxonomy of IoT client architectures. IEEE Softw. 35 (3), 83–88.
Technology Council, F., 2018. 14 predictions for the future of smart home technology.
Tekeoglu, A., Altiparmak, N., Tosun, A.S., 2011. Approximating the number of active nodes behind a NAT device. In: 2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN). IEEE, pp. 1–7. doi:10.1109/ICCCN.2011.6006048.
Than, I., 2016. StarHub: cyber attacks that caused broadband outages came from customers' infected machines.
The HDF Group, 2000–2010. Hierarchical Data Format Version 5.
Torabi, S., Bou-Harb, E., Assi, C., Galluscio, M., Boukhtouta, A., Debbabi, M., 2018. Inferring, characterizing, and investigating internet - scale malicious IoT device activities : a network telescope perspective. In: 48th International Conference on Dependable Systems and Networks (DSN-2018), Luxembourg. DOI: 10.1109/DSN.2018.00064
TP-LINK, a. Smart wi-fi led bulb lb130.
TP-LINK, b. Tl-wa901nd.
Trustwave, 2015. Singtel Acquires Trustwave to Bolster Global Cyber Security Capabilities.
Valdez, E., Pendarakis, D., Jamjoom, H., 2019. How to discover IoT devices when network traffic is encrypted. In: 2019 IEEE International Congress on Internet of Things (ICIOT). IEEE, pp. 17–24.
van der Elzen, I., van Heugten, J., 2017. Techniques for Detecting Compromised IoT Devices. University of Amsterdam Ph.D. thesis.
Verde, N.V., Ateniese, G., Gabrielli, E., Mancini, L.V., Spognardi, A., 2014. No NAT'd user left behind: fingerprinting users behind NAT from NetFlow records alone. In: 2014 IEEE 34th International Conference on Distributed Computing Systems. IEEE, pp. 218–227. doi:10.1109/ICDCS.2014.30.
Victor, G.J., Rao, M.S., Venkaiah, V.C., 2010. Intrusion detection systems-analysis and containment of false positives alerts. Int. J. Comput. Appl 5 (8), 27–33.
WEMO,. Wemo Insight Smart Plug.
Zhang, Z.K., Cho, M.C.Y., Wang, C.W., Hsu, C.W., Chen, C.K., Shieh, S., 2014. IoT security: ongoing challenges and research opportunities. In: Proceedings - IEEE 7th International Conference on Service-Oriented Computing and Applications, SOCA 2014 doi:10.1109/SOCA.2014.58.

**Yair Meidan** is a PhD candidate in the Department of Software and Information Systems Engineering (SISE) at Ben-Gurion University of the Negev (BGU). His research interests include machine learning and IoT security. Contact him at yairme@post.bgu.ac.il.

**Vinay Sachidananda** is a Research Scientist with the National University of Singapore. He has been researching in the area of cyber security focusing on Internet of Things (IoT). Apart from security, he also has been researching privacy issues related to IoT. Contact him at comvs@nus.edu.sg.

**Hongyi Peng** is an BSc student at the National University of Singapore. His research interests include machine learning and software engineering. Contact him at hongyi_peng@u.nus.edu.

**Racheli Sagron** is an BSc student in the SISE Department at BGU. Her research interests include machine learning and software engineering. Contact her at gvilyr@post.bgu.ac.il.

**Yuval Elovici** is a professor in the SISE Department, director of the Telekom Innovation Laboratories, and head of the Cyber Security Research Center at BGU, as well as research director of iTrust at SUTD. His research interests include computer and network security, and machine learning. Elovici received a PhD in information systems from Tel-Aviv University. Contact him at yuval_elovici@sutd.edu.sg.

**Asaf Shabtai** is a professor in the SISE Department at BGU. His research interests include computer and network security, and machine learning. Shabtai received a PhD in information systems from BGU. Contact him at shabtaia@bgu.ac.il.