Scalable Device Identification for IoT Networks using Binary Classification Models at the Edge

Roman Kolcun
University of Cambridge

Richard Mortier
University of Cambridge

ABSTRACT

With the proliferation of IoT devices in households, network-level management is essential for users' security and control. Identifying IoT devices through their network profiles enables the detection of anomalies, such as hacking attempts, misconfigurations, or firmware updates. However, the variability in the types and numbers of devices across households makes training separate models for each household or a single global model impractical. Solutions based on single multiclass classification are not scalable considering the diverse range of devices in households and the constant introduction of new devices. In this technical report, we propose a system that employs separate binary classification models for each device. We evaluate its accuracy in classifying the network traffic over a long period of time. We also investigate the decline in accuracy over time and propose mitigation strategies. Furthermore, we assess the models' sensitivity to changes in network traffic patterns.

To address this, our system utilizes scalable binary classification models that can be adjusted to individual households by downloading only the necessary device-specific models. The system is protocol-agnostic and capable of classifying network traffic, whether it is local network communicatio or over the Internet. Our results show a promising F_1 score of 90-95% on the trained dataset, with accuracy remaining around 80% even after three months. This indicates that periodic model retraining every three months is sufficient.

Additionally, we evaluate the system's ability to detect non-device traffic and find that it can detect deviations even when only 30% of the traffic does not belong to the device. This demonstrates the system's sensitivity to changes in an IoT device's network communication profile, providing users with potential insights into device-related issues.

1 INTRODUCTION

The Internet of Things (IoT) devices are penetrating our homes in even bigger quantities. These devices are always connected to the Internet and are gathering information about our lives. Often, these devices are equipped with microphones and cameras that are always on and can leak private information about our lives. As working from home became a new norm in the post-Covid era, these devices can be represent a larger attack surface that can be used by malignous users to penetrate the home network and subsequently to attack work laptops or phones in order to gain access to employee's work networks.

These devices would benefit from being managed on a network level. However, in order to be managed, they firstly need to be identified. The easiest way how to identify these devices is by their network communication profile, as they are, by definition, connected to the Internet. By learning their network profile, we can also potentially detect, when the device's communication has changed. This change can be caused by various reasons, *e.g.,* a device being hacked, misconfigured, or a firmware update. The obvious place to perform device identification is at the edge, *i.e.,* the home router. In this case the privacy of the user can be preserved as well as the latency can be kept at minimum. The low latency is required not only in terms of communication overhead, but also in terms of providing near real-time network classification. It is not viable to create a network profile from an hour worth of data and classifying it afterwards, as it would create large delays in informing users about device's misbehaviour.

Device identification brings various challenges. The solutions for IoT device identification presented so far focus on creating a single *multiclass* classification model that classify network traffic into one of the pre-trained devices. However, virtually every household contain different set of IoT devices and new devices are released on daily basis. Therefore, it is not viable to either train a specific model for each household or to train one global model classifying every possible IoT device on the market. Training a model at the edge is also not viable due to the computational complexity of model training.

We prose to create a system where a separate model is trained for each device. This model is a *binary* classification model. When a user connects a new IoT device to the network, the router can either ask the user to identify the device and its pre-trained model can be downloaded from a server. Alternatively, the router can create a network profile of the device and send the samples to the server. The server can then run inference on

all the models and suggest to the user which IoT device was connected.

The contribution of this paper are as follows:

- We design a system based on multiple binary classification models and evaluate its accuracy in classification of IoT devices.
- We study how the accuracy declines over time and suggest how this problem can be mitigated.
- We evaluate the sensitivity of our models to the change of network traffic pattern.

The rest of the paper is structured as follows: in (§2) we present related work, in (§3) we present the design of our system, describe the training and evaluation dataset as well as how the models were created and trained. In (§4) we evaluate our system and we conclude our work in (§6).

2 RELATED WORK

IoT device classification has gained large popularity in recent years. It was inspired by network traffic classification while using some characteristics of IoT traffic. Below we list the most popular approaches to IoT network classification and compare them our approach.

2.1 Traditional Machine-Learning Based Methods

Traditionally, several machine-learning techniques have been used to classify IoT network traffic [4, 12, 15–17, 20]. These techniques usually rely on features extracted from TCP/UDP network flows and rely on an expert to specify the set of features. Sivanathan *et al.* [17] built a testbed containing 28 different IoT devices. They used collected data to train a multi-stage classifier using features extracted from 1 hour period of network traffic. These features included port numbers, flow volumes, and DNS requests.

Marchal *et al.* [12] proposed AuDI which extracts patterns of periodic communication generated by 23 IoT devices. Then, kNN model is trained and used to classify newly generated traffic.

Perdisci *et al.* [15] designed IoTFinder, a system for IoT device identification purely based on non-encrypted DNS traffic. They trained a model using the domain names and periodicity with which various DNS requests are sent.

2.2 Deep Learning Based Methods

Deep learning techniques gained popularity mainly in the area of network flow classification [2, 7–9, 14, 18, 21]. Lopez-Martin *et al.* [9] extracted features from first 20 packets of TCP/UDP flows collected from a Spanish backbone Internet network. They used several deep convolutaional ane recurrent neural networks to categorise these network flows. They claim that this approach could be also used for detecting traffic of IoT devices, however, this claim was not evaluated.

Kotak and Yuval [6] converted payloads of TCP flows (while ignoring all other types of traffic) into binary images. They trained a deep neural network on a dataset containing 9 IoT devices and achieved near perfect accuracy.

Ma *et al.* [11] developed MAFFIT, a system based on covolutional neural network for identification of IoT devices. MAFFIT is focused on reliably identifying devices that are represented in a database by very small dataset. They extracted two sets of features, the first based on the byte representation of the packets in a TCP flow, the second based on the packet lengths in the same TCP flow. They evaluated their system on two datasets comprising 16 and 18 devices They achieved accuracy over 99%, however, the features based on byte representation of the packet contained source and destination IP address of each packet.

Zhao *et al.* [22] presented Festic, a system similar to MAFFIT which extracts features from a sequence of packets represented as bytes. It uses convolutaional neural network with few-shot learning method in order to learn about under-represented devices. It was evaluated on two datasets containing 14 and 13 devices rescpectively. They achieved 99% accuracy for the first dataset and 87% for the second one.

2.3 How Is Our Work Different

All of these methods (apart of [10, 13]) use single multi-class classification of the network traffic. These models are not scalable and are tailored for one particular setting or a specific dataset. In our system, we aim to create an easily scalable system that can adapt to any environment with various types and numbers of devices. Additionally, these methods rely on a rather limited dataset, that rarely exceed 25 devices. They also train and evaluate their models on the whole dataset (usually following the 80-20 split). However, as it has been shown, the models trained this way lose their accuracy over time [1, 5, 19]. This deterioration of accuracy was not fully evaluated by prior work and possible mitigation techniques were not studied.

Unlike our approach which is intended to run on an edge device, Ma *et al.* [10] focuses on identifying IoT devices behind a NAT.

Some of them rely on long inference time [12, 17].

Many of these approaches rely on extracting features only from certain protocols [] or knowing the the values of some features in advance (*e.g.,* the number of IP addresses or ports contacted, DNS queries, etc.) [13, 17]. Our approach does not rely on any specific protocol and does not require any knowledge of the network traffic in advance. Additionally, it can classify any type of network communication, *i.e.,* all network protocols.

3 SYSTEM DESIGN

3.1 Dataset

Our training and evaluation dataset consists of `pcap` files collected from our test bed consisting of 40 different IoT devices. The list of devices can be found in Table 1. Data were collected over slightly more than one year period. All devices were connected to a computer serving as a DHCP server and router. All traffic passing through the server was collected and stored in separate files for each device in a `pcap` format.

Given that the network traffic was collected from a test bed, the data collected are often from idle devices. Additionally, some devices might have been temporarily removed from the test bed or disconnected for arbitrary long time. This might have been caused by an error on a device requiring to restart (or reset) the device or by accidentally unplugging it. The test bed was occasioanlly used to perform various experiments using automated scripts. For all of these reasons, the distribution of data samples collected varies over time depending on the device. Therefore, when splitting the dataset we are not doing so by time but by percentage, *i.e.,* when we use first quater of the data set, we do take data from the first 91 days, but the first 25% of data, no matter how long it took to collect these data.

3.1.1 `pcap` to Training Dataset. To transform raw network traffic in a `pcap` format into a data set suitable for Convolutional Neural Network (CNN), we treat the data packets as a raw signal. The size of each packet sent or received represents a data point. We assign a positive value to packets sent and a negative value to packets received. Because the packets are sent at arbitrary time, we need to resample the values at fixed rate. Because we aim to classify the IoT device in near real-time, we chose rather small sampling value of $T = 0.1$ seconds. For each sampling interval, we sum the sizes of sent and received packets. Once we have a function sampled at fixed interval we can perform Fast Fourier Transform (FFT). We perform FFT on a fixed width of samples w_s with 50% overlap over consecutive runs of FFT. As an input, we take the absolute value of the first half $h = w_s/2$ (due to the symmetrical output) of the apmlitudes returned by FFT. This output represents one column of the input. If there is no communication in given sampling window, *i.e.,* no packets are sent or received, we ignore the window and produce no output.

An input for CNN is a single layer of 2D grid of fixed height and width $h \times w$. Each column is generated by one run of FFT. New columns are appended from the right and the leftmost column is discarded. The number of the samples for FFT affect how often we generate a new data point, *i.e.,* it takes $t_s = h \times s$ seconds to generate a new column. The width of the input affects how long it takes to generate the first data

Table 1: List of IoT devices in our test bed and the number of samples collected for each device.

Device	Num of data points
appkettle	1,335,258
blink-security-hub	2,618,290
bosiwo-camera-wifi	1,908,817
lefun-cam-wired	134,750
insteon-hub	303,080
echoplus	3,209,758
meross-dooropener	1,945,704
smartlife-bulb	1,138,796
xiaomi-ricecooker	4,089,849
ubell-doorbell	2,807,987
appletv	1,300,405
tplink-bulb	1,534,824
google-home	4,421,204
icsee-doorbell	2,325,103
t-wemo-plug	1,945,960
echospot	3,402,838
nest-tstat	1,535,129
sousvide	4,685,930
smartlife-remote	4,813,953
netatmo-weather-station	1,472,515
wansview-cam-wired	3,433,043
xiaomi-plug	2,607,633
xiaomi-hub	4,498,146
lightify-hub	1,465,097
bosiwo-camera-wired	4,035,061
tplink-plug2	1,235,519
allure-speaker	4,525,653
honeywell-thermostat	1,593,642
smarter-coffee-mach	1,376,565
roku-tv	1,606,423
yi-camera	3,309,018
firetv	2,638,748
echodot	2,834,508
smartthings-hub	3,129,344
reolink-cam-wired	922,265
t-philips-hub	1,179,407
switchbot-hub	1,584,022
ring-doorbell	675,907
blink-camera	150,224
samsungtv-wired	2,623,008

point that can be used as an input for CNN, *i.e.,* it takes $t_b = w \times t_s$. We refer to this period as a *bootstrapping period*. This period is the minimum period after which the system can start classifying the network traffic. Note, that this period can be longer if the device communicates only sporadically. After this bootstrapping period, a new network classification is done every h seconds (provided that the device communicates over the network). It is important to note, that we do not distinguish between various protocols or whether the communication is on local network or over the Internet. All communication is treated equally and all types of communication can be classified.

In this paper we evaluate four sets of input sizes: $75 \times 60, 75 \times 120, 150 \times 60, 150 \times 120$. In the case of $h = 150$ (and $h = 75$ respectively) and $T = 0.1$ a new network sample is produced every 15 seconds (7.5 seconds, respectively). The minimal bootstrapping phase ranges from $t_b = 30$ minutes in the case of input size 150×120 to $t_b = 7.5$ minutes in the case of input size 75×60.

3.1.2 Sample Normalisation. The input into the CNN must be normalised. Usually, the input is normalised to fit within the $(-1, 1)$ interval. Uniform normalisation of the whole dataset led to untrainable data due to several extremely high values that rendered all other values very close to zero. Therefore, we used *per sample* normalisation, where each sample is normalised independently. We trained the model and evaluated it on a small subset of trainig data using serveral normalisation methods. These methods were composed of three simple operations. Let x_{in} be the input sample. Then these three simple operations were: (*i*) *logarithm* where, $x_{out} = \log_{10} x_{in}$, (*ii*) *max* where, $x_{out} = x_{in}/max(x_{in})$, and (*iii*) *centre* where $x_{out} = 2 \times x_{in} - 1$ (where x_{in} is already normalised value from interval $(0, 1)$).

The tested normalisation methods were: (*i*) logarithm + max, (*ii*) logarithm + max + centre, (*iii*) max + centre, and (*iv*) max.

3.1.3 Training Data Set. For the training purposes we randomly select samples of fixed width and height. If necessary, we select samples from predefined interval given by percentage interval, *e.g.,* $(0 - 0.25)$ refers to the first quater of samples for each device. As we mentioned above, we do not select by date as the traffic of each device occurred at diffrent time frames.

Because we train a separate model for each device, for each sample belonging to the device we randomly choose 5 other samples belonging to other devices. During the training phase we set the class weights in reverse ratio 5:1, *i.e.,* each sample belonging to the device has five times more weight than other samples. We also investigated higher samples ratios, but their contribution to higher accuracy was very limited, while their significantly increased the training time.

3.1.4 Testing Data Set. For the evaluation purposes, we simulate the operation of our system in the real world. Therefore, we select a consecutive stream of samples of fixed length l from random positions. Two consecutive samples differ virtually by one column. The leftmost column is removed, a new column is added to the right, and all other columns are shifted to the left. We choose l so that it represents a significant length of operation on real-world data. Unfortunately, due to the large number of samples in our dataset it is not possible to evaluate the models on the whole dataset using a sliding window of 1. Therefore, rather than choosing random samples from the data set, we use random starting points at least l positions apart. From each starting position we sample l samples using a sliding window of 1.

3.1.5 Fault Injection. The goal of our system is not only to be able to classify network traffic of a connected IoT device during its normal activity, but also to be able to identify when its communication profile changes. This change can be caused by a device being hacked, misconfigured, or due to a firmware update. We believe, that in this case, the owner of the device should be notified about the change and it is upon the user to take further actions. We simulate the change in the network communication of the device by generating random numbers as the output of FFT. This leads to insertion of a column of random numbers into the input sample. Our intention is to imitate communication that is not typical for the device. Our system can insert atypical communication either into: (*i*) random columns, (*ii*) into every $n - th$ column, or (*iii*) into every column (special case where $n = 1$. The reasoning behind this three scenarios is that if a device is hacked, the hacker might want to hide the excessive communication by duty cycling it with normal communication. These three scenarios simulate situations when the device communicate (*i*) with random delay, (*ii*) with exact duty cycling, (*iii*) constantly. We investigate wheter and how fast can our system detect different types of rouge traffic.

3.2 CNN Model

We built our model using Keras and TensorFlow library. The structrure of our Convolution Neural Network is depicted in Figure 1. It is a series of Convolutional Layers with a Dense Layer at the end. The last layer is a single neuron with sigmoid activation. Binary crossentropy was

Table 2: Evaluation of various normalisation techniques.

Input size	log	centre	F_1 score
150 × 120	✓	✓	0.18
	✗	✓	0.16
	✓	✗	0.55
	✗	✗	0.76
150 × 60	✓	✓	0.18
	✗	✓	0.15
	✓	✗	0.60
	✗	✗	0.87
75 × 120	✓	✓	0.26
	✗	✓	0.20
	✓	✗	0.46
	✗	✗	0.86
75 × 60	✓	✓	0.17
	✗	✓	0.15
	✓	✗	0.55
	✗	✗	0.87

used as a loss function and Adam as an optimizer with learning rate of $1e-4$.

4 EVALUATION

In the evaluation of our system we are focusing on three metrics: (*i*) accuraccy of classification ((§4.3), (*ii*) detection of misbehaviour ((§4.4), and (*iii*) stability of classification over time ((§4.5)). The accuraccy of classification is important to correctly classify the network traffic and to be able to identify which device is connected to the network. It is also important to identify when a device deviates from its normal operation and the network traffic is differs from the device's regular operation.

4.1 Normalisation

Various normalisation techniques were tested on a small subset of the training data set, where just 2000 samples were randomly sampled for each device. Models were trained for each device while using various normalisation techniques described in (§3.1.2). In every case the per-sample normalisation was applied, *i.e.,* $x_{norm} = x/max(x)$ for each sample individually. As can be seen in Table 2 the simplest per-sample normalisation is the most effective, achieving on average F_1 score of 84%. The second best normalisation is using logarithm prior to the normalisation and it achieves the average F_1 score of 54%. Using the logarithm and centering of value around 0 achieves on average F_1 score of 20%. The worst performance is achieved with just centering the values around 0, where the average F_1 score is just 17%.

4.2 Data Set Split

The data set was split into four disjoint *intervals*, where each interval contained 25% of continuous network traffic of each device. For training purposes, from each interval we randomly sampled $5k$ data samples for each device. Therefore, there were $20k$ ($5k \times 4$) data points sampled for each device, and the whole training dataset contained $800k$ data samples ($20k$ for each of 40 devices). The training data represent between 0.13% for Blink camera to only 0.004% for Smartlife Remote device. On average, the training data represent just under 0.01% of the whole dataset.

We chose the random sampling because other sampling techniques, *e.g.,* continuous stripes or samples at fixed intervals led to lower classification accuracy.

During training, the data set was split with 80:20 ratio between training and testing. The models were trained as if they would have been trained in real-world scenario, *i.e.,* continuously on each file. The files were ordered in ascending order depending on from which interval they were sampled. Next, the model was trained on each file for the duration of 10 epochs and saved on local disk. When training on the next file, the previously saved model was loaded, *i.e.,* the model trained on forth internval, was already trained on previous three intervals.

For the evaluation, we tried to immitate the real-case scenario, where the model is used to continuously evaluate the incoming traffic. For each device and interval we chose 100 random points (at least 100 samples apart). From these points we sampled 100 continuous data samples. This represents as if a router was sampling and evaluatiing the network traffic generated by an IoT device for the duration of 100 sampling intervals. Therefore we ended up with $10k$ data samples for each of device–interval pair. Data from each interval were evaluated independently.

4.3 Accuracy of Classification

The accuracy of classification shows how well the model can classify the network traffic generated by the device. We use a standard evaluation metric [3], F_1 score for the overall measure of the model's accuracy and is defined as:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}, F_1 \in \langle 0, 1 \rangle$$

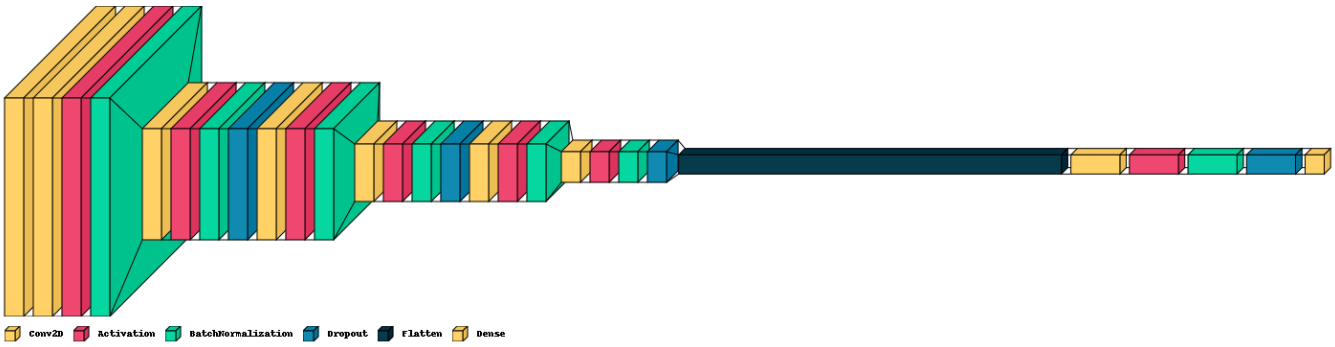

Figure 1: Convolutional Neural Network

where $True\ Positive$ represents the number of times when a device was correctly classified, $False Positive$ the number of times when the device was classified as some other device, $True\ Negative$ the number of times when the device is correctly not classified, and $False Negative$ the number of times when a device is classified instead of a correct device. F_1 score represents a harmonic mean of precision and recall.

We evaluate the model's accuracy in two scenarios: (*i*) *highest confidence*, and (*ii*) *confident enough*. In the first scenario, we run inference of a sample on *all* models and as the predicted device we choose the model with the highest confidence. This is similar to a scenario, where a multi-class classification model is used. The sample is classified correctly, if the predicted IoT device is the same as the device that generated the sample.

In the *confident enough* scenario we run the inference of a sample only against the model representing the device that generated the sample. We consider the sample to be correctly classified if the model's confidence is over 0.5. This represents a real-life scenario, where a sample is run only against the model representing given device. If the confidence drops below 0.5, we can assume that the model does not recognise the traffic as generated by the device.

As expected, the confident enough measurement increases the F_1 score between 2 and 18 percentage point (5 percentage points on average). The increase is not significant when the models were evaluated on the same interval as they were trained on – 3 percentage points on average. However, the difference increases to 6 percentage points when the model is evaluated on the following interval, and reaches 9 percentage points when the model is evaluated on the interval 2 periods apart. This suggests that a good enough approach can keep the accuracy of the model for longer period when compared to the approach with the highest confidence.

We also evaluate the influence of the input size on the accuracy of the model. We use four different sample sizes as described in (§3.1.1). The sizes of the input are $75 \times 60, 75 \times 120, 150 \times 60, 150 \times 120$. It can be seen from the Table 3 that the larger the input size is, the higher F_1 score it achieves. We scale the input size in both directions: (*i*) the sampling interval (*i.e.,* heigh of the input) and (*ii*) the overall sampling length (*i.e.,* width of the input). The results show that scaling up in any direction increases the F_1 score between 2 and 2.5 percentage points and by scaling up in both directions the F_1 score increases by 4.5 percentage points.

4.4 Detection of Misbehaviour

Detecting when an IoT device's network traffic profile starts to deviate from its normal operation is a hard task. Here, we inject the regular network traffic with random noise using techniques described in (§3.1.5). We focus on evaluating three metrics: (*i*) speed of detection, (*ii*) trigger point of detection, and (*ii*) streaks of mis-classification.

It is important to detect when a device starts to misbehave as soon as possible. To study the speed of detection we inject various percentage of faults into each sample and measure how when the system is able to detect the sample as not generated by given device.

We asssume that a potential attacker might want to hide its activity and spread it over longer period of time, *i.e.,* hiding it within the normal traffic. Therefore, we study several scenarios where the faulty traffic might appear. It can be either continuous, appearing at random times, or appearing at regular intervals.

We also evaluate *streaks* of mis-classification, *i.e.,* how many consecutive samples are mis-classified. The idea behind this metric is to find out whether malicious traffic leads to longer streaks of mis-classification and whether

Table 3: F_1 score of various classifiers. *hc* shows the *highest confidence* scenario while *ce* shows the *confident enough* scenario.

Input Size	75 × 60		75 × 120		150 × 60		150 × 120	
Prediction Window	hc	ce	hc	ce	hc	ce	hc	ce
0	0.90	0.94	0.91	0.95	0.91	0.94	0.93	0.95
1	0.72	0.78	0.75	0.80	0.74	0.78	0.76	0.80
2	0.47	0.55	0.50	0.56	0.50	0.56	0.54	0.59

this can be used as an additional signal to identify that device was hacked.

4.5 Classification Stability

It has been shown that accuracy of classification of IoT devices decreases over time [5]. More importantly, the accuracy decreases immediately when tested on data that were not part of the training set. It is important for a system to keep the accuracy at acceptable level for a reasonable amount of time after the model was trained. Therefore, we train models on a subset of data that cover continuous period of time. Next, we evaluate it on another period that was not part of the training data. The details of how the data set was split is detailed in (§3.1.3).

When the system is evaluated on the same interval as it was trained on, it achieves the average F_1 score of 91% (94% respectively for the confident enough scenario). When the model is evaluated on the interval following immediately after the training one, the F_1 score decreases to 69% (75% respectively) in the case of the 75 × 60 model. Evaluating on data two intervals apart leads to even further decrease of F_1 score to 47% (55%), while three model separated by three intervals achieves F_1 score of 42% (54%).

Even though this decrease seems to be large, it is important to remember that each interval represents roughly 3 months worth of data. This means that the system can achieve over 75% accuracy for half a year without retraining. However, the further decrease in accuracy suggests that the system will require a sporadic update to the models or to locally retrain them.

5 PROTOTYPE

5.1 Data Preparation

The dataset consists of separate `pcap` files. Each file represents network communication of one device over the course of 24 hours. To generate our evaluation dataset, for most of the devices we selected the largest file, *i.e.,* the largest amount of data a device generated within 24 hours. If a device generated more than 150 MB, we chose the median sized file from files over 150 MB. The reason why we did not choose the largest files for all devices is that we wanted to generate a dataset representing larger than usual data traffic, but we also did not want to overload our evaluation platform. There were 12 devices that generated more than 150 MB. The amount of data generated by each device ranged between 5.3 MB for the kettle and 491 MB for the Apple TV device. On average, each device generated 95 MB of network traffic. This dataset represents a more than average network traffic than can be expected on a network of 40 IoT devices.

Next, timestamps of all files were synchronised to the same time using `editcap`[1] utility. Subsequently, all files were merged together using `mergecap` utility. The final version of the file contains data traffic of all devices during a 24-hour window was 3.8 GB. In every 15 second window of our evaluation dataset, there are on average 33 devices sending or receiving network traffic.

To replay the network traffic, the file was uploaded to Raspberry Pi 4 (8 GB RAM) and replayed using `Tcpreplay`[2] utility. Because the network card on Raspberry Pi does not support MTU larger then 1500, we truncated the larger packets to this length.

5.2 Prototype Design

We implemented the prototype of the edge IoT device identification in Python. The prototype runs on a Raspberry Pi 4 (8 GB RAM) listening to specific interface. During our evaluation the prototype was listening to the local Ethernet port, because `Tcpreplay` does not support replaying the network traffic on a wireless network card.

The prototype listens to a specific port for a predefined *sampling window* t_s using `scapy`[3] library. During this sampling window the system collects source and destination MAC addresses, timestamp, and the packet size of each packet. Upon expiry of the sampling window, the packets are split into separate lists depending on the source and destination MAC address. Depending on

[1] `editcap` and `mergecap` are part of the Wireshark network protocol analyser wireshark.org
[2] tcpreplay.appneta.com
[3] scapy.net

whether the packet was sent or received by the device, the value of the packet size is assigned either positive or a negative value. Each list is resampled at 10 Hz by summing the packet sizes and data are pushed into a buffer.

Next, a Fast Fourier Transform (FFT) is performed on the buffer and the result is stored in the final buffer of predefined size. This buffer represents the network spectrogram of given device. Once, the spectrogram is full, new data replaces the oldest ones. Last, the inference is run on the model that represents given device.

The models were converted to a TensorFlow Lite[4] model and TensorFlow Lite runtime was used to run inference on a model.

5.3 Prototype Evaluation

In our evaluation we focus on three parameters: (*i*) CPU utilisation, (*ii*) memory usage, and (*iii*) processing delay. The system was evaluate using three spectrogram (and therefore also model) sizes: (*i*) 75×30, (*ii*) 75×60, and (*iii*) 150×120. These sizes show how scalable the system is and what are the minimum and maximum requirements.

5.3.1 CPU Utilisation. To measure the CPU utilisation we used `mpstat` utility. We logged the CPU load every second. To establish the baseline we run `mpstat` for an hour while running the `Tcpreplay` utility. We found out, that `Tcpreplay` utilises one core of the 4-core CPU to 100% virtually all the time. Next, we run the IoT device identification system for 24 hours while measuring the CPU utilisation. From the final average CPU utilisation we deduct the CPU utilisation used by `Tcpreplay`.

Figure 2 depicts the average CPU utilisation for a five minute rolling window using various spectrogram sizes. On average, the CPU utilisation for the smallest spectrogram size (75×30) is only 9 percentage point, while the for the 75×60 spectrogram it is on average 10 percentage point. The largest spectrogram on average utilises the CPU by less than 12 percentage points.

It can also be seen that the highest utilisation is achieved at the beginning when the models are being loaded up and data structures are being initialised. After this initial period, the CPU utilisation drops to the average level of utilisation. The CPU utilisation does not depend on the overall amount of data produced by IoT devices but by the number of devices that communicated within the sampling window. This can be expected as the most CPU is used to calculate FFT and run inference

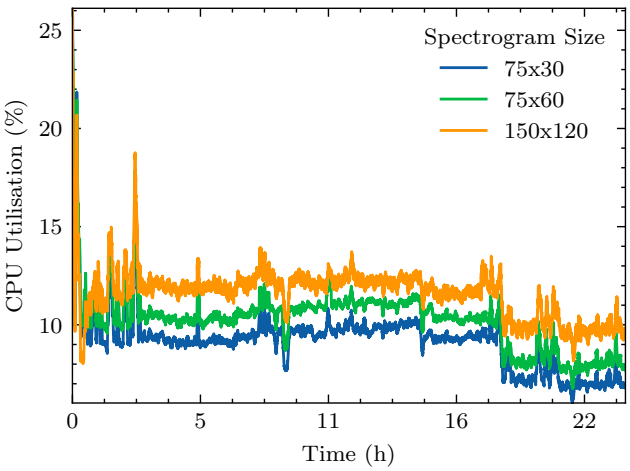

Figure 2: CPU utilisation of Raspberry Pi 4 with various spectrogram sizes.

of the model and not on calculating sizes of the packets and resampling of data.

5.3.2 Memory Usage. To measure the memory usage of the system, we utilised `memory-profiler`[5] Python library. This utility measures the memory usage of the program every 0.1 second and logs it into a file. Figure 3 depicts the memory utilisation for different spectrogram sizes. It can be seen that the memory usage highly depends on the size of the spectrogram, and therefore also the model size. After the initialisation and loading the models into the memory, the memory usage stabilises and does not increase over time. The maximum memory usage for the spectrogram of size 75×30 is 294 MB, for 75×60 is 395 MB, and for 150×120 is 951 MB. It is important to note, that this memory usage is for 40 IoT devices. A household with fewer devices would require smaller amount of memory.

5.3.3 Processing Delay. This part of evaluation focuses on the processing delays caused by each step of the IoT device identification process. In particular, we measured the delay of (*i*) creating summaries for each sampling window, (*ii*) resampling data, (*iii*) calculating Fast Fourier Transform (FFT), and (*iv*) running inference on a model. After every sampling window t_s we perform each of the action in sequence for all the data received for each device. Because, in each sampling window the number of devices that send and/or receive data varies, we normalise the time as an *average delay per device*.

[4] tensorflow.org/lite

[5] github.com/pythonprofilers/memory_profiler

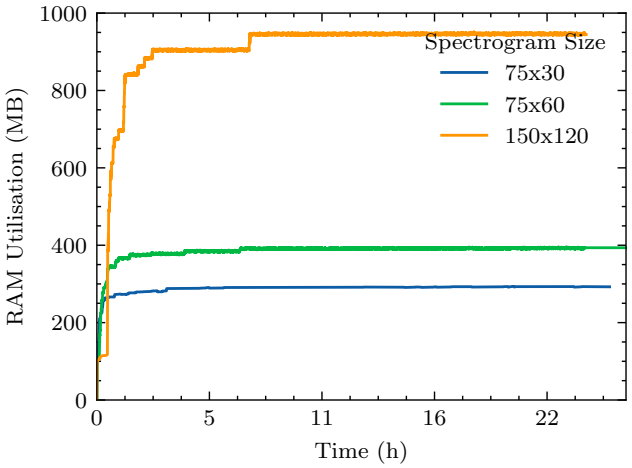

Figure 3: Memory utilisation of Raspberry Pi 4 with various spectrogram sizes.

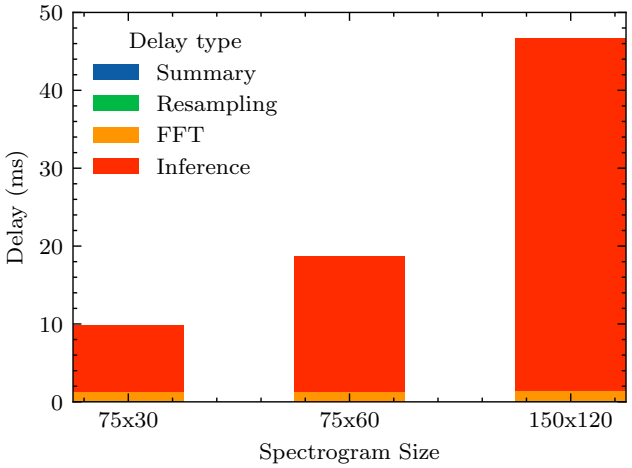

Figure 4: Memory utilisation of Raspberry Pi 4 with various spectrogram sizes.

Figure 4 displays each type of IoT device identification delay for various spectrogram sizes. As can be seen, the largest delay is caused by running inference on *tflite* model. The second largest contributor to the delay is the calculation of FFT. The delay of creation of summaries and resampling of data is virtually non-existent. The overall delay per device is 9.8 ms for spectrogram size 75×30, 18.6 ms for 75×60, and 46.8 ms for 150×120. It means, that with 7.5 second sampling window (15 second respectively for the 150×120 spectrogram), the router can process roughly 760 devices for the smallest and 320 for the largest spectrogram, which is well above the threshold of a typical household.

The IoT device identification delay can be reduced by parallelisation and utilising all available CPU cores. This can significantly speed-up the inference time by executing inference on several models at the same time. Another possible speedup can be achieved by utilising eBPF to pre-process the packet sizes inside the kernel.

6 CONCLUSION

The number of IoT devices deployed in our households is ever growing. These devices represent a large surface attack for malicious users who might want to take advantage of these devices. Users would benefit from being able to manage these devices on network level. In order to be able to manage them, we must be able to identify them. The obvious way how to identify them is to use their network profile, as all of these devices are connected to the Internet. By creating their network profile, it is also possible to detect when their network communication pattern changes. The change can be a sign of an IoT device being hacked, misconfigured, or its firmware was updated.

However, one of the challanges is that no two homes are homes are the same. The number and type of devices changes from one household to another. It would be infeasable to train a separate model for each household. Similarly, it would be infeasable to train a single model capable of classifying tens of thousands of devices currently available at the market and retraingin the model every time a new device is introduced.

In this technical report we present a system that is built on top of multiple binary classification models. This provides the system with scalability as it is able to adjust to any household by downloading only the models for the devices deployed in given household.

The system is not focused on any specific type of traffic or protocol and is capable to classify any network traffic, whether on local network or over the Internet.

We have evaluated our system on data collected from a testbed comprising of 40 IoT devices for over a year. We have evaluated our system on data samples of various sizes and studied the influence of the size of the sample on the overall accuracy of the model. We have achieved F_1 score of $90 - -95\%$, depending on the size of the input, when the models were evaluated on the dataset as they were trained on. We have showed that the accuracy of the model degrades over time. However, the accuracy of the model stayed at nealy 80% even after three month. This suggest that retraingin of the model can be sporadic an occur every three months.

We have also evaluated the ability of the system to detect the traffic that does not belong to the device. We

showed that the system is capable to detect even when only 30% of the traffic does not belong to the device. This suggests that our system is sensitive enough to detect when the IoT device's network communication profile changes. This can give a hint to the user about a potential problem with the device.

In our future work we would like to focus our effort on making the models more stable over time in terms of accuracy. We will explore methods how to keep the accuracy at high level over longer periods of time, thus decreasing the need to retrain the models. We would also like to focus on making the model inference run faster by leveraging caching of intermediary results.

REFERENCES

[1] Dilawer Ahmed, Anupam Das, and Fareed Zaffar. 2022. Analyzing the feasibility and generalizability of fingerprinting Internet of Things devices. *Proceedings on Privacy Enhancing Technologies* 2022, 2 (2022), 578–600.

[2] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. 2018. Acquisitional Rule-Based Engine for Discovering Internet-of-Thing Devices. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18)*. USENIX Association, USA, 327 – 341.

[3] Gavin Hackeling. 2014. *Mastering Machine Learning With Scikit-Learn*. Packt Publishing.

[4] Shilpa P Khedkar and R. AroulCanessane. 2020. Machine Learning Model for classification of IoT Network Traffic. In *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 166–170. https://doi.org/10.1109/I-SMAC49090.2020.9243468

[5] Roman Kolcun, Diana Andreea Popescu, Vadim Safronov, Poonam Yadav, Anna Maria Mandalari, Richard Mortier, and Hamed Haddadi. 2021. Revisiting iot device identification. *arXiv preprint arXiv:2107.07818* (2021).

[6] Jaidip Kotak and Yuval Elovici. 2021. IoT Device Identification Using Deep Learning. In *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, Álvaro Herrero, Carlos Cambra, Daniel Urda, Javier Sedano, Héctor Quintián, and Emilio Corchado (Eds.). Springer International Publishing, Cham, 76–86.

[7] Rui Li, Xi Xiao, Shiguang Ni, Haitao Zheng, and Shutao Xia. 2018. Byte Segment Neural Network for Network Traffic Classification. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS.2018.8624128

[8] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. FS-Net: A Flow Sequence Network For Encrypted Traffic Classification. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1171–1179. https://doi.org/10.1109/.2019.8737507

[9] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access* 5 (2017), 18042–18050. https://doi.org/10.1109/ACCESS.2017.2747560

[10] Xiaobo Ma, Jian Qu, Jianfeng Li, John CS Lui, Zhenhua Li, and Xiaohong Guan. 2020. Pinpointing hidden IoT devices via spatial-temporal traffic fingerprinting. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 894–903.

[11] Xiaotian Ma, Yipeng Wang, Yingxu Lai, Wenxu Jia, Zijian Zhao, Huijie He, Ruiping Yin, and Yige Chen. 2023. A Multi-perspective Feature Approach to Few-shot Classification of IoT Traffic. *IEEE Transactions on Green Communications and Networking* (2023), 1–1. https://doi.org/10.1109/TGCN.2023.3269842

[12] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and N. Asokan. 2019. AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1402–1412. https://doi.org/10.1109/JSAC.2019.2904364

[13] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2177–2184.

[14] Jorge Ortiz, Catherine Crawford, and Franck Le. 2019. DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices. In *Proceedings of the International Conference on Internet of Things Design and Implementation (IoTDI '19)*. Association for Computing Machinery, New York, NY, USA, 106–117.

[15] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. 2020. IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. 474–489. https://doi.org/10.1109/EuroSP48549.2020.00037

[16] Ola Salman, Imad H Elhajj, Ali Chehab, and Ayman Kayssi. 2022. A machine learning based framework for IoT device identification and abnormal traffic detection. *Transactions on Emerging Telecommunications Technologies* 33, 3 (2022), e3743.

[17] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. 2019. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2019), 1745–1759.

[18] Chong Sun, Narasimhan Rampalli, Frank Yang, and AnHai Doan. 2014. Chimera: Large-scale Classification Using Machine Learning, Rules, and Crowdsourcing. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1529–1540.

[19] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2020. Packet-level signatures for smart home devices. In *Network and Distributed Systems Security (NDSS) Symposium*, Vol. 2020.

[20] Shi-Jie Xu, Guang-Gang Geng, Xiao-Bo Jin, Dong-Jie Liu, and Jian Weng. 2022. Seeing Traffic Paths: Encrypted Traffic Classification With Path Signature Features. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2166–2181. https://doi.org/10.1109/TIFS.2022.3179955

[21] F. Yin, L. Yang, Y. Wang, and J. Dai. 2021. IoT ETEI: End-to-End IoT Device Identification Method. In *2021 IEEE Conference on Dependable and Secure Computing (DSC)*. 1–8. https://doi.org/10.1109/DSC49826.2021.9346251

[22] Zijian Zhao, Yingxu Lai, Yipeng Wang, Wenxu Jia, and Huijie He. 2022. A Few-Shot Learning Based Approach to IoT Traffic Classification. *IEEE Communications Letters* 26, 3 (2022), 537–541. https://doi.org/10.1109/LCOMM.2021.3137634