Find my IoT Device – An Efficient and Effective Approximate Matching Algorithm to Identify IoT Traffic Flows

Thomas Göbel[1], Frieder Uhlig[2], and Harald Baier[1]

[1] Research Institute CODE, Universität der Bundeswehr München, Munich, Germany
[2] Technical University Darmstadt, Darmstadt, Germany
{thomas.goebel,harald.baier}@unibw.de
frieder.uhlig@stud.tu-darmstadt.de

Abstract. Internet of Things (IoT) devices has become more and more popular as they are limited in terms of resources, designed to serve only one specific purpose, and hence cheap. However, their profitability comes with the difficulty to patch them. Moreover, the IoT topology is often not well documented, too. Thus IoT devices form a popular attack vector in networks. Due to the widespread missing documentation vulnerable IoT network components must be quickly identified and located during an incident and a network forensic response. In this paper, we present a novel approach to efficiently and effectively identify a specific IoT device by using approximate matching applied to network traffic captures. Our algorithm is called `Cu-IoT` and is publicly available. `Cu-IoT` is superior to previous machine-learning approaches because it does not require feature extraction and a learning phase. Furthermore, in the case of 2 out of 3 datasets, `Cu-IoT` outperforms a hash-based competitor, too. We present an in-depth evaluation of `Cu-IoT` on different IoT datasets and achieve nearly 100% classification performance in terms of accuracy, recall, and precision, respectively, for the first dataset (*Active Data*), and almost 99% accuracy and 84% precision and recall, respectively, for the second dataset (*Setup Data*), and almost 100% accuracy and 90% precision and recall, respectively, for the third dataset (*Idle Data*).

Keywords: Internet of Things (IoT) · IoT Device · Device Classification · Device Identification · Network Forensics · Network Traffic Fingerprinting · Approximate Matching · Multi Resolution Hashing (MRSH) · Cuckoo Filter

1 Introduction

Typically, Internet of Things (IoT) devices have limited security capabilities, because their hardware is often too weak and their software is often too focused on a specific use case. There have been many documented security flaws found in the past on consumer IoT devices such as baby monitors, security cameras,

doorbells or smart thermostats [31]. As patching is practically impossible, IoT devices are a primary target for attackers, especially considering that many of these devices have an extremely long lifetime (e.g., smart home devices such as a coffee maker or washing machine). After a successful attack, compromised IoT devices are often used as relays for further attacks. For instance, IoT devices have been used in the past to build large-scale botnets such as Mirai or Bashlite [21, 18]. The malware targets unprotected IoT devices and turn them into bots. The attacker is then able to launch the actual attack (e.g., a distributed denial-of-service (DDoS) attack) by commanding all bots through a central Command-and-Control (C&C) server.

A well-known target of such an IoT-based DDoS attack was the website Krebs on Security[3]. According to Akamai (the digital security service provider of the website Krebs On Security), the DDoS attack was close to 620 Gbps (Gigabits of traffic per second). A second prominent victim of such an attack paradigm was the French WebHost and cloud service provider OHV[4], where the DDoS attack traffic peak using Mirai malware was 1.1 Tbps (Terabits of traffic per second). These massive attacks highlight the risks resulting from inadequate security mechanisms in IoT devices.

However, besides the missing patching ability and the ease with which the security mechanisms of IoT devices typically can be circumvented, the topology of networks comprising IoT network devices are often documented poorly [14]. Hence it is important to support the network forensic process to efficiently and effectively identify IoT devices in a network on base of their network traffic fingerprint. The findings of the survey [32] show that there is still a general lack of IoT forensics tools. The authors state that further research should focus on developing tools in IoT forensics to identify and acquire relevant IoT data.

In this paper, we show the efficiency and effectiveness of approximate matching to identify common IoT devices using their network captures. We present our algorithm `Cu-IoT`, which is an adapted version of the `mrsh-cf` algorithm [13] (the name `Cu` reminds on both the use of Cuckoo filters to represent the approximate hash and the task "See you" to find an IoT device). We show that `Cu-IoT` is superior to previous machine-learning approaches [1, 20], because it does not require feature extraction and a learning phase. Furthermore, `Cu-IoT` outperforms its hash-based competitor LSIF [8] which is based on the Nilsimsa hash in one out of three trials. We present an in-depth evaluation of `Cu-IoT` on three different datasets that include network traffic collected of a variety of different IoT devices. The captures include data of three different device states, specifically in their setup-phase [20], on an idle state [25], or on an active-state [9]. We achieve between 83% and almost 100% classification performance in terms of accuracy, recall, and precision, depending on the respective dataset. Our evaluation shows that the classification performance of `Cu-IoT` is at least as good as related work

[3] https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/
[4] https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/

algorithms without the computational overhead for feature extraction and model training typically associated with machine learning algorithms.

In detail the contributions of this paper are as follows:

1. Detailed presentation of our own approach `Cu-IoT`, an adapted version of the approximate matching algorithm `mrsh-cf`, to efficiently and effectively identify an IoT device based on the approximate hash of its network traffic capture.
2. Full publication of `Cu-IoT` including its source code to the digital forensics community via the website `github.com/dasec/Cu-IoT`.
3. Evaluation of `Cu-IoT` on three different IoT device traffic datasets containing many different IoT devices in different device states, such as setup-, idle-, and active-phase.
4. Comparison of `Cu-IoT` with its competitors from both the field of Locality-Sensitive Hashing, i.e., `LSIF` and `TLSH`, as well as from feature-extraction based approaches.

The remainder of this paper is organized as follows. Section 2 presents related work focusing on IoT device identification. Section 3 provides background information on approximate matching in general as well as information on the IoT device datasets used for our evaluation. Section 4 describes our selection process to find an appropriate classical approximate matching method for IoT device identification. Section 5 then shows how `Cu-IoT` works in detail. Moreover, this section presents the results of our experimental evaluation of `Cu-IoT` and provides a comparison with its competitors. Section 6 summarizes this paper and points to tasks for future work.

2 Related Work

In this section, we present related work to identify an IoT device based on its network capture. Identifying IoT devices based only on their MAC address and DHCP negotiation is an unreliable solution on a large scale, as stated by Sivanathan et al. [29], since these can be faked, spoofed, or changed easily. Therefore, many different approaches in natural language processing, multi-class machine learning classifiers, one-class classifiers, and neural networks have been published recently. We first turn to the class of machine learning-based approaches and then discuss an approximate hash-based method.

We first turn to machine learning approaches. Aksoy proposes in his Ph.D. thesis an IoT identification method called SysID [2], which was later published jointly with Gunes [1]. SysID can classify an IoT device using machine learning and genetic algorithms. SysID extracts features of submitted TCP/IP packet headers based on genetic algorithms and then applies machine learning algorithms (e.g., Decision Trees) to classify the device based on the protocol features selected by the genetic algorithm. Miettinen et al. published IoT Sentinel [20]. IoT Sentinel follows a similar approach to SysID, i.e. it uses packet headers for

identification and subsequent security measures of IoT devices. It identifies device types by its device signature using a machine learning-based classification model. Another interesting approach was published by Bezerra et al., who proposed the Internet of Things Detection System (IoTDS) [4], which generates a device signature by extracting features from the device's CPU utilization and temperature, memory consumption, and the number of running tasks, meaning that it does not make use of network traffic data. This approach was evaluated using four different one-class classification algorithms (Elliptic Envelope (EE), Isolation Forest, Local Outlier Factor, and One-class Support Vector Machine (OSVM)). Other approaches, like that of Dong et al. [10] and Bai et al. [3], use neural networks for IoT traffic fingerprinting. Unlike these supervised machine learning algorithms, our approach `Cu-IoT` does not require feature extraction, training, and multiple adjustments of a machine learning model, which typically requires expert knowledge and high computational power.

On the other hand, an algorithm similar to established approximate matching methods was used to identify IoT devices. Charyyev and Gunes introduced Locality-Sensitive IoT Fingerprinting (`LSIF`) [9], which is described as a framework and makes use of the Nilsimsa algorithm [30] for the detection of devices in networks. Nilsimsa is a Locality-Sensitive Hashing algorithm originally proposed for spam detection. In contrast to our algorithm `Cu-IoT`, LSIF is not publicly available. Nevertheless, based on the datasets used in [9] and public implementation of Nilsimsa, we can show that our approach is superior to LSIF concerning run time efficiency and detection performance, respectively. Furthermore, our algorithm comes with a publicly available implementation.

3 Approximate Matching and IoT Device Datasets

This section introduces approximate matching algorithms and then turns to IoT device datasets, which we will use for our evaluation.

3.1 Approximate Matching Algorithms

Approximate matching is called fuzzy hashing or similarity hashing, too. It has already been used in a variety of contexts, its baseline, however, is to identify a known digital artefact from a given dataset automatically. A typical use case is the matching of binary data, such as documents, executables, memory dumps and network traffic, against the filter of the approximate matching algorithm. For example, approximate matching has been used for file recognition [5], for malware detection [24] [16] and as a data loss prevention solution [12].

Compared to cryptographic hashes, fuzzy hashes are robust to changes in the input. While cryptographic hashes change entirely when a single bit is flipped (so-called avalanche effect), fuzzy hashes account for this change with a hash similar to the unchanged original.

Multi Resolution Similarity Hashing (`MRSH`) is a well-established 'classical' approximate matching algorithm. It comprises three steps: (i) selecting features

from the input, (ii) generating a digest, and (iii) comparing that digest with another. During the comparison in the third step `MRSH`, as well as other approximate matching algorithms, rely on a specific filter to look up familiar hashes. Several iterations of the original `MRSH` algorithm [26] have been published in the past, such as `mrsh-v2` [6], `mrsh-net` [7], and `mrsh-hbft` [17], all using different filters and the most efficient ones at the time of their release. The latest version of the `MRSH` algorithms is known as `mrsh-cf` [13]. It is equipped with a Cuckoo filter which is considered the fastest lookup filter and is superior to the Bloom filter of previous versions of the algorithm [11]. While other approximate matching algorithms are built to compare a specific file or data types, `MRSH` can compare data regardless of its context, but only based on its content at the byte-level, making it universally applicable.

Further, besides the previously mentioned MRSH family, other well-known approximate matching algorithms exist, such as the `sdhash` [27] algorithm and the de-facto standard algorithm `ssdeep` [15] which is known to be used on Google's VirusTotal platform. Trendmicro's `TLSH` [19] algorithm is also one of the Locality-Sensitive Hashing (LSH) algorithms, along with the `LSIF` algorithm mentioned above.

3.2 IoT Device Datasets

We evaluate our algorithm `Cu-IoT` on three different publicly available datasets containing 22 IoT devices that are on an active state [9], 31 IoT devices that are being set up [20], and 81 IoT devices that are on an idle state [25], respectively. All three IoT datasets originate from the network activities of various illumination devices, smart plugs, doorbells, cameras, coffeemakers, radios, TVs, smart speakers (e.g., Amazon Echo, or Google Home), and other smart home appliances. The same devices from all three datasets that we used for our research are shown in Table 1.

The traffic flow data of the first IoT dataset [9] was collected over 20 days, i.e., it contains measurements for each of the 22 devices over a period of 20 days. The data within this dataset represents network traffic collected when users actively interacted with the IoT devices. Charyyev and Gunes assembled this dataset for testing their IoT traffic flow identification approach using Locality-Sensitive Hashes.

The second IoT dataset [20] represents the traffic emitted during the initial setup phase of 31 smart home IoT devices in a network of 27 different device types (4 types are represented by two devices each) and different vendors (e.g., D-Link, Edimax Plug, Hue, TP-Link Plug, etc.). However, only 23 of these devices have at least 20 recorded traces from their setup phase available. Therefore, we used precisely these 23 devices in our research.

The third IoT dataset [25] consists of 81 smart home IoT devices that are in an idle state, i.e., when there is no interaction with the device. These IoT devices are deployed in two testbeds, one at the Northeastern University, US, and in the Imperial College London, UK. The dataset consists of traces for 55 devices each. For 26 devices, these traces are available twice. However, we use

Table 1. List of IoT devices in the three datasets used in our evaluation

#	**First Dataset [9] (*Active Data*)**	**Second Dataset [20] (*Setup Data*)**	**Third Dataset [25] (*Idle Data*)**
1	Chime_Doorbell	D-Link WiFi Day Camera DCS-930L	Allure Speaker with Alexa
2	D-Link_Cam936L	D-Link Door & Window sensor	Amazon Cloud Cam
3	Gosuna_LightBulb	D-Link Connected Home Hub DCH-G020	Amcrest Cam
4	Gosuna_Socket	D-Link HD IP Camera DCH-935L	Anova Sousvide
5	Goumia_Coffemaker	D-Link Smart plug DSP-W215	Apple TV
6	LaCrosse_AlarmClock	D-Link Water sensor DCH-S160	Behmor Brewer
7	Lumiman_Bulb600	D-Link Siren DCH-S220	Blink Cam
8	Lumiman_Bulb900	D-Link WiFi Motion sensor DCH-S150	Blink Hub
9	Lumiman_SmartPlug	Philips Hue Bridge model 3241312018	Bosiwo Cam
10	Minger_LightStrip	Philips Hue Light Switch PTM 215Z	D-Link Cam
11	Ocean_Radio	SmarterCoffee coffee machine SMC10-EU	D-Link Mov Sensor
12	Renpho_SmartPlug	Smarter iKettle 2.0 water kettle SMK20-EU	Echo Dot
13	Ring_Doorbell	TP-Link WiFi Smart plug HS110	Echo Plus
14	Smart_Lamp	TP-Link WiFi Smart plug HS100	Echo Spot
15	Smart_LightStrip	Edimax SP-1101W Smart Plug Switch	Fire TV
16	Tenvis_Cam	Edimax SP-2101W Smart Plug Switch	Flux Bulb
17	Wans_Cam	Fitbit Aria WiFi-enabled scale	GE Microwave
18	Wemo_SmartPlug	Homematic pluggable switch HMIP-PS	Google Home
19	itTiot_Cam	Osram Lightify Gateway	Google Home Mini
20	oossxx_SmartPlug	Ednet.living Starter kit power Gateway	Honeywell T-stat
21	tp-link_LightBulb	MAX! Cube LAN Gateway for MAX! Home automation sensors	Insteon Smart Hub
22	tp-link_SmartPlug	WeMo Link Lighting Bridge model F7C031vf	Invoke Speaker with Cortana
23		Withings Wireless Scale WS-30	Lefun Cam
24			LG TV
25			Lightify Smart Hub
26			Luohe Cam
27			Magichome Strip
28			Microseven Cam
29			Nest T-stat
30			Netatmo Weather
31			Philips Bulb
32			Philips Hue Smart Hub
33			Ring Doorbell
34			Roku TV
35			Samsung Dryer
36			Samsung Fridge
37			Samsung TV
38			Samsung Washer
39			Sengled Smart Hub
40			Smarter Brewer
41			Smarter iKettle
42			Smartthings Smart Hub
43			TP-Link Bulb
44			TP-Link Plug
45			TP-Link Plug 2
46			Wansview Cam
47			WeMo Plug
48			WiMaker Spy Camera
49			Wink 2 Smart Hub
50			Xiaomi Cam
51			Xiaomi Cleaner
52			Xiaomi Smart Hub
53			Xiaomi Rice Cooker
54			Xiaomi Strip
55			Yi Cam
56			ZModo Doorbell

them only once because using both capture sets would bias the results in favor of the 26 duplicate devices. The captures of the devices on idle state cover an average of 8 hours per night for one week for both labs, i.e., 112 hours in total of idle experiments.

4 Suitable Approximate Matching Algorithms

This section shows our selection process to find the best suitable classical approximate matching method for IoT device identification. We compare the `mrsh` approximate matching algorithm using three different representations of the approximate hashes (Bloom filter, Cuckoo filter, Hierarchical Bloom filter), `ssdeep`, and `TLSH`.

As a first general test, which approximate matching algorithm shows a promising performance, we used the so-called All-vs-All test, which sets a baseline for the algorithm's performance with data-at-rest. Every examined algorithm generates its filter of the well-known *t5-corpus* [28] in a first step. In a second step, the algorithm with this filter is given the complete *t5-corpus* (1.8 GB). The time it takes for every algorithm to generate the filter and to apply it onto every file in the corpus is shown in Table 2.

We assume that speed is a key indicator of the suitability of an algorithm for the quick identification of a specific IoT device in a large network. This is why we tested five of the prevalent algorithms for their performance when matching the *t5-corpus* with itself. Due to its good performance, we chose `mrsh-cf` for our further evaluation steps. Another candidate that performed well in our All-vs-All test is `ssdeep`. However, it was already shown that this approximate matching algorithm does not perform good with small fragments, i.e., it only performs well when fragments contain at least 25% - 50% of the original file [22], which is why we did not consider `ssdeep` further. To analyze not only an algorithm of the MRSH family but also a maintained, efficient and optimized algorithm of the LSH family, we also consider the `TLSH` algorithm in our further evaluation. Further, to be able to compare the performance of these algorithms with the results of our competitor `LSIF`, which is based on `Nilsimsa`, we also use the `Nilsimsa` algorithm in our evaluation.

Table 2. Time for filter generation and application

	`mrsh-cf`	`mrsh-net`	`mrsh-hbft`	`ssdeep`	`TLSH`
Filter Gen. (in sec)	12.51	32.90	274	14.90	17.18
All-vs-All (in sec)	12.94	67.84	300	27.37	78.29

It is important to understand that `TLSH` is one of the best performing algorithms for similarity hashing out-of-the-box, but it has a certain limitation in the input of data to be compared. As far as we know, the algorithm can only compare "1 to n" but not "n to n" efficiently. A "1 to n" test with `TLSH` has

to be through comparing the "1" consecutively with every "n," which means a slight loss in performance compared to the other algorithms. This is why the All-vs-All test in Table 2 is slightly slower for `TLSH`. In detail, this means you can give the algorithm the hashes of several files to compare them with one file. Also, reverse order works, i.e., compare the hash of one file vs. multiple files. Furthermore, it is possible to compare all files of a folder, each with each, by the algorithm. However, it is not intuitively possible to give the algorithm the hash values of several files and compare them with several other files unless you do this outside the algorithm code in a script (as was done in our **All-vs-All** test in Table 2) and compares each file with the algorithm's filter. However, as we will see in our further evaluations, `TLSH` is still a valid option for hashing IoT traces.

For a rough estimation of the performance of the most promising algorithms (`mrsh-cf`, `TLSH`, and `Nilsimsa`) with IoT device data, we performed a trivial test. We tested the algorithms' performance for the simple task of hashing a pcap-file and comparing it to itself. Note that Charyyev and Gunes [8] evaluated their `LSIF` method against `TLSH` in greater detail, but since `LSIF` is not open source we relied on a well documented Java version of the `Nilsimsa` algorithm for our initial performance tests[5]. The size of the input trace file used in our first test was 307.4 KB.

Table 3. Naive benchmark for IoT device network capture

	Hashing (in ms)	**Comparison (in ms)**
`mrsh-cf`	43	22
`Nilsimsa`	7150	2380
`TLSH`	14	3

Table 3 shows that `TLSH` performs best in this limited scenario. The algorithm's codebase is well maintained due to its use in commercial products, such as Google's VirusTotal, which accounts for its fast execution. `mrsh-cf` is positioned in the midfield, whereas `Nilsimsa` takes a comparatively long time to hash and compare. It is important to note that `Nilsimsa` examines strings for their similarity, and input must first be converted into string form, whereas the other two algorithms can perform direct-byte-wise comparisons. This overhead is, of course, not included in our test. Given the possible use of the algorithm in a highly automated network scenario, this point should be considered.

For further understanding, it is essential to know that `Cu-IoT` is fundamentally different from the other two algorithms in terms of its recognition of the difference between input and filter. Compared to other similarity hashing algorithms, both the `mrsh-cf` and the `Cu-IoT` algorithm based on it do not have a static similarity score. The result of a hash comparison performed by `mrsh-cf` is a comparison of the total chunks, that the input item has, and the number of

[5] https://github.com/weblyzard/nilsimsa

chunks that where detected. This means that the results have to be interpreted as relational matching results. The file that was recognized for the most part also matches the previously unidentified input file with the utmost certainty.

Table 4. Performance metrics of `mrsh-cf` compared to `Nilsimsa` and `TLSH`

	Model Size	**Feature Size**	**Response Time**	**Processing Speed**
`mrsh-cf`	16.8 MB	8.4 KB	94 ms	7.467 x 10^8 bits/s
`Nilsimsa`	8.24 MB	4.12 KB	112.0 ms	5.886 x 10^8 bits/s
`TLSH`	258 KB	0.129 KB	57 ms	0.3621 x 10^8 bits/s

The results of a second, more reliable test, are summarized in Table 4, where the processing costs for `mrsh-cf` compared to `Nilsimsa` (on which `LSIF` is based) are shown with regards to the model size (size of the signature database), feature size (size of the one hash generated from flow), the response time (the time required to identify the flow), and processing speed (speed of generating the digest of the flow). In this test, we assume that the filter consists of 20 devices with 100 x 10-minute traces per device. Table 4 shows that `mrsh-cf` works more efficiently than its competitors. Crucial for the efficiency of any matching process is the underlying lookup mechanism. Assuming that `LSIF` works with a database that is not particularly designed for lookup-efficiency, the time efficiency will be linear ($O(n)$), while Cuckoo filters have a time efficiency for this operation of $O(1)$. `TLSH` is very space-efficient, so its model size is only a fraction of that of its competitors. `TLSH`'s response time might be faster, but in terms of processing speed, `mrsh-cf` takes the lead. Based on `mrsh-cf`'s good performance and its flexibility, we chose to use it as the basis of our approach to IoT device fingerprinting, namely `Cu-IoT`.

5 Evaluation

In this section, we present our evaluation methodology as well as our evaluation setup for the three different datasets and show how `Cu-IoT` works in detail on our setup and the datasets. Moreover, this section presents the results of our experimental evaluation of `Cu-IoT` and provides a comparison with its competitors `TLSH` and `LSIF`.

5.1 Evaluation Setup of the First and Third Dataset

The first (*Active Data*) and third (*Idle Data*) dataset consist of relatively large device records that vary in size but were all collected over a more extended period compared to the second (*Setup Data*) dataset. As already mentioned in section 3.2, the first dataset represents 20 days, and the third dataset represents approximately 2.33 days overall. However, the second dataset represents only a

shorter time interval, namely the setup phase of each device, so the filter for the second dataset must be different, as shown in section 5.2.

For the first and third datasets, we divide all traces into 10-minute segments. One hundred of these segments are randomly selected and form the filter for a device, which is used to find the remaining traces of the device among all the others. The traces are ranked according to their detected proportion. For the first dataset, the apparatus for which a higher relative proportion was detected are ranked higher. This behavior is shown in Figure 1. For the third dataset, the highest-ranking is given to those files from which the most "chunks" were found regardless of how much of the total trace this represents. The different approaches yield better results with the respective data (operational data - relational ranking; idle data - non-relational ranking).

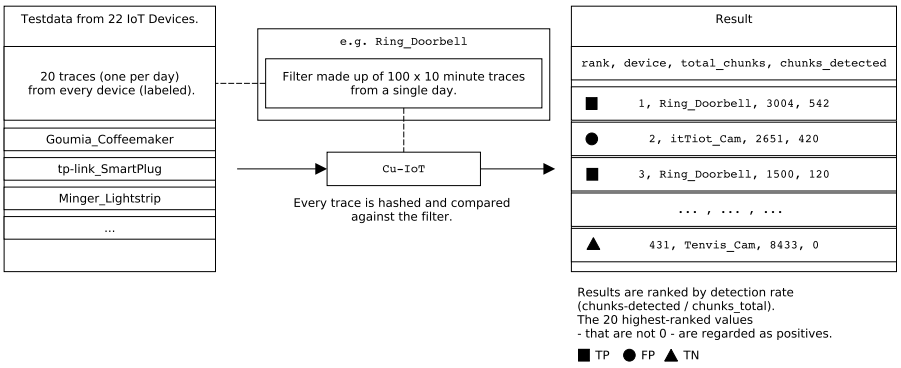

Fig. 1. Evaluation setup of the first and third dataset: Testrun with `Ring_Doorbell` as filter device. For the third dataset the calculation `(chunks detected / chunks total)` is ignored and results are ranked according to how much of their chunks were recognized overall.

5.2 Evaluation Setup of the Second Dataset

In contrast to the first (*Active Data*) and the third (*Idle Data*) dataset, the second dataset (*Setup Data*) consists of comparatively little data from the actual lifecycle of IoT devices. Since we are working with network captures of relatively similar but short duration, namely those of the setup phase, the tests with the algorithms on these data must also be handled differently than on the first and third datasets. For each device, 20 setup phases were recorded. One of them (i.e., 5% of the total data) serves as a filter to identify the other 19 among the traces of other devices. The results are ranked according to the devices from which the most chunks were found, and thus the same evaluation methodology is used for the third dataset. The top 19 traces are considered positives (either

true positives or false positives), and the rest are considered negatives (either true negatives or false negatives).

5.3 Evaluation Methodology

In the following, we discuss the results of the devices traffic matching using `Cu-IoT` on three different datasets. We compare the results of `Cu-IoT` with the results of `LSIF` and `TLSH` for each of the three IoT datasets in Table 5, 6, 7, respectively. It is important to mention that based only on the information given in the paper by Charyyev and Gunes [8], it is not clear which exact implementation of the `Nilsimsa` algorithm was used for their `LSIF` approach. However we were in close contact with the original authors of `LSIF` and were able to rebuild Charyyev's algorithms in the Go programming language. Therefore, with the only exception of the `TLSH` values in Table 7[6], we were able to do our own measurements using our own implementation of `LSIF` using `Nilsimsa` and our own `TLSH` implementation. These measurements helped us to conduct fair comparisons with our new algorithm `Cu-IoT`. The exact classification performance measurements can be found in Table 5, 6, 7. In addition, the source code can be found and verified in the previously mentioned GitHub repository.

For every dataset we measured the classification performance in terms of Precision, Recall, F1-score, Accuracy, Specificity, AUC, True-Positive Rate (TPR), False-Positive Rate (FPR), True-Negative Rate (TNR), and False-Negative Rate (FNR). The exact meaning of these metrics in relation to our evaluation setup, as well as those of a True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN), are explained as follows:

- **Precision**: $\frac{TP}{TP+FP}$
- **Recall**: $\frac{TP}{TP+FN}$
- **F1-score**: $\frac{2}{\frac{1}{Precision}+\frac{1}{Recall}}$
- **Accuracy**: $\frac{TP+TN}{TP+FP+TN+FN}$
- **Specificity**: $\frac{TN}{FP+TN}$
- **AUC**: $\frac{1}{2} \cdot (\frac{TP}{TP+FN} + \frac{TN}{TN+FP})$
- **True Positive (TP)**: Is a trace in the top 20 ranked traces that belongs to the same device that is used for the filter.
- **False Positive (FP)**: Is a trace in the top 20 ranked traces that does not belong to the same device that is used for the filter.
- **True Negative (TN)**: Is a trace that is not ranked in the top 20 traces that does not belong to the same device that is used for the filter.
- **False Negative (FN)**: Is a trace that is not ranked in the top 20 traces that belongs to the same device that is used for the filter
- **TPR**: $\frac{TP}{TP+FN}$
- **FPR**: $\frac{FP}{FP+TN}$
- **TNR**: $\frac{TN}{TN+FP}$
- **FNR**: $\frac{FN}{FN+FP}$

[6] Since in the case of `TLSH` on *Idle Data* without the original source code, we were not able to rebuild the data preprocessing and had to rely on the existing measurements.

5.4 Evaluation Results on Active Data

For the first dataset, i.e., based on the *active data*, in Table 5 we can see the slightly better performance of `Cu-IoT` compared to `LSIF` in almost all metrics. How much of a device's traces were matched on average given a filter from a device (a) or (b) (while (a) stands for the same device, or (b) stands for a different one) are presented in Figure 2. Overall `Cu-IoT` matches the correct traces given a filter of the same device very accurately. However, it is noteworthy that devices, that emit little data during their active phase (e.g., *Goumia_Coffeemaker* or *tpLink_LightBulb*), still are well detected by `Cu-IoT`. This clearly distinguishes it from its competitor `LSIF`, where the authors claim that simple traffic flows may hurt the classification performance of `LSIF` [9]. Figure 2 represents the average recognition of a specific device given a filter from the same or another device. As we can see, the traces from the devices *Goumia_Coffeemaker* and *tpLink_LightBulb* are usually only detected to a small extent, but on average still identified correctly. The traces of these two devices are probably never completely recognized, as is the case with other devices with a low similarity score, too. Meaning that a trace of these two devices is never fully recognized in its entirety. Nevertheless, it is always correctly identified as belonging to the correct device. What we try to illustrate with this example is that `Cu-IoT` does not need to recognize a trace in its entirety to connect it to the correct device.

While the first dataset is quite heterogeneous and we achieve overall an average precision, recall, and accuracy of 97.5%, 98.4%, and 99.8% on *Active Data*, respectively, it remains to be validated how well the identification works on a more homogeneous dataset. However, in a larger dataset with more devices from the same vendor (as it is the case in the second dataset), which might also rely on similar transmission protocols, these devices might become indistinguishable for `Cu-IoT`. We will examine the algorithm's behaviour on such a homogeneous dataset in section 5.5.

Table 5. Average evaluation results of `Cu-IoT` and `TLSH` compared to `LSIF` for the first dataset (*Active Data*)

	Precision	**Recall**	**F1-Score**	**Accuracy**	**Specificity**	**AUC**	**TPR**	**FPR**	**TNR**	**FNR**
`Cu-IoT`	97.5%	98.4%	97.9%	99.8%	99.9%	95.5%	98.4%	0.16%	99.8%	1.5%
`LSIF`	92.4%	90.8%	92.1%	99.8%	98.5%	95.2%	90.0%	0.03%	99.6%	0.1%
`TLSH`	90.1%	85.3%	85.2%	99.0%	99.2%	92.1%	84.8%	0.03%	99.6%	1.5%

5.5 Evaluation Results on Setup Data

We now evaluate the algorithms behavior in case of the *setup data*, i.e., on traffic captures of IoT devices performing their setup phase. For each device, we have 20

[8] Relational similarity score on a scale from 0 to 100 (0 means that nothing was found and 100 means that the entire trace was found).

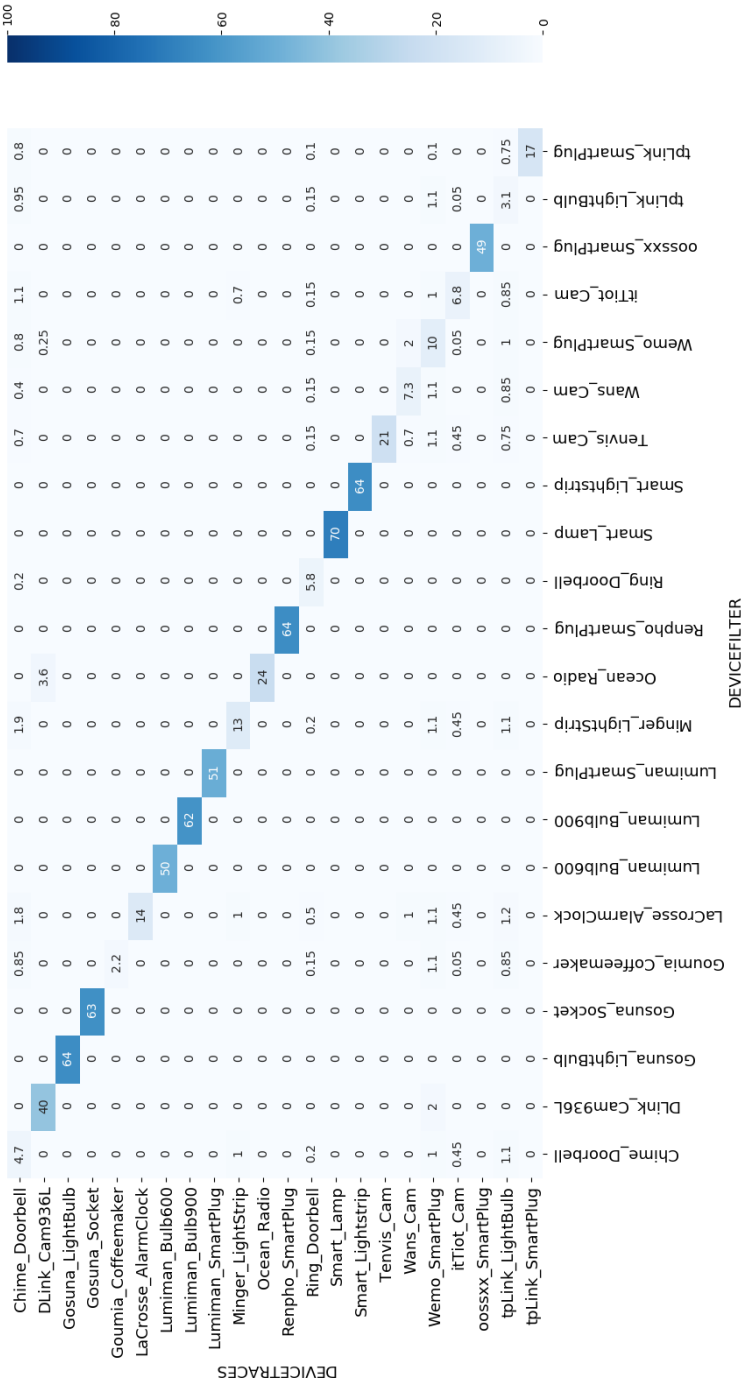

Fig. 2. Average similarity scores calculated by `Cu-IoT` for the 22 IoT devices of the first dataset (*Active Data*)[8]

traces that represent the devices setup phase. The evaluation was done by using one measurement as the filter for the device and the other 19 traces together with all the traces from all the other devices form the test data so that the filter represents only 5% of the overall data from a respective device, and we have no bias with respect to a specific device. Our measurements for `Cu-IoT` are shown in Table 6 together with the corresponding measurements for `LSIF` by Charyyev and Gunes. However, it should be mentioned that according to [8], the filter for their `LSIF` algorithm consists of 14 traces instead of 1 (as in our case) for each device. Therefore, their filter represents 70% of the overall data for a single device, which is a huge difference from our approach. The reason we still chose to built the filter from a single trace rather than from 14 traces is that we can compare our results to those of feature extraction-based methods in terms of accuracy, as is depicted in Figure 5. However, despite these serious differences in the structure of the filter, our measurements for `Cu-IoT` and `TLSH` still hold up very well against `LSIF`. Overall the average precision, recall, and accuracy of `Cu-IoT` on *Setup Data* is 83.3%, 83.9%, and 98.6%, respectively.

Table 6. Average evaluation results of `Cu-IoT` compared to `LSIF` for the second dataset (*Setup Data*)

	Precision	**Recall**	**F1-Score**	**Accuracy**	**Specificity**	**AUC**	**TPR**	**FPR**	**TNR**	**FNR**
`Cu-IoT`	83.3%	83.9%	83.6%	98.6%	99.2%	91.6%	81.5%	0.8%	99.0%	18.5%
`LSIF`	80.2%	79.9%	80.5%	97.6%	99.1%	89.3%	87.9%	0.1%	98.0%	20.1%
`TLSH`	80.8%	80.8%	80.8%	98.5%	99.2%	89.9%	80.8%	0.9%	99.4%	19.3%

Figure 3 shows the average similarity score assigned by `Cu-IoT` to all devices, i.e. we can see the average matching results of the device traces given a certain filter. Figure 4 represents the confusions between filter-device and input-device. Especially the high confusion rate for devices from the vendor D-Link is striking. This is due to the number of similarities in the setup protocols of those devices. While for some devices, the setup phase might accumulate only a few kilobytes, for others (especially those from the vendor D-Link) the setup phase might produce a few hundred kilobytes of traces. However, suppose we pay attention to the highest similarity scores per device. In that case, almost all of them are correctly identified on average, with the only exception of the *D-LinkSwitch*, which was confused with the devices *D-LinkSiren* and *D-LinkWaterSensor*, which have a higher similarity score.

Please note, that the results in Figure 3 cannot be derived directly from the values in Table 6. The table shows how many traces of the devices were detected correctly on average, while the graph shows how much of them was detected on average.

[10] Relational similarity score on a scale from 0 to 100 (0 means that nothing was found and 100 means that the entire trace was found).

[12] See footnote 10.

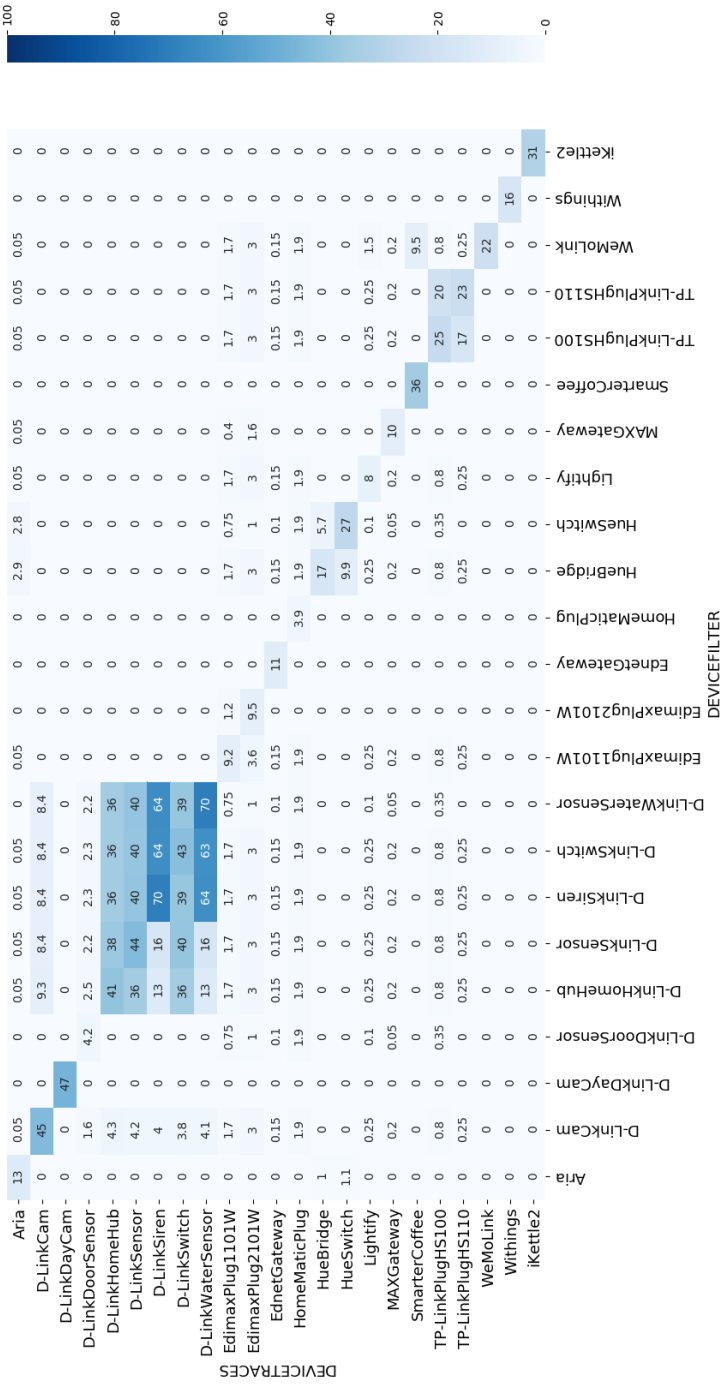

Fig. 3. Average similarity scores calculated by `Cu-IoT` for the 23 IoT devices of the second dataset (*Setup Data*)[10]

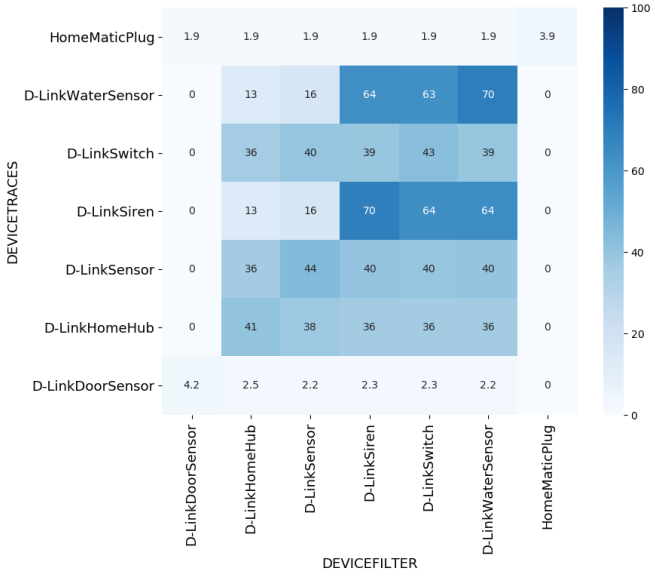

Fig. 4. Average `Cu-IoT` similarity scores for IoT devices in the second dataset (*Setup Data*) that were most often misidentified[12]

Figure 5 shows the relation of `Cu-IoT`'s device matching results compared to the average comparison results that were achieved using the machine learning approaches Elliptic Envelope (EE) and One-class Support Vector Machine (OSVM) [4], SysID [1], IoT Sentinel [20], as well as LSIF [9]. The bars each represent the accuracy of the different algorithms. `Cu-IoT` revealed itself to have much better accuracy on several occasions than feature extraction-based approaches. Most notably is that `Cu-IoT` performed very well at classifying devices such as *SmarterCoffee* and Smarter *iKettle2*. These devices have very short setup phases and therefore are harder to classify for most feature extraction-based approaches except SysID [1], as previous research on the same dataset already made clear [9]. `Cu-IoT` seems to be a preferable solution in scenarios with minimal inputs. All setup phases of devices of the vendor D-Link are notably long, which is why the traces of these devices are also the largest ones in the dataset. As Figure 4 already showed, these are the devices that were most often misidentified by the `Cu-IoT` algorithm. So we conclude that the increased search space compensates for the detection advantage the `Cu-IoT` algorithm has at smaller traces. Larger traces seem to be easier to match using feature extraction-based approaches.

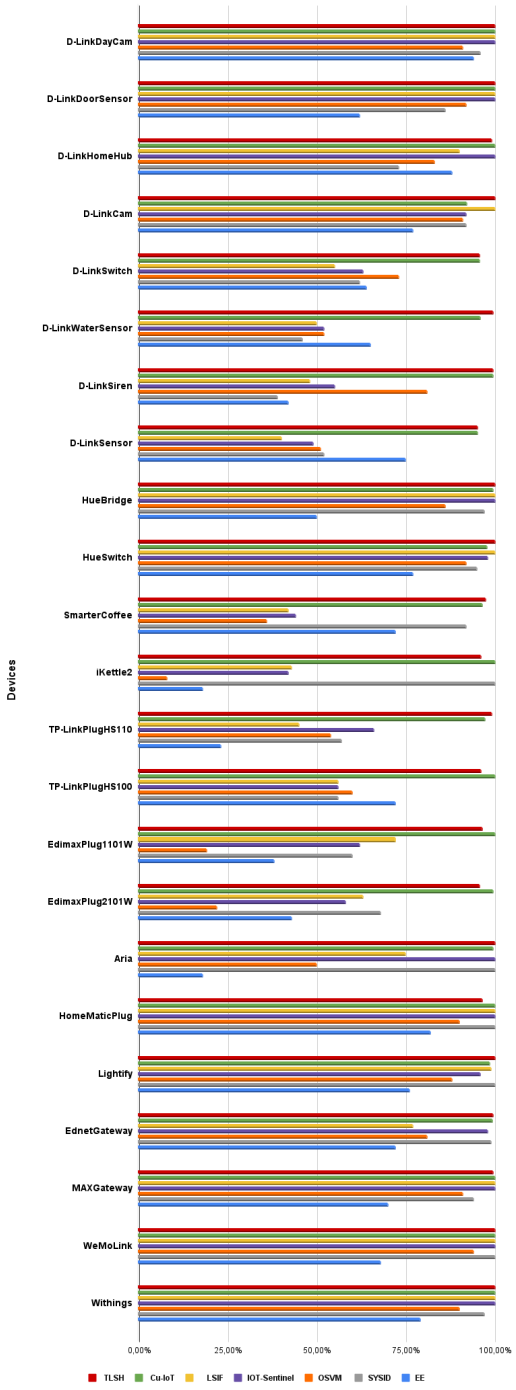

Fig. 5. Accuracy achieved per device on the second dataset (*Setup Data)*. The metrics for `EE`, `OSVM` and `SYSID` in this chart are taken from the work of Charyyev et al. [9]. All other metrics are based on the measurements of our own algorithms.

5.6 Evaluation Results on Idle Data

The third dataset consists only of devices in an idle state and thus is isolated from human interactions. This test was performed similarly to our first test. For each device, there exist about 56 hours of recorded traffic. Of these, 100 x 10 minutes were taken for the respective filter of an algorithm, and then the remaining capture, together with all captures of the other devices, were evaluated as a test environment. We will now look into the peculiarities of this particular test. Again, Table 7 shows the classification performance of `Cu-IoT` in comparison with `TLSH` and `LSIF` on this dataset.

Important for the understanding of the results is that the cited results from Charyyev and Gunes [8] is based on the data of 55 unique devices, but 56 device results were evaluated. In fact, for the *TP-Link Plug* there exist two different recordings. To be able to compare our results again with those of Charyyev and Gunes, these two different recordings were also included in our evaluation of `Cu-IoT`. The special thing about the data that IoT devices produce in their idle state is that they can be very different in size. For example, on the one hand, a device like *D-Link Camera* (a surveillance camera) produces only a few bytes of network data within 56 hours. On the other hand, a device like the *Wansview Webcam* produces several Megabytes of network data at the same time. The transmission contents during the idle stage mostly consist of simple heartbeats or update checks that are managed with the same protocols. Compared to the first two datasets, the test data is even more homogeneous, which also increases the chance of confusion. In this dataset, there is much confusion between devices from the same manufacturer (remember, in the second dataset, devices from D-Link were difficult to distinguish) and between devices from different manufacturers and for different purposes. As can be seen in Table 7, `Cu-IoT` performs slightly worse than `LSIF` but still better than `TLSH`. Overall the average precision, recall, and accuracy of `Cu-IoT` on *Idle Data* is 90.1%, 90.0%, and 99.8%, respectively.

Table 7. Average evaluation results of `Cu-IoT` compared to `LSIF` for the third dataset (*Idle Data*)

	Precision	**Recall**	**F1-Score**	**Accuracy**	**Specificity**	**AUC**	**TPR**	**FPR**	**TNR**	**FNR**
`Cu-IoT`	90.1%	90.0%	97.2%	99.8%	99.9%	98.3%	89.9%	0.1%	99.0%	9.7%
`LSIF`	91.5%	92.0%	91.1%	99.8%	99.2%	97.4%	95.3%	0.9%	99.0%	4.1%
`TLSH`[13]	83%	78%	75%	99%	100%	89%	-	-	-	-

6 Conclusion and Future Work

In this paper, we introduced a novel IoT device identification method using a re-engineered `mrsh-cf` algorithm – called `Cu-IoT` – that can be applied on

[13] In contrast to the other datasets, in the case of *Idle Data* the `TLSH` results are taken from [8] and must be considered unverified.

arbitrary IoT devices network captures. Unlike other existing approaches, our approach uses approximate matching and therefore does not require multiple iterations of feature extraction from traffic, tuning of model parameters, and retraining of the model. To the best of our knowledge, during the time of writing, `Cu-IoT` shares these benefits with only two other algorithms, which both rely on Locality-Sensitive Hashing instead of Multi-Resolution Similarity Hashing. Our evaluations have shown that `Cu-IoT` performs significantly better than its competitors on *active data* with a precision and recall of 97.5% and 98.4%, respectively (as shown in section 5.4), and reaches slightly better precision (83.3%) and higher recall (83.9 %) on *setup data* (as shown in section 5.5). However, on *idle data* (as shown in section 5.6), `Cu-IoT` performs slightly worse than `LSIF` in terms of precision (89.7% vs. 94%) and recall (89.5% vs. 93%), and slightly better in terms of accuracy (99.8% vs. 99%). All in all, `Cu-IoT` can keep up well with `LSIF`, while the latter - and thus also `Cu-IoT` - is competitive with typical machine learning approaches. Our work showed for the first time that a well-established 'classical' approximate matching algorithm applies to the task of IoT device identification. This was validated using three different data sets consisting of many different IoT devices. Therefore, the publicly available `Cu-IoT` algorithm is capable of supporting the network forensics process to efficiently and effectively identify IoT devices in a network during an incident. Since IoT devices pose a poor degree of security, tools like `Cu-IoT`, that focus on IoT forensics, in particular, will become increasingly important in the future.

As future work, on the one hand, we are optimistic that we can still improve our results in our finished work, and on the other hand, we still need to verify what and how exactly the results were obtained with `LSIF`. Until the final version of our paper is due, we will verify the results obtained with `LSIF` when the algorithm is made available to us. Furthermore, separate measurements for `TLSH` are performed in case of *active data* and *idle data*. We will further elaborate on how the composition of network traffic in different device states affects the identification process. In addition to the classification metrics of the three algorithms presented so far, we will also provide similar data based on our measurements for the feature extraction based approaches, such as `SySID` and `IoT Sentinel`, on all three datasets to allow a direct comparison of the feature extraction algorithms with our approximate matching approach on all datasets - not only in the case of the *setup data*.

We will further look into things like cross-testbed identification, since there are standard devices in the datasets used, and the issue of device confusion in case of the same vendor, as it is manifested in Subsection 5.2, through the means of common block elimination. This technique could potentially benefit the precision and recall of the `Cu-IoT` with significant homogeneous traces originating from devices with very similar protocols. As was shown in section 5, `Cu-IoT` performs well with small heterogeneous device traces but struggles with larger, more homogeneous ones. Through most common-block elimination, the larger traces can be reduced to smaller ones that could be easier to recognize.

In the future, we would like to perform the analysis using `Cu-IoT` on further, preferably larger, IoT datasets and examine the applicability of other prominent approximate matching algorithms for network device identification. Additionally, we want to analyze how to approximate matching can be used to detect anomalies in the behavior of IoT devices and thus prevent prevalent attacks such as botnets or DDoS. It is feasible to extend `Cu-IoT` to reliably detect such anomalies since the signature generated by an anomalous traffic flow significantly differs from the signature of the benign traffic stored in its filter.

References

1. A. Aksoy and M. H. Gunes, "Automated IoT device identification using network traffic," in Proc. IEEE Int. Conf. Commun. (ICC), Shanghai, China, 2019, pp. 1–7.
2. A. Aksoy, "Network Traffic Fingerprinting using Machine Learning and Evolutionary Computing Automated IoT device identification using network traffic," PhD thesis, University of Nevada, 2019.
3. Bai, L., Yao, L., Kanhere, S. S., Wang, X., Yang, Z. (2018, October). Automatic device classification from network traffic streams of internet of things. In 2018 IEEE 43rd conference on local computer networks (LCN) (pp. 1-9). IEEE.
4. Bezerra, V.H.; da Costa, V.G.T.; Barbon Junior, S.; Miani, R.S.; Zarpelão, B.B. IoTDS: A One-Class Classification Approach to Detect Botnets in Internet of Things Devices. Sensors 2019, 19, 3188. https://doi.org/10.3390/s19143188
5. P. Bjelland, K. Franke and A. Arnes, Practical use of approximate hash-based matching in digital investigations, Digital Investigation, vol. 11(S1), pp. 18—26, 2014.
6. F. Breitinger and H. Baier, Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2, in *Digital Forensics and Cyber Crime*, M. Rogers and K. C. Seigfried-Spellar (Eds.), Springer Berlin Heidelberg, pp. 167–182, 2013.
7. F. Breitinger, I. Baggili, File Detection on Network Traffic Using Approximate Matching, *Journal of Digital Forensics, Security and Law*, vol. 9(2), pp. 23–36, 2014.
8. B. Charyyev and M. H. Gunes, "Locality-Sensitive IoT Network Traffic Fingerprinting for Device Identification," in IEEE Internet of Things Journal, vol. 8, no. 3, pp. 1272-1281, 1 Feb.1, 2021, doi: 10.1109/JIOT.2020.3035087.
9. B. Charyyev and M. H. Gunes, "IoT Traffic Flow Identification using Locality Sensitive Hashes," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1-6, doi: 10.1109/ICC40277.2020.9148743.
10. Dong, S., Li, Z., Tang, D., Chen, J., Sun, M., Zhang, K. (2019). Your smart home can't keep a secret: towards automated fingerprinting of IoT traffic with neural networks. arXiv preprint arXiv:1909.00104.
11. Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT '14). Association for Computing Machinery, New York, NY, USA, 75–88. DOI:https://doi.org/10.1145/2674005.2674994
12. T. Göbel, F. Uhlig, H. Baier. "Empirical Evaluation of Network Traffic Analysis using Approximate Matching Algorithms". In Advances in Digital Forensics XVII, Springer, Cham, 2021.

13. V. Gupta and F. Breitinger, How Cuckoo Filter Can Improve Existing Approximate Matching Techniques, in *Digital Forensics and Cyber Crime*, J. I. James, F. Breitinger (Eds.), Springer International Publishing, Cham, pp. 39–52, 2015.
14. M. M. Hossain, M. Fotouhi and R. Hasan, "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things," 2015 IEEE World Congress on Services, 2015, pp. 21-28, doi: 10.1109/SERVICES.2015.12.
15. J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, *Proceedings of the Sixth Annual Digital Forensic Research Workshop*, vol. 3, pp. 91–97, 2006
16. L. Liebler, H. Baier. "Towards exact and inexact approximate matching of executable binaries." Digital Investigation, vol. 28, pp. 12–21, 2019.
17. D. Lillis, F. Breitinger and M. Scanlon, Expediting MRSH-v2 Approximate Matching with Hierarchical Bloom Filter Trees, in *Digital Forensics and Cyber Crime*, P. Matoušek, M. Schmiedecker (Eds.), Springer International Publishing, Cham, pp. 144–157, 2018.
18. A. Marzano et al., "The Evolution of Bashlite and Mirai IoT Botnets," 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 00813-00818, doi: 10.1109/ISCC.2018.8538636.
19. J. Oliver, C. Cheng and Y. Chen, TLSH – A Locality Sensitive Hash, *Proceedings of the Fourth Cybercrime and Trustworthy Computing Workshop*, pp. 7–13, 2013.
20. M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi and S. Tarkoma, "IoT SENTINEL: Automated device-type identification for security enforcement in IoT", Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. Workshops (ICDCS), pp. 2177-2184, 2017.
21. C. Kolias, G. Kambourakis, A. Stavrou and J. Voas, "DDoS in the IoT: Mirai and other botnets", Computer, vol. 50, no. 7, pp. 80-84, 2017.
22. A. Lee and T. Atkison, A Comparison of Fuzzy Hashes: Evaluation, Guidelines, and Future Suggestions, *Proceedings of the SouthEast Conference*, pp. 18–25, 2017.
23. Meidan, Yair, et al. "N-baiot—network-based detection of iot botnet attacks using deep autoencoders". IEEE Pervasive Computing 17.3 (2018), pp. 12-22, doi: 10.1109/MPRV.2018.03367731
24. F. Pagani, M. Dell'Amico, D. Balzarotti. "Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis". *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, ACM, pp. 354-365, 2018.
25. Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In Proceedings of the Internet Measurement Conference (IMC '19). Association for Computing Machinery, New York, NY, USA, 267–279. DOI:https://doi.org/10.1145/3355369.3355577
26. V. Roussev, G. G. Richard and L. Marziale, Multi-resolution similarity hashing, *Digital Investigation*, vol. 4, pp. 105–113, 2007.
27. V. Roussev, Data fingerprinting with similarity digest, in *Advances in Digital Forensics VI*, K.-P. Chow, S. Shenoi (Eds.), Springer Berlin Heidelberg, Germany, pp. 207–226, 2010.
28. V. Roussev, An evaluation of forensic similarity hashes, *Digital Investigation*, vol. 8, pp. 34–41, 2011.
29. Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., Sivaraman, V. (2018). Classifying IoT devices in smart environments us-

ing network traffic characteristics. IEEE Transactions on Mobile Computing, 18(8), 1745-1759.
30. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi , P. Samarati, An Open Digest-based Technique for Spam Detection, *Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing Systems*, pp. 559–564, 2004.
31. Shwartz, O., Mathov, Y., Bohadana, M., Elovici, Y., Oren, Y. (2017, November). Opening Pandora's box: effective techniques for reverse engineering IoT devices. In International Conference on Smart Card Research and Advanced Applications (pp. 1-21). Springer, Cham.
32. Tina Wu, Frank Breitinger, and Ibrahim Baggili. 2019. IoT Ignorance is Digital Forensics Research Bliss: A Survey to Understand IoT Forensics Definitions, Challenges and Future Research Directions. In Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19). Association for Computing Machinery, New York, NY, USA, Article 46, 1–15. DOI:https://doi.org/10.1145/3339252.3340504