Identifying Devices of the Internet of Things using Machine Learning on Clock Characteristics

Pascal Oser[1,2(✉)] [0000−0002−2461−343X], Frank Kargl[2] [0000−0003−3800−8369], and Stefan Lüders[1] [0000−0001−8676−2353]

[1] European Organization for Nuclear Research CERN, Geneva 1211, Switzerland
{p.oser,stefan.lueders}@cern.ch
[2] Ulm University, Ulm 89081, Germany
frank.kargl@uni-ulm.de

Abstract. The number of devices of the so-called Internet of Things (IoT) is heavily increasing. One of the main challenges for operators of large networks is to autonomously and automatically identify any IoT device within the network for the sake of computer security and, subsequently, being able to better protect and secure those.
In this paper, we propose a novel approach to identify IoT devices based on the unchangeable IoT hardware setup through device specific clock behavior. One feature we use is the unavoidable fact that clocks experience "clock skew", which results in running faster or slower than an exact clock. Clock skew along with twelve other clock related features are suitable for our approach, because we can measure these features remotely through TCP timestamps which many devices can add to their packets. We show that we are able to distinguish device models by Machine Learning only using these clock characteristics. We ensure that measurements of our approach do not stress a device or causes fault states at any time.
We evaluated our approach in a large-scale real-world installation at the European Organization for Nuclear Research (CERN) and show that the above-mentioned methods let us identify IoT device models within the network.

Keywords: Internet of Things; Identification; Security; Clock Characteristics; Machine Learning.

1 Introduction

A problem we face today is that the number of embedded devices is exploding and therefore introduces vulnerabilities on an uncountable number of networks. Gartner, Inc. published a report that the world will face 20.4 billion connected things by 2020 [1]. This indicates the breakthrough of integrating these devices

The final publication is available at Springer via https://doi.org/10.1007/978-3-030-05345-1_36.

into open enterprise networks. In large heterogeneous computer networks like at CERN, we find tens of thousands of registered general purpose devices.
The specialty of an organization like CERN is that it pursues an open network policy where staff can register a network device of any type needed for their work. This results in an immense growth of various smart interconnected devices, like closed-circuit television (CCTV) cameras, IP-phones, printers, network attached storages, oscilloscopes, industrial control systems etc. and increases the complexity of protecting this network against cyber-threats. IoT manufacturers approach a short time-to-market and do not provide firmware updates for a longer time, as PC operating system providers do. Furthermore, IoT manufacturers design hardware, but they often re-use source code and libraries that are outdated at the time of releasing a new IoT device. Due to this, IoT devices are generally and intrinsically insecure and it becomes easier to compromise them. This is why using them in home or open enterprise networks is so dangerous that it becomes a serious threat, especially when the devices have security, safety or operational implications.
A first step towards securing an open network is to know which IoT devices exist within the network, which becomes most important if a new security vulnerability comes up. The operator then needs to know which device models are affected and endanger the overall network. Making an inventory of thousands of different IoT devices by hand is not feasible, since there is no way to gather model related information from devices easily. Moreover, getting this information for different IoT device manufacturers over the network was not yet done in large scale.
To address this problem, we propose and evaluate a novel approach based only on thirteen clock characteristics to remotely identify different IoT device models of various manufacturers in a more reliable and non-invasive way compared to related work. The approach will not harm the device or attached equipment and works without any preconditions. The approach does not need a reference device that is present all the time either. The widely used and simple TCP timestamp feature allows us to identify multiple devices under test in parallel and a device does not even need to be in the same subnet than the fingerprinter. These facts point out the benefit of using our approach within a complex network environment.

1.1 Contributions

In this paper, we present a novel approach to identify IoT device models based only on clock characteristics and evaluate it on a large heterogeneous network with Machine Learning. We focus on interference-free network packets, the ability to scan large networks, being adaptable for heterogeneous devices and not causing faults on remote devices. Installing or modifying anything on the device under test (DUT) is not needed. In the following,

- we show a new approach to IoT device fingerprinting by measuring clock behaviors of embedded devices;

– we fingerprint 562 physical IoT devices in a highly heterogeneous, large-scale network at CERN;
– we validate our approach by distinguishing 51 different device models of our network;
– our evaluation shows that we are able to detect IoT devices with 97.03% precision, 94.64% recall and 99.76% accuracy on a validation-set using Machine Learning.

Section 2 introduces related work and background on computer clocks in general. It shows clock characteristics and why these clock characteristics occur. After that, it shows how one can measure them. Section 3 identifies clock characteristics in TCP timestamps and how we can use them to identify devices. After that, the section introduces our dataset and the features we use for Machine Learning. Finally, we show our evaluation results before we conclude.

2 Background and Related Work

This section introduces related work and the origin of the features we use for our approach. It begins with the definition of a timestamp clock that generates the timestamp values we measure. Afterwards, the section points out how we measure the clock characteristics and how clock skew is defined. The section then shows timestamp overflows we recognized on several devices and how we define this behavior.

2.1 Related Work

Most of the related work to identify network devices require preconditions on the infrastructure. Passive fingerprinting approaches need a software-defined-network [2] to interact with IoT devices or to mirror [3] the network traffic of multiple routers. Thus, they distinguish by the generated network traffic of the IoT device when it is in a setup process [2] or operation [3]. Active fingerprinting approaches like port scanners are too inaccurate and bring up operational and safety implications for IoT devices. Other approaches need to take over control on the equipped auxiliary, e.g. measuring opening times [4] of a valve when connected to a programmable logic controller to identify the IoT device. Another active approach is to send malformed packets to an IoT device and detect the device on the replied error message [5]. Other work focuses only on clock skew approaches and calculate the clock skew based on a reference device that always needs to be present [6] [7]. Researchers [8] also use clock skew to fingerprint the users of a cloud environment via asynchronous JavaScript and XML (AJAX) that is not adaptable for IoT environments, since IoT devices can not be forced to open and process a JavaScript web-page. In addition, all before-mentioned approaches were tested only on a few devices and therefore miss a large and heterogeneous dataset for verification.

2.2 TCP's Timestamp clock

A clock is a continuous counter triggered by an oscillator. The oscillator for the majority of devices with a real-time clock is a quartz that oscillates with a fixed frequency. We use the characteristic feature of how monotonic timestamp clocks generate TCP [9] timestamp values to detect model specific characteristics. A monotonic clock avoids clock corrections that would change the timestamp value periodically, which leads to a systematic clock drift that one can measure over time. The resolution [10] of the timestamp clock is defined to be in the range between 1 millisecond up to 1 second per tick [11] that describes the step-size of the counter. A timestamp clock is designed to generate incrementing values which a system can use for different measurements. In case of our usage, the timestamp values are stored in the option field [11] of TCP packets that let the communication partner measure the round trip time.

2.3 Defining clock skew

Every clock experiences clock skew that one can measure as clock drift over time. In comparison to an exact clock, the quartz crystal, integrated in most IoT devices, can have a typical error of ± 100 parts per million which results in a clock drift of ± 8.64 sec per day. The clock drift rate at a point in time is the clock skew and can be positive or negative, if the clock is faster or slower than an exact clock. This positive or negative skew remains constant over time as Kohno et al. [12] showed. The clock skew becomes measurable by periodically sending TCP packets to the DUT and processing the included TCP timestamps. The clock skew $\alpha(t)$ is a derivation over time and is defined as follows:

$$\theta(t) = C_B(t) - C_A(t) \tag{1}$$

$$\alpha(t) = \frac{d\theta(t)}{dt} = \delta_B(t) - \delta_A(t) \tag{2}$$

C_A and C_B in Equation 1 are the current clock values of a computer and the DUT in absolute time t. We calculate the relative clock offset between these clocks for our approach. Equation 2 specifies $\delta_B(t)$ and $\delta_A(t)$ that is the frequency error of a clock at an absolute point in time t. Moreover, Equation 2 shows that clock skew is calculated by the frequency error of an oscillator. We use these equations to calculate the clock skew of every device model and add it as a feature for Machine Learning in Section 3.2.

2.4 Timestamp overflows

A conspicuous feature we detected on longer scans is an overflow of the TCP timestamp values that happened on several IoT devices during our tests. This overflow of the timestamp values is caused, because the monotonic timestamp clock does not synchronize the time periodically. Thus, the timestamp values will not be corrected if the clock drifts and timestamp overflows will happen

more often. In the following, we show how we detect timestamp overflows and how we take advantage of this behavior. Figure 1 shows timestamp overflows of a printer and a telepresence device. The Hewlett-Packard Laserjet P3010 printer drops from the maximum timestamp value d_m of $114,221,300$ to a lower value d_b with $34,731$ at packet number 25. The Matrox Monarch HD telepresence device drops from the maximum of $8,598,140$ to the lower value $18,775$ at packet number 144. We detect these overflows by multiple Machine Learning features and use this behavior to identify devices more accurate.

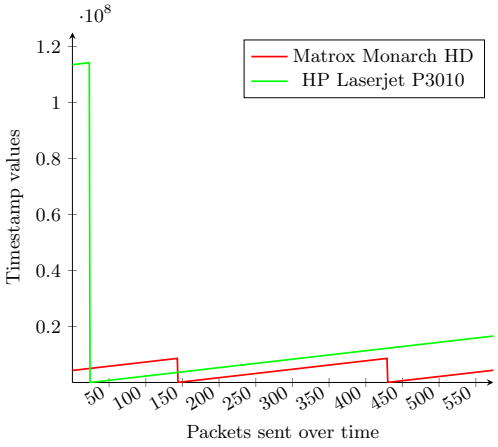

Fig. 1. Overflows of timestamp clocks

We specify the observed clock characteristics in the following formula:

$$clock_d(n) = d_b + n * d_r \bmod d_m \tag{3}$$

$$d_b = \begin{cases} d_b = 0, & \text{initially} \\ d_b = \varphi \text{ and } n = 0, & \geq d_m \end{cases} \tag{4}$$

We define Equation 3 that calculates the timestamp clock value $clock_d$ in relation to n ticks that happened during start. This equation is device model specific and starts on boot with d_b set to zero. When an overflow occurred, it is set to an implementation specific value φ. The device specific constant d_r is the resolution of the timestamp clock. Every iteration n adds this constant d_r, where n zeroes after an overflow has occurred. The upper limit of d_m is defined by RFC1323 [11] to the upper limit for unsigned 32 bit integers, but implementations sometimes differ and set a smaller value where they wrap. We observed that the drops in Figure 1 occur in a device-model-specific periodicity. This shows us that d_m in Equation 3 corresponds not always to the highest value defined by RFC1323. An additional implementation specific behavior occurs, when a timestamp clock begins to increment after an overflow greater than zero as $0 < d_b < d_m$. It is

obvious that the closer d_m is to the upper limit for unsigned 32 bit integers, the longer the scans need to be to detect an overflow. Using this information about overflows and lower value characteristics increases the preciseness for distinguishing specific models.

3 Clock-characteristic-based Device Identification using Random Forest classifier

This section introduces the different device classes of our approach along with the amount of scans in the different data sets. After that, the section lists which Machine Learning features we define to detect the different clock characteristics. The section then shows the comparison of different Machine Learning algorithms for our approach and points out the results we achieved with the Random Forest classifier.

3.1 Dataset of devices at CERN

CERN has a large-scale infrastructure with tens of thousands of devices, where 1000 of them are IoT devices. For our evaluation, we use a subset of these devices and limit our dataset to 562 devices. We encountered that 100 of the devices are only sporadically online. Hence, we are currently not able to gather enough scans for the training phase of Machine Learning. We also found four device models of different manufacturers that do not support TCP timestamps or reply with zero continuously. The remaining amount of 300 IoT devices are within a restricted network for accelerator purposes only. Therefore, one is not able to get access to this network due to security and safety restrictions. The device classes and physical quantities of our dataset are listed in Table 1. This table represents 51 device models categorized by their device class. Summing all quantities together results in 562 physical devices for our dataset. All network devices at CERN are registered at a database, where we extracted hostnames, device manufacturer and model information of all IoT devices. Since device manufacturer and model information are not mandatory fields, we investigated manually for certain devices to identify them correctly. Correct labeled information is mandatory for the training and validation phase of Machine Learning.
Due to the high quantity of *Printers* one can find at CERN, our dataset contains a comparatively larger amount of scans for this device class. On the one hand, more devices help us to generate our dataset faster. On the other hand, we mention in Section 3.3 that the classification algorithm is not biased by this fact. Table 1 also shows scans of the training and test set that we use in Section 3.3 for the Machine Learning algorithm. We hereby point out that a single scan consists of 576 values taken over time from the TCP timestamps. Our evaluation showed that the first timestamp overflows happen within this range for different devices. Before we can start with Machine Learning in Section 3.3, we split the dataset into a training and test set first as we show in Table 1. We train the Machine Learning algorithm after this split on the training set and use the test

Table 1. Device overview with physical quantity and scans per data-set

Device Class	Models	Quantity	Training set	Test set	Validation set
Arduino	1	4	225	49	14
IP to serial converter	1	4	787	224	58
IP phone	1	2	68	9	7
Light management	1	11	63	22	3
Network attached storage	16	35	4021	1054	294
Oscilloscope	1	2	70	21	9
Printer	11	390	78903	21006	5231
Projector	3	12	384	111	28
Telepresence system	4	37	2934	795	195
Video streaming system	1	25	12177	3263	810
Webcam	11	40	2416	658	155

set to put the classifier to the proof. After that, we use the validation set shown in Table 1 to verify our approach on never processed scans, point out the features for Machine Learning in Section 3.2 and show the results in Section 3.3.

3.2 Defining features for Random Forest classifier

We first used clock skew as a single feature to identify devices of our dataset, which was not giving us the intended results for our large dataset. One can see in Figure 2 of Section 3.3 that clock skew is the fourth most important feature for the Machine Learning algorithm to identify devices. This makes clear that by using clock skew as the only feature, one would not be able to identify the majority of devices within our dataset. Thus, we use 12 additional features besides clock skew to better identify devices, which are specified in Table 2. We defined these features to detect the changes of consecutive timestamps and the properties of the overall scan. Timestamp overflows are also detected with the combination of several features. All features are based on two types of sets. First, we define the set $A = a_0, ..., a_{n-1}, a_n$ as the set of all timestamp values of one scan period from a device model. One scan period is represented by 576 timestamp values; hence a is one single timestamp value of A. Secondly, we define the set $O = \{a_{i+1} - a_i\}_{i=1}^{N-1}$ that represents the offset of two consecutive timestamps of the set A.

3.3 Detection of devices

We take the dataset that we introduced in Section 3.1 and the features of Section 3.2 as input and evaluated our approach with different Machine Learning algorithms. We compared algorithms designed for classification problems, as in multilayer perceptron [13], support vector machine [14] and Random Forest [15] on the test set shown in Table 1. With the multilayer perceptron algorithm, we get a precision [16] of 91.65%, a recall [16] of 92.47% and an accuracy [16] of 99.22%. Support vector machine results in a precision of 92.05%, a recall of

Table 2. Features used for Random Forest classifier

ID	Features	Definition
0	Clock skew	See Section 2.3
1	Increase over all consecutive timestamp values	$a_n - a_1,$ where $n = \|A\|$
2	Largest increase of two consecutive timestamp values	$max(O)$
3	Mean increase over all consecutive timestamp offsets	$mean(O)$
4	Smallest increase of two consecutive timestamp values	$min(O)$
5	Timestamp overflow occurred	if $a_{i+1} < a_i,$ where $a \in A$
6	Sum over all consecutive timestamp offsets	$\sum_{i=1}^{N} o_i$
7	Last timestamp in A	$a_n,$ where $n = \|A\|$
8	Largest timestamp in A	$max(a)$
9	Median over all timestamps	$median(A)$
10	Smallest timestamp in A	$min(a)$
11	Cardinality of timestamp values	$\|A\|$
12	Sum over all timestamp values	$\sum_{i=1}^{N} a_i$

83.27% and an accuracy of 99.14%. The Random Forest algorithm shows a precision of 93.61%, a recall of 93.44% and an accuracy of 99.67%.
Our approach makes use of the uniqueness of device models in the TCP timestamp changes and makes them detectable. Distinguishing between different classes is a typical classification problem [17], because the Machine Learning algorithm needs to detect unique timestamp characteristics and assign it to a device model. Thus, we chose features that clearly separate the devices what one can see in the results above. The results for precision, recall and accuracy of Random Forest reached in average higher values in comparison to other algorithms that we show in the beginning. In the following, we continue with Random Forest.
Random Forest creates one decision tree classifier [18] for each device model during the training phase. In the test phase, the algorithm takes new scans as inputs. Each tree outputs the probability of every scan belonging to a certain device model. Finally the predicted device model is the tree with the highest mean probability estimate of all trees. Since our results are very distinct, we already mentioned in Section 3.1 that the Machine Learning algorithm is not biased. We examined this by splitting the training set into a test and validation set. First, we trained the Machine Learning algorithm with the training and test set. After that, the algorithm classified the scans of the validation set that were never processed before. We even used our classifier to identify scans of never seen devices and identified them successfully. The Random Forest results of predicting the device models of the validation set reach a precision of 97.03%, a recall of 94.64% and an accuracy of 99.76%.
As mentioned in related work, other researchers as Bratus et al. [5] or Kohno et al. [12] rely only on clock skew as a feature to identify devices. We applied our features related to TCP timestamp changes and the Random Forest algorithm to the initial clock skew feature, which resulted in better results that we present in this work. These additional features help us to identify different devices more clearly on a large-scale network with a bigger dataset compared to related work [2] [4] [19]. To increase our results, we performed several iterations of feature se-

lection on our dataset. Removing features by method [20] with low information gain resulted in choosing the features in Table 2. Furthermore, we show in Figure 2 the importance [21] of every single feature, where 100% importance would be 1.0 on the y-axis. Our evaluation showed that even the feature with the lowest importance correlates with other features; hence removing it would worsen our results significantly. For reasons of clarity, we use the Feature-IDs of Table 2 to match the numbers shown in Figure 2.

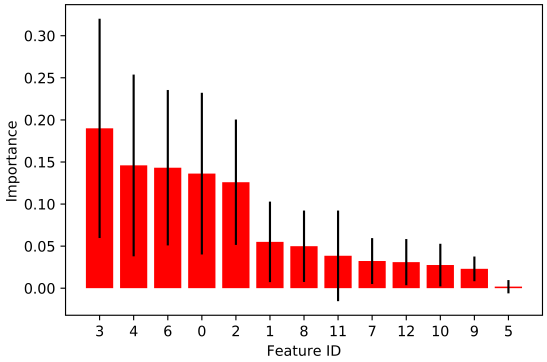

Fig. 2. Feature importance of the Random Forest classifier

4 Conclusion

We presented a novel approach to reliably identify various IoT device models in a real-world environment. Preconditions on the infrastructure, like software-defined-networks, network mirroring or reference devices are not needed. We showed that the approach is working on a large-scale network with a larger dataset compared to related work. Moreover, no other work was able to classify this amount of heterogeneous IoT device models by using just a single and easy accessible information source like TCP timestamps.
Due to the fact that our classification approach is non-intrusive, one can also use this approach for other infrastructures as in industrial IoT environments. For future work, we want to identify new types of IoT devices that come up together with industrial IoT devices on our accelerator complex test bed.

Acknowledgment

This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research.

References

1. GARTNER https://www.gartner.com/newsroom/id/3598917. Last accessed 05 February 2018.
2. MIETTINEN, Markus, et al. IoT Sentinel: Automated device-type identification for security enforcement in IoT. In: Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on. IEEE, 2017. S. 2177-2184.
3. MEIDAN, Yair, et al. ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In: Proceedings of the Symposium on Applied Computing. ACM, 2017. S. 506-509.
4. JAAFAR, Fehmi. An Integrated Architecture for IoT Fingerprinting. In: Software Quality, Reliability and Security Companion (QRS-C), 2017 IEEE International Conference on. IEEE, 2017. S. 601-602.
5. BRATUS, Sergey, et al. Active behavioral fingerprinting of wireless devices. In: Proceedings of the first ACM conference on Wireless network security. ACM, 2008. S. 56-61.
6. KASSEM, Mohamed M., et al. A Clock Skew Addressing Scheme for Internet of Things. In: Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on. IEEE, 2014. S. 1553-1557.
7. HUANG, Ding-Jie, et al. Clock skew based node identification in wireless sensor networks. In: Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE. IEEE, 2008. S. 1-5.
8. HUANG, Ding-Jie, et al. Clock skew based client device identification in cloud environments. In: Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on. IEEE, 2012. S. 526-533.
9. Internet Engineering Task Force, Transmission Control Protocol. https://tools.ietf.org/html/rfc793. Last accessed 05 July 2018.
10. KAMP, Poul-Henning. Timecounters: Efficient and precise timekeeping in SMP kernels. In: Proceedings of the BSDCon Europe, 2002.
11. Internet Engineering Task Force, TCP Extensions for High Performance. https://tools.ietf.org/html/rfc1323/. Last accessed 05 July 2018.
12. KOHNO, Tadayoshi, et al. Remote physical device fingerprinting. In: IEEE Transactions on Dependable and Secure Computing, 2005, 2. Jg., Nr. 2, S. 93-108.
13. HORNIK, Kurt. Approximation capabilities of multilayer feedforward networks. In: Neural networks, 1991, 4. Jg., Nr. 2, S. 251-257.
14. VAPNIK, Vladimir. Estimation of dependences based on empirical data. In: Springer Science & Business Media, 2006.
15. BREIMAN, Leo. Random forests. In: Machine learning, 2001, 45. Jg., Nr. 1, S. 5-32.
16. DAVIS, Jesse; GOADRICH, Mark. The relationship between Precision-Recall and ROC curves. In: Proceedings of the 23rd international conference on Machine learning. ACM, 2006. S. 233-240.
17. LIAW, Andy, et al. Classification and regression by randomForest. In: R news, 2002, 2. Jg., Nr. 3, S. 18-22.
18. SAFAVIAN, S. Rasoul; LANDGREBE, David. A survey of decision tree classifier methodology. In: IEEE transactions on systems, man, and cybernetics, 1991, 21. Jg., Nr. 3, S. 660-674.
19. RADHAKRISHNAN, Sakthi Vignesh, et al. GTID: A technique for physical device and device type fingerprinting. In: IEEE Transactions on Dependable and Secure Computing, 2015, 12. Jg., Nr. 5, S. 519-532.

20. KRASKOV, Alexander, et al. Estimating mutual information. In: Physical review E, 2004, 69. Jg., Nr. 6, S. 066138.
21. BREIMAN, Leo. Classification and regression trees. In: Routledge, 2017.