DAF: Device Annotation Framework

Matej Hulák[1,2], Václav Bartoš[2], Josef Koumar[1,2], and Tomáš Čejka[2]

[1]Czech Technical University in Prague, Prague, Czech Republic
[2]CESNET, a.l.e., Prague, Czech Republic
Corresponding author email: hulakmat@fit.cvut.cz

***Abstract*—Accurate identification of device type and operating system in network traffic is crucial for effective network monitoring, security enforcement, and anomaly detection. Nevertheless, creating datasets for this task is limited due to problematic annotation in a real-world environment. We propose Device Annotation Framework (DAF), a modular and extensible open-source framework for annotating large-scale network datasets with operating system and device type labels. To improve annotation precision in complex environments, DAF combines multiple independent annotation sources by label fusion and conflict resolution. We evaluate DAF on a ground truth dataset collected from a mid-sized network and demonstrate its capability to produce high-quality annotations.**

***Index Terms*—annotation, datasets, os fingerprinting, device type fingerprinting, network traffic classification, network traffic monitoring, machine learning**

I. Introduction and Background

Precise identification of the target operating system and device type is foundational to effective network monitoring, management, and security. Each operating system and device type can exhibit different network traffic patterns, protocol implementations, and known vulnerabilities [1]. Recognition of the operating system and device type is essential for improving Quality of Service and Experience, enforcing individual security policies for each operating system and device type, detection of abnormal changes in connected devices, and finding potential vulnerable devices connected to the network [2].

Due to the increasing proportion of encrypted traffic, the classical approaches of operating system and device type recognition are no longer feasible. Therefore, in recent years, the adoption of Machine Learning (ML) has gained popularity in the research community [3]. The operating system recognition was targeted by ML in recent works [2], for example, Hagos et al. [4] propose a two-stage LSTM-based tool that first predicts the underlying TCP congestion-control variant for classification, Laštovička et al. [5] propose a decision-tree classifier on extracted TLS handshake, and Zhang et al. [6] propose a bidirectional GRU network for modelling sequences of flows. Similarly, device type recognition was targeted in several studies [7], for example, Bai et al. [8] propose an LSTM–CNN cascade model on statistical and temporal features, Shahid et al. [9] design a Random Forest–based classifier trained on bidirectional flow packet sequences of the first ten packets, and Perdisci et al. [10] introduce IoTFinder framework that builds fingerprints over domain–query probabilities and uses cosine similarity with learned thresholds to identify co-located IoT devices at ISP scale. However, the successful adoption of ML is dependent on credible datasets [11].

Adoption of ML for network traffic classification shows that the laboratory-created datasets are not sufficient for the real-world network traffic classification [12]. Therefore, the relevant dataset should be created in a real-world environment. As described by Gong et al. [11], the creation of a good classification model requires a precisely and truthfully annotated dataset, as any false annotations in training data may compromise experiments and real-world deployment of the model. However, the creation of such a dataset from the real-world network in the network traffic classification domain poses a significant challenge as obtaining reliable ground truth for all devices may not be feasible [13]. Furthermore, large networks often contain more complex topologies, with the presence of bridges, NAT devices, gateways, and other intermediary infrastructure components, which further complicate the annotation process.

In recent years, multiple datasets have been published for both tasks, as shown in Table I. Some of them were captured in a fully controlled environment, either a laboratory network or the servers were under complete control of the authors. In such settings, it is difficult to obtain a large enough dataset that represents all behaviour types that occur in real networks. Nevertheless, without full control over the network devices, the creation of datasets is problematic due to annotation. However, there are ways to annotate such traffic. Previous works usually use HTTP user-agent [15] or reverse DNS lookup [19], but these methods alone might not be precise and often are able to annotate only a small part of devices in the dataset. Moreover, our previous experiments have shown that even with great care, it is usually impossible to truthfully annotate a real-world traffic dataset using only one source of information [21].

Motivated by these problems, we propose the Device Annotation Framework (DAF), which is a multi-source annotation framework specifically designed for use in large-scale ISP networks, where diverse and ambiguous traffic patterns are present.

This research was funded by the SOCCER project, Grant Agreement No. 101128073 supported by the European Cybersecurity Competence Centre., and also by the Grant Agency of the CTU in Prague, grant No. SGS23/207/OHK3/3T/18 funded by the MEYS of the Czech Republic.

978-3-903176-75-1 ©2025 IFIP

TABLE I: Table of related datasets for operation system and device type recognition with annotation method

Type	Name	Citation	Origin	Annotation
OS	Dataset Using TLS Fingerprints for OS Identification in Encrypted Traffic	[14]	University network	DHCP and RADIUS
	Passive Operating System Fingerprinting Revisited	[15]	University network	HTTP useragent
	Transferability of TCP/IP-based OS fingerprinting models	[16]	ISP network	DAF framework
Device	YourThings	[17], [18]	Laboratory	Manual
	CESNET-TimeSeries24	[19]	ISP network	Reverse DNS
	IoT-deNAT	[20]	Laboratory	Manual

The key contributions of this paper are as follows:

- We propose a framework for automatic annotation of types of devices and their operating systems in network datasets, leveraging multiple data sources.
- We evaluate the annotation accuracy of the framework by a ground truth dataset.
- We offer the framework to the community as open-source software.

Moreover, usability of the DAF was already proved by the annotation of real-world datasets used in an analysis of the transferability of OS fingerprinting methods across networks [16].

II. Device Annotation Framework

Device annotation framework (DAF) is a modular Python framework for annotation of network flow data by assigning the operating system and device type to each device (usually identified by its IP address) in a network. The main advantage of the framework is the use of multiple data sources for annotation, combined with label fusion for the final label assignment. This approach enhances annotation precision and coverage (i.e., the fraction of devices annotated) and enables creating more reliable datasets (and thus also models) for the detection of operating systems and device types.

DAF is open-source software available on GitHub[1]. The framework features a modular architecture that enables easy integration of new annotation sources, and supports central configuration and logging, parallel processing, re-annotation capabilities, IP prefix filtering, and precision threshold adjustment. The annotation process consists of several steps:

1) **Input:** A CSV flow dataset containing the fields required by the enabled annotators. The exact set of required fields depends on the chosen configuration.
2) **Modular Annotation:** Annotator modules process the flows grouped by source IP address (or another identifier, such as source MAC address) and assign an annotation to each device.
3) **Label fusion:** The annotations are merged using a voting mechanism.
4) **Output:** A final annotation for each IP address, annotated input dataset, as well as detailed information about the whole annotation process.

[1]https://github.com/CESNET/DAF

TABLE II: Device type annotation taxonomy

Group	Class
server	web, mail, dns, dhcp, ntp, syslog, vpn, honeypot, data, git, metacentrum, bot, authentication, smtp, proxy, multipurpose, development
net-device	core router, wifi router, firewall
end-device	workstation, mobile, tablet, wifi client, printer, voip, ups, payment terminal, ip camera, tv, smartwearable

TABLE III: Operating system families and types

OS family	OS type
windows	windows, server
macos	macos, ios, ipados
linux	debian, ubuntu, centos, rhel, gentoo, fedora, opensuse, arch linux, manjaro, oracle linux, rocky, cisco ios, oracle
unix	freebsd
android	android

A. Annotation taxonomy

To maintain consistency, it is necessary to define a common taxonomy used by all the annotation modules. Currently there is one taxonomy defined for classification of device types and one for operating systems. The device type taxonomy contains two fields: *group*, a general tag of a device type, and *class*, a more specific tag. The operating system taxonomy consists of three fields: *os_family*, *os_type*, and *os_version*, arranged in a hierarchy from general to specific. Assignment of more specific classes is always optional, used only if enough data is available. Allowed values for both taxonomies are listed in Table II and Table III. Since there are countless operating system versions, *os_version* is not treated as a categorical tag and may contain any value.

B. Annotation modules

Individual annotation modules are tasked to assign annotation labels in both taxonomies to each device. Each annotator uses a different method or data source. It is important that the annotators only assign a label if they can do it with very high

confidence; otherwise they do not assign any[2]. Annotators also have a possibility to mark IP addresses as possible NAT if the data suggest there might be multiple devices or OSes.

Currently, there are 5 annotation modules implemented in DAF:

Hand annotator uses a database of known devices to assign annotation labels. Data in the database is manually created from the knowledge of the network administrator. Hand annotation is assumed to always give correct labels and supersedes all other modules. Conflicts with the handwritten rules are logged as *hand_miss* and should always be reviewed. In order to test and use DAF, we created a database of known devices in our network. However, due to its specificity, we do not publish it.

Hostname annotator uses reverse DNS lookup to obtain the hostname from the IP address of a device, which is then compared against a database of known patterns (regular expressions applied on the whole hostname or its parts). To function correctly, the annotator must be run within a short period of time after the dataset capture, as later annotation attempts may produce false annotations due to potential changes in the network. We publish the database without patterns specific to our network. In order to use the full potential of the module, we recommend creating a tailored version of the patterns for each network.

Useragent annotator leverages information from the HTTP user-agent string, which contains information about the client browser, including its name, version, and the operating system. Based on a database of known OS signatures, gained from *WhatIsMyBrowser* (WIMB)[3], the annotator derives the OS and device type of the device. Since the full database of known signatures is very large and slow to search, the module first looks at a small set of keywords, such as common operating system names. Afterwards, it searches an optimised version of the WIMB database, where entries that differ only in details irrelevant to the analysis (such as browser version numbers) are merged, and the irrelevant parts are replaced with placeholders. As access to the *WIMB* database is subscription-based, we cannot publish even our optimised version, but we provide the script to perform the optimization.

SNI annotator uses information from TLS, QUIC, HTTP, and DNS headers to identify which websites a device visited. Some of these websites are specific to certain operating systems. For example, each OS connects to a set of URLs upon start and/or in certain periods to check for internet connection, system updates, or to sync the time. By analysing these connections, the module can often determine which operating system the device is using. The database, published with the module, was compiled manually based on our previous experience, analysis of traffic of known systems, and a dataset of idle traffic of different OSes [22].

[2]This makes the *annotators* different from *classifiers*, which always assign the most likely class, even if there is not enough information for confident decision.

[3]https://www.whatismybrowser.com/

The previous two annotators (Useragent, SNI) classify individual flows, and only assign a label to a device if there are at least *min_annotation_count* related classifiable flows and they all lead to the same result (conflicts lead to setting the NAT flag instead).

Shodan annotator uses information from *Shodan*[4], an internet scanner, to determine the type and operating system of the device. It first checks whether data is available for a given IP address using *Shodan*'s free InternetDB API, which has less strict rate limits and is suitable for high-volume queries. If relevant data are available, the module retrieves detailed information from the main *Shodan* API.

MAC annotator uses MAC address prefixes, which are known as OUIs (Organizationally Unique Identifiers). In some cases it is possible to derive the operating system based on the manufacturer of the device obtained by OUI. For example, devices manufactured by Apple will most likely use iOS, macOS, iPadOS, or some derivatives, which all belong to *macos os_family*. The module uses a simplified OUI database derived from the data available at MAC Address Lookup[5], and is published with the module.

It is worth noting that many devices connecting to the wireless networks are using MAC address masking (a.k.a. spoofing), preventing annotation. However, generated MACs should not conflict with any of the known OUIs. Therefore, even in this case, it should not cause mislabelling.

NAT detector is a special module designed to detect NAT, i.e. it never assigns any annotation label, only the NAT tag. The detection is based on two parameters: the number of unique source ports and the number of unique TTL values. NAT devices are expected to use a high number of source ports, as each new connection from the translated IP requires a new port. Observation of different TTL values in outgoing traffic is also a strong indicator of NAT. This approach is rudimentary and produces some false positives, but it has allowed us to detect and validate previously unknown NAT devices in our network. In addition, we are developing a more sophisticated version based on device communication characteristics, which will be published when available.

C. Label fusion

After all modules finish their work, the label fusion stage derives the final annotation for devices. Each device is evaluated separately, and each annotation field (a layer of a taxonomy) is processed independently. A final annotation is assigned only if: i) a defined minimum count of annotators (*min_annotators_count*) assigned a label, and ii) all the labels are the same, i.e. there are no conflicts. If a conflict occurs, no final annotation is assigned, but several cases are distinguished: if all but one annotator agree, the event is logged as *one_miss*, if more than two different labels are present, *possible_NAT* flag is set; if a hand annotation conflicts with another annotator, the final annotation follows the hand annotation, but the case is

[4]https://www.shodan.io/

[5]https://maclookup.app/

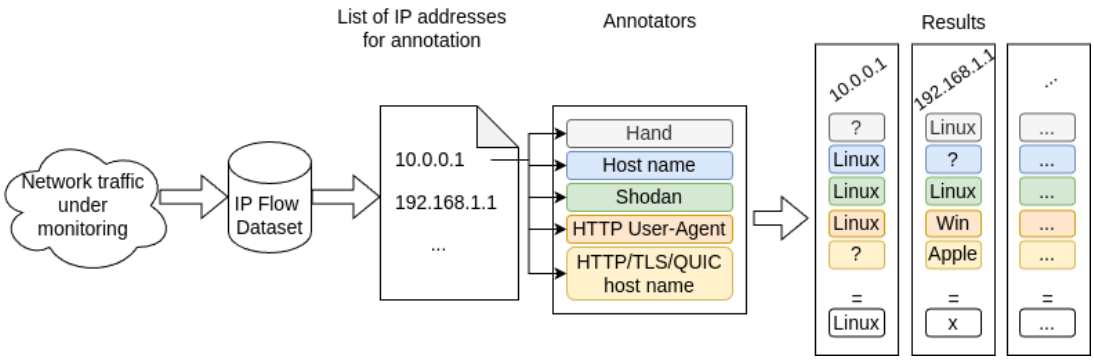

Fig. 1: Visual representation of the annotation process. IP addresses with consistent results from the annotators are labelled and included in the annotated dataset.

logged as *hand_miss*. The whole annotation process, including conflict scenarios in label fusion, is depicted in Fig. 1.

Even though the annotations in individual taxonomies are independent, there is a clear dependency between OS and device type. For example, if *os_type* is labelled as *ios*, it is highly unlikely (even though not impossible) that its *group* is *server*. Therefore, a set of rules is defined to detect and log suspicious combinations for further review.

The label fusion stage thus filters most of potential imprecisions which, despite the effort to make the annotators reliable, may still be present in the individual annotations.

The balance between annotation precision and coverage (fraction of devices that get annotated) can be controlled by the parameters *min_annotation_count* (on the module level) and *min_annotators_count* (on the label fusion level) – higher values lead to more reliable results at the cost of lower number of successfully annotated devices.

D. NAT detection

One of the main sources of false annotation in large networks is the presence of NAT, which hides multiple devices behind a single IP address. If only one of these devices actively communicates in a way that is used for annotation (e.g. sends unencrypted HTTP requests with User-Agent header), traffic from the remaining devices can be wrongly labelled. To counter this issue, we try to explicitly detect NATs in several ways:

- **Annotation modules:** each annotator can independently assign a NAT flag if it observes conflicting samples (e.g. multiple user agents or SNI values).
- **NAT detector:** a dedicated module that identifies NAT devices based on communication characteristics.
- **Label fusion:** if annotations from different modules conflict, the device is also labelled as NAT[6].

Combining all sources greatly improves NAT detection and reduces false annotations, making NAT handling one of the key features of the framework.

[6]Except when all but one annotator agree, which is treated as a likely mislabel.

E. Output

Besides the final annotated dataset, the annotation process results in several other outputs, which allow to examine and verify the results, debug any possible issues, or reuse some data in repeated runs. The outputs are as follows:

1) Annotated dataset: A copy of the input dataset with five new columns/fields added, which represent labels of the two taxonomies.
2) *ip_annotation_list*: A CSV file containing all discovered IP addresses with their *final_annotation*. If *export_full_annotation* configuration parameter is set, all labels assigned by the individual annotators are added as well. This helps to understand which annotators were used to derive *final_annotation* or to examine a conflict.
3) JSON data: This file contains all data from each module, it can be used for deep examination or for re-annotation purposes.
4) logging: Each part of the framework logs all important events, including any possible annotation conflicts.
5) stats: Reports include counts of processed IPs, successful and failed annotations, unannotated cases, possible NAT detections, overall flow statistics, success rate, and per-annotator results.

F. Re-annotation

Sometimes it is necessary to annotate a new dataset captured in the same network, where most devices are already known. Since the annotation process can be time and resource consuming, results from a previous run can be loaded and reused, so that only the new, previously unseen devices are annotated. The same approach can also be applied to large datasets, which may need to be divided into smaller chunks.

This is also useful when the data remain the same but module parameters, framework parameters, or module databases have changed. In this case, no additional API calls or feature extraction from the dataset are required, since all data gathered for each device are stored and can be reused.

III. Evaluation

A high-quality networking dataset must be realistic, large, diverse (i.e. capture the range of characteristics that occur in real networks), and precisely labelled. Automated annotation helps to build such datasets from real network flow data, but the accuracy of the resulting labels still needs to be verified, which requires a reliable source of ground truth.

To enable such evaluation, we captured flow data from a mid-sized network (part of a university network) where we were also able to obtain additional information about the connected devices. The network is configured to restrict access to pre-registered devices only, preventing unauthorised users or devices from connecting without prior approval. We were able to obtain the registration data which, although anonymised with respect to user identity, typically included sufficient detail to infer each device's operating system. Furthermore, we incorporated additional information sources: the vendor name derived from each device's MAC address, as well as the *vendor_class_id* [23] and *hostname* [24] fields from DHCP requests. The dataset was captured in September 2024 using *ipfixprobe*[7], with the following plugins enabled: `HTTP`, `TLS`, `QUIC`, `PSTATS`, `RTSP`, `SIP` (i.e. besides traditional NetFlow fields the data also contained some headers extracted from these protocols).

There were over 1,500 devices in the network. We used all the available data to determine operating system and device type of each of them. Only devices for which the available information provided sufficient confidence were included in the final annotated dataset. Although the process was labour-intensive and time-consuming, it resulted in a reliably labelled dataset comprising 949 devices and around 17 million flows. This dataset is used as the ground truth. The distribution of operating systems across the annotated devices is shown in Figure 2. Notably, Linux-based operating systems are under-represented, likely due to the nature of the network, which consists primarily of end-user devices, where Linux is rarely used. To ensure privacy, all sensitive data used during annotation was anonymised or filtered to the minimum necessary for the annotators, keeping only device and operating system information, with no user-identifying data.

Next, we used *DAF* to annotate the dataset. All modules were active except *hand_annotator*, whose use would compromise evaluation in this case, and *mac_annotator*, whose data are usually unavailable on large-scale networks. We also did not specially modify the pattern database of *hostname_annotator* to ensure fair testing. We tested three parameter settings for *min_annotation_count* and *min_annotators_count*: `1/1`, `2/1`, and `2/2`.

The dataset contained data from all annotated devices and data from 32 identified NAT devices (mostly home routers and hotspot-enabled machines). As detection of NAT devices is independent of annotation parameters, the framework detected 31 NAT devices in all test cases. After review, we found that *DAF* identified 29 out of the 32 known NAT devices, but also revealed two previously undetected NAT devices that were missed during manual annotation. Three NAT devices missed by *DAF* showed no observable NAT behaviour and had been identified during manual annotation using DHCP and local database data. These results demonstrate that the framework is highly effective at detecting NAT devices.

[7]https://github.com/CESNET/ipfixprobe

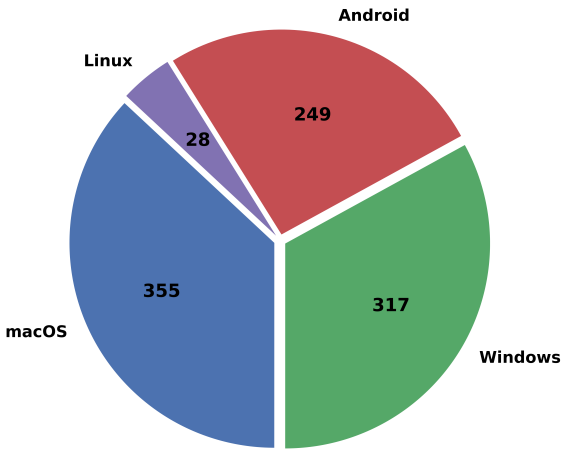

Fig. 2: Distribution of operating systems among annotated devices in the created dataset.

We then evaluated the precision and coverage of the annotation process, focusing on the *os_family* label, as we did not have sufficient data to evaluate *os_type* and *os_version*. The results are summarised in Table IV. Configurations `1/1` and `2/1` achieved identical performance, annotating 87 % of devices with 97 % precision, which suggests that all annotators had sufficient data. With the stricter `2/2` configuration, coverage dropped to 72 % while precision increased to 99 %. In this case, given that no unusual devices were present, the false positives are likely due to deficiencies in the databases and irregular behaviour of the devices. Out of the 23 falsely annotated devices in `1/1` configuration, one was labelled by both *useragent* and *SNI* annotators, 17 by *SNI_annotator* alone, and five by *useragent_annotator* alone. Annotation statistics also showed that *shodan_annotator* and *hostname_annotator* gathered little information and were generally unable to provide annotations, likely due to the nature of the network – there are only three servers, all the other devices are workstations, laptops, or smartphones, which usually do not have any open ports or descriptive hostnames.

These results indicate that even with reduced coverage, the framework maintains high precision and minimises false positives, which is highly advantageous for reliable annotation. In practice, the choice of configuration can be guided by the desired balance between coverage and precision. Configurations like `1/1` or `2/1` offer greater coverage with solid precision, while `2/2` provides increased precision with reduced coverage.

We also evaluated the accuracy of *device type* annotation against the ground truth (focusing only on the *group* label),

TABLE IV: Coverage and precision of *os_family* annotations under different label fusion configurations.

Configuration	Coverage	Precision
1/1	87 %	97 %
2/1	87 %	97 %
2/2	72 %	99 %

although it is not very informative due to the network characteristics, where all devices except the three servers fall into the *end-device* group. In every configuration, DAF correctly identified the servers, while the other devices were either labelled as *end-device* (correct) or unlabelled. This means 100% precision, while the coverage was 67 %, for the 1/1 configuration and only 48 %. for the stricter 2/2 configuration.

This shows that DAF is able to automatically label a large portion of devices in a dataset while maintaining very high precision.

Besides this experimental evaluation, DAF also proved its quality and usefulness in practice. It was used to create a series of annotated datasets, published on Zenodo [25], which was then used to research practical usability of ML-based OS-fingerprinting models, including their transferability across networks [16]. The results show that good classification results can only be achieved with large, diverse, and precisely annotated datasets – something that would be very difficult to obtain without an automated annotation framework like DAF.

IV. Conclusion

We presented device annotation framework, an open-source, modular system for annotating large-scale network datasets. By integrating multiple independent annotation sources and applying label fusion with configurable thresholds, DAF annotates most of the dataset, while keeping the number of false annotations very low. In addition, its ability to detect NAT at several stages further increases the reliability of the results.

This represents a significant improvement compared to previous works in the OS fingerprinting field, which only relied on a single source of annotation, leading to a higher chance of errors and lower coverage (which may introduce a significant selection bias). Moreover, its modular architecture enables easy integration of new annotation sources, making it highly customisable.

By publishing the framework as well as the series of datasets to the research community, we aim to support reproducible research and development of high-quality ML models for network traffic analysis.

References

[1] B. Anderson and D. McGrew, "Os fingerprinting: New techniques and a study of information gain and obfuscation," in *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2017, pp. 1–9.

[2] M. Laštovička, M. Husák, P. Velan, T. Jirsík, and P. Čeleda, "Passive operating system fingerprinting revisited: Evaluation and current challenges," *Computer Networks*, vol. 229, p. 109782, 2023.

[3] M. S. Sheikh and Y. Peng, "Procedures, criteria, and machine learning techniques for network traffic classification: a survey," *IEEE Access*, vol. 10, pp. 61 135–61 158, 2022.

[4] D. H. Hagos, A. Yazidi, Ø. Kure, and P. E. Engelstad, "A machine-learning-based tool for passive os fingerprinting with tcp variant as a novel feature," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3534–3553, 2020.

[5] M. Laštovička, S. Špaček, P. Velan, and P. Čeleda, "Using tls fingerprints for os identification in encrypted traffic," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–6.

[6] Z. Zhang, L. Tan, and D. Lv, "Fine-grained passive operating system identification based on network traffic sequences," in *2024 2nd International Conference on Computer, Vision and Intelligent Technology (ICCVIT)*. IEEE, 2024, pp. 1–5.

[7] H. Jmila, G. Blanc, M. R. Shahid, and M. Lazrag, "A survey of smart home iot device classification using machine learning-based network traffic analysis," *IEEE Access*, vol. 10, pp. 97 117–97 141, 2022.

[8] L. Bai, L. Yao, S. S. Kanhere, X. Wang, and Z. Yang, "Automatic device classification from network traffic streams of internet of things," in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, 2018, pp. 1–9.

[9] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Iot devices recognition through network traffic analysis," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 5187–5192.

[10] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis, "Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020, pp. 474–489.

[11] Y. Gong, G. Liu, Y. Xue, R. Li, and L. Meng, "A survey on dataset quality in machine learning," *Information and Software Technology*, vol. 162, p. 107268, 2023.

[12] D. Arp and et. al., "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988.

[13] D. Soukup, P. Tisovčík, K. Hynek, and T. Čejka, "Towards evaluating quality of datasets for network traffic domain," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 264–268.

[14] M. Laštovička, S. Špaček, P. Velan, and P. Čeleda, "Dataset using tls fingerprints for os identification in encrypted traffic," Sep. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3461771

[15] M. Laštovička, M. Husák, P. Velan, T. Jirsík, and P. Čeleda, "Passive operating system fingerprinting revisited - network flows dataset," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7635138

[16] M. Hulák, V. Bartoš, and T. Čejka, "Transferability of tcp/ip-based os fingerprinting models," in *2025 IFIP Networking Conference (IFIP Networking)*, 2025.

[17] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis, "Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis," in *2020 IEEE european symposium on security and privacy (EuroS&P)*. IEEE, 2020, pp. 474–489.

[18] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE symposium on security and privacy (sp)*. IEEE, 2019, pp. 1362–1380.

[19] J. Koumar, K. Hynek, T. Čejka, and P. Šiška, "Cesnet-timeseries24: Time series dataset for network traffic anomaly detection and forecasting," *Scientific Data*, vol. 12, no. 1, p. 338, 2025.

[20] Y. Meidan, V. Sachidananda, H. Peng, R. Sagron, Y. Elovici, and A. Shabtai, "Iot-denat: Outbound flow-based network traffic data of iot and non-iot devices behind a home nat," Jul. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3924770

[21] M. Hulák, V. Bartoš, and T. Čejka, "Evaluation of passive OS fingerprinting methods using TCP/IP fields," in *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2023, pp. 1–4.

[22] M. Novotná and V. Bartoš, "CESNET Idle OS Traffic [dataset]," Zenodo, Mar. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15004766

[23] B. Littlefield, "Vendor-identifying vendor options for dynamic host configuration protocol version 4 (dhcpv4)," RFC Editor, RFC 3925, Oct. 2004, rFC 3925. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3925

[24] S. Alexander and R. Droms, "Dhcp options and bootp vendor extensions," RFC Editor, RFC 2132, Mar. 1997, rFC 2132. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2132

[25] M. Hulák, V. Bartoš, and T. Čejka, "Transferability of TCP/IP-based OS fingerprinting models," https://doi.org/10.5281/zenodo.14703490, 2025.