Transferability of TCP/IP-based OS fingerprinting models

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

[1]Czech Technical University in Prague, Faculty of Information Technology, Czech Republic
Email: {hulakmat, cejkato2}@fit.cvut.cz
[2]CESNET, Prague, Czech Republic
Email: bartos@cesnet.cz

***Abstract*—OS fingerprinting provides valuable information about devices connected to a network infrastructure. Machine Learning (ML) models are presented as a feasible technology in existing studies. However, there is a lack of high-quality datasets to train a well-performing classifier that can be successfully transferred to different network environments. This paper proposes an improved annotation process to create more reliable datasets. Additionally, the paper showcases a new feature, TCP Maximum Segment Size (MSS), that proves to improve passive flow-based OS fingerprinting. The new datasets are used to train and evaluate ML classifiers for OS fingerprinting that are more transferable, with an average F1-score of 86 %. Furthermore, we conducted a thorough analysis, testing models across different networks and scenarios to identify key factors affecting performance. Along with the paper, we publish five annotated datasets that can be used for further research on this topic.**

***Index Terms*—OS fingerprinting, machine learning, traffic monitoring, model transferability**

I. Introduction

Network traffic is evolving every day, and the engineering community works hard to improve privacy and security. To keep up with this evolution of computer networks, continuous research for new network monitoring solutions is necessary to maintain the ability to detect security threats.

Classification of operating systems (OS) running on a device in the network, OS fingerprinting, in short, is one of the vital elements of every traffic monitoring system. Information about operating systems can greatly improve situational awareness [21] in the network and help network administrators or security specialists. The advantage is even bigger in the case of ISPs or large organizations, where users' end devices are often out of the control of the operators, and any vulnerable machine might become a security threat to the infrastructure.

OS fingerprinting tools are either active, which offer detailed information but are potentially intrusive, or passive, which discreetly classify the devices by observing their traffic. The passive methods are able to classify even devices hidden behind a firewall or Network address translation (NAT) or those not responding to incoming connections on any port. This study deals only with passive tools and methods, primarily due to their non-invasive nature.

The standard method for passive network monitoring today is the use of network flows. The state-of-the-art methods of OS fingerprinting are based on analysing selected TCP and IP headers of the traffic generated by the device, utilising the fact that different operating systems assign distinct default values to certain headers. Machine learning (ML) techniques are commonly used to create such classifiers. Additionally, TCP/IP features are not affected by encryption and are readily available, allowing them to be applied in almost all scenarios.

Indeed, there are several works that have proposed and evaluated OS fingerprinting methods based on TCP/IP features and ML classifiers with promising results. However, our previous experiments with large-scale network traffic revealed challenges in creating a well-performing OS fingerprinting model that is transferable across different networks [9]. This transferability is crucial to ensuring reliable classification in diverse environments.

In this paper, we evaluate TCP/IP-based OS fingerprinting more thoroughly, focusing specifically on real-world applicability. While previous research has shown promising results in passive OS fingerprinting, it often relied on single, narrowly focused datasets that provided limited insight into real-world applicability. To address this limitation, we collected multiple large-scale datasets simultaneously from distinct network environments, allowing us to assess both the accuracy and transferability of fingerprinting models more rigorously.

Reliable evaluation of ML models requires accurate annotation, as correctly annotated datasets form the basis of every machine learning experiment. To improve annotation, we developed a novel annotation technique that leverages information from multiple sources, significantly improving annotation accuracy. Additionally, we evaluate the effect of different traffic features, including

ISBN 978-3-903176-72-0 © 2025 IFIP

TCP Maximum Segment Size (MSS), on flow-based OS fingerprinting.

The main goal of this paper is to provide deeper insight into how this class of methods performs in different situations by conducting several experiments on new datasets with improved annotation, as well as to publish these datasets to the community to support further research in this field.

Specifically, the contributions of this paper are as follows:

- Introducing multiple large-scale datasets collected simultaneously from distinct network environments, enabling evaluation of model transferability.
- Proposing a novel annotation technique, which combines multiple annotation sources to ensure correctness.
- We confirm through extensive experimentation that models remain effective (achieving 82 – 91 % macro-averaged *F1-score*) when transferred across networks, provided the training datasets are sufficiently large and precisely labelled.
- We highlight the critical importance of dataset diversity and precise annotation, cautioning against evaluations based solely on single datasets.

In this research, we conclude that a model trained in one network can be used effectively in other networks, although its performance slightly decreases after transfer. We also emphasise that reliable OS fingerprinting models require large and diverse datasets with precise annotations.

II. Related work

The first OS fingerprinting tools like *p0f* [26], *siphon* [6] or *ettercap* [20], introduced in early 2000s, used databases of manually created fingerprints. Although methods for automatic fingerprint creation were introduced later (e.g., [4]), they struggled with several inherent challenges and limitations, as described by Richardson et al. in [22].

One of the first concepts of passive OS fingerprinting without the use of a signature-based database was introduced by Lippmann et al. in 2003 [17]. This study introduced a method for identifying operating systems based on specific attributes of TCP/IP packet headers (such as TCP initial window sizes, time-to-live (TTL) values, and IP header options) and machine learning techniques. By conducting comparative studies, they demonstrated that ML methods can offer better accuracy and efficiency in OS identification compared to traditional approaches.

Flow-based OS fingerprinting was introduced in 2014 by Jirsík et al. [11] and Matoušek et al. [19]. Both works presented an OS fingerprinting method that used network flows to determine the operating system of the device. Jirsík et al. used selected features and the *p0F* database, while Matoušek et al. derived a new database from *p0Fs* and additionally presented a clustering method that did not use any database at all. Both works demonstrated that OS identification using network flows is feasible, achieving high efficiency with only minor trade-offs in accuracy compared to traditional deep packet inspection.

In 2018, Laštovička et al. [13] extended the flow-based OS fingerprinting approach by evaluating three methods based on HTTP headers, TCP/IP features, and domain names. The authors also evaluated the possibility of a combination of all methods. Based on their observation, the User-Agent method was the most accurate, but it was able to classify only 64 % of the user sessions. In contrast, the TCP/IP-based method was able to cover the most user sessions, 88 %, with *Accuracy* of 81 %. Alongside the study, they published source codes used for the experiments and an annotated dataset.

In the same year, Laštovička et al. presented a new study [14], where they combined the flow-driven approach with machine learning techniques (namely Naive Bayes, Decision Tree, Support Vector Machine, and K-Nearest Neighbors). The experiments used the same dataset and the same three features as in [13]. The results clearly show that machine learning algorithms and TCP/IP parameters obtained by flow monitoring can be effectively used for OS fingerprinting.

The dataset from Laštovička et al. was also used in two studies that focused on solving the issue of high class imbalance in the data. Zhang et al. [27] proposed an active learning based method designed to maximize training efficiency for minority classes. Li et al. [16] compared multiple undersampling and oversampling methods along with their newly proposed method, which combines both the under- and over-sampling and uses a deep learning model. Although the newly proposed method achieved the best results, the differences between the tested methods were subtle. Thus, this study confirmed that it is important to balance the dataset, but even the traditional under- and over-sampling methods are still viable.

In 2023, Laštovička et al. published a comprehensive study [15] on current approaches and challenges in OS fingerprinting. The authors examined different OS fingerprinting techniques based on TCP/IP, application protocols, machine learning, and databases. They analysed features from TCP, IP, HTTP, TLS, and other protocol headers to identify which features can provide information about the operating system. To evaluate these fingerprinting methods, the authors created a new dataset and used it to assess the effectiveness of machine learning approaches, as well as methods based on manual analysis and signature-based databases. In conclusion, the authors evaluated machine learning methods

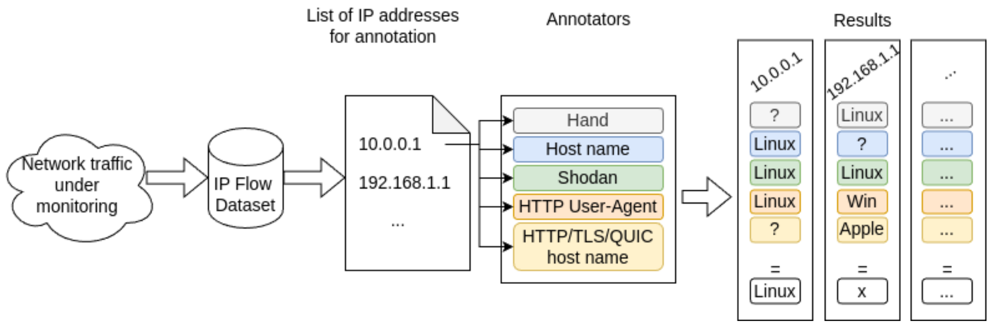

Fig. 1. Visual representation of the annotation process used to create new datasets. IP addresses with consistent results from the annotators are labelled and included in the annotated dataset.

as more suitable for modern, large-scale network OS fingerprinting. They also discussed current challenges and issues, such as the lack of datasets, reproducibility, and comparability, further emphasizing the need for ongoing research and methodological refinement in OS fingerprinting.

The need for further research was later confirmed in our previous experiments [9], which revealed severe issues with the transferability of ML models across networks. In the work, we examined the transferability of models between two publicly available datasets and a newly created one. We found that the classification performance of any model is significantly lower when applied to a different dataset than the one it was trained on (*F1-score* dropping from 64 – 88 % to 34 – 74 %).

It is expected that features of TCP/IP headers used for classification depend solely on the device's OS, so a model trained on data from one network should be well usable in other networks. This is also a requirement for practical usage since it is infeasible to train a separate model for each network (due to the need for a large annotated dataset).

There are several potential explanations — the used TCP/IP features actually depend on the environment, not just the OS (which is unlikely), the features change significantly over time (the datasets were captured at different times), or the datasets are of a low quality — they do not cover the diversity of behaviour of different operating systems sufficiently, or the annotation is not precise.

In summary, although the previous works has shown the effectiveness of the passive OS fingerprinting approach using machine learning, there is still lack of good datasets from multiple networks as well as insufficient understanding of behaviour of the method across different networks or changes of its performance over time. This work aims to fill this gap by providing new datasets and performing new experimental evaluations on them.

III. Data preparation

In general, a good dataset for machine learning must be representative of the real-world environment, ensuring the diversity of inputs is adequately captured. As stated in [24], creating a *good dataset* in the network traffic domain is challenging. This applies to OS fingerprinting datasets as well. Creation of a dataset from a small, well-known network (as in [1], for example) may cause the dataset to be too limited and risk insufficient diversity and sampling bias [3]. On the contrary, when a big dataset is captured in a large-scale network, it is almost impossible to get ground truth for all the devices there, so annotation is not feasible within the boundaries of most studies. Additionally, in many cases, there can be multiple devices or virtual machines (with potentially different operating systems) on the same IP address, e.g. because of NAT, which presents a risk of label inaccuracy [3].

The most popular annotation methods used in [7], [9], [13], [15] leverage information present in HTTP or DHCP packet headers. However, both methods have shortcomings, and none can surely reveal NAT[1].

A. Publicly available datasets

To the best of our knowledge, there are only two publicly available datasets, which were both published by Laštovička et al. [13], [15]. Both datasets contain data from a university network. The first, here referred to as *lasto18*, was captured in May 2017 and was annotated using DHCP and RADIUS log information. The second one, *lasto23*, is from 2023 and was annotated using HTTP User-Agent strings (by correlating flow data with web server logs). Both datasets can be downloaded from the university's website. The properties of the datasets are shown in Table I.

[1]For example, consider two devices behind a NAT, sharing the external IP address. If one of them sends an HTTP request, which is captured and used for annotation, all traffic from that IP address is labelled accordingly, including the one from the second device with a different OS, leading to incorrect labels.

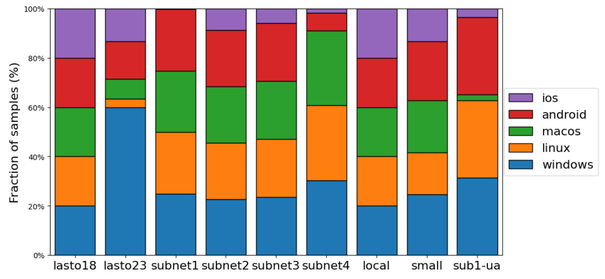

Fig. 2. Distribution of OS classes in the datasets after the preprocessing.

TABLE I
SUMMARY OF DATASET PROPERTIES, INCLUDING FLOW COUNT, NUMBER OF UNIQUE DEVICES, FEATURES, AND OPERATING SYSTEMS.

Dataset	Flow count	Unique IPs	Features	Unique OS
lasto18	6.9M	21746	17	25
lasto23	110k	–	112	12
subnet1	2M	8634	20	5
subnet2	5.5M	4667	20	5
subnet3	4.1M	5266	20	5
subnet4	14.1M	5680	20	5
local	17M	984	6	5
small	13k	9	20	5
sub1-ua	126k	5147	20	5

B. Local network

For this work we created several new datasets. The first one was captured in a local network of a university dormitory.

This network does not allow new users or devices without prior registration. We obtained the registration data related to the devices (user info was anonymised), which, in most cases, contained enough information to infer the operating system of the device. This information was supplemented by additional annotation sources: vendor name extracted from the device's MAC address[2], and *vendor_class_id* [18] and *hostname* [2] fields from DHCP request packets.

As part of the annotation process, we examined over 1,500 devices in the network, leveraging all available information to determine each device's operating system. Devices were included in the final annotated dataset only if the available data provided sufficient certainty about the OS. While the process was labour-intensive and time-consuming, it resulted in a reliably annotated dataset with 987 devices (IP addresses) and 17 million flows. This dataset is referred to as *local* in this study.

C. Datasets from a large network

In order to obtain larger and more diverse datasets, we captured flow data from four distinct subnetworks of *CESNET3*, which serves as the national research and education network of the Czech Republic. The first subnet (*subnet1*) contains networks of several small institutions, and the other three (*subnet2-4*) belong to large universities. All data were captured on the same day in September 2024. The properties of the datasets are shown in Table I (*sub1-ua* and *small* are special subsets of the other ones, described later in Section IV-C).

1) Annotation: Again, the data must be labelled, and the labels should be as precise as possible. But in this case, due to the size of the datasets, the annotation process must be automated.

[2]By using a publicly available database of OUIs, prefixes of MAC addresses assigned to manufacturers.

There are multiple possible approaches for automatic annotation, but since none of them is able to reliably label all devices, we decided to use multiple methods and combine their results. This allows us to increase the number of annotated devices as well as the reliability of the labels. It also allows to detect many cases of multiple devices behind a single IP address. The list of the annotation methods used is shown below:

1) *Hand annotator*: Uses a database of over 100 IP addresses manually labelled using our internal knowledge of the network.
2) *Hostname annotator*: DNS hostname assigned to the IP address sometimes reveals useful information, including the OS type (or that it is a NAT, for example). The method performs reverse DNS lookups and applies a set of carefully crafted rules to the results.
3) *Shodan annotator*: Uses information from *Shodan* [23], an internet scanner, to determine the device's operating system.
4) *HTTP User-Agent annotator*: Uses the User-Agent strings observed in plain HTTP requests and searches them in the database [25] of known signatures.
5) *HTTP Host annotator*, *TLS annotator* and *QUIC annotator*: Analyse names of visited domain names, extracted from HTTP Host headers, TLS SNI value, or QUIC SNI value, respectively, and compare it against a database of known domains associated with certain operating systems.

Each annotation method independently assigns an appropriate label within the limits of the defined taxonomy: `Android`, `iOS`, `Linux`, `macOS`, `Windows`. Annotators are implemented such that a label is assigned only if it is known with high confidence. The results of individual annotators are then compared: if all annotations are consistent, the final annotation label is assigned to the device. If there is any conflict between individual annotations or no method is able to annotate it, the address cannot be

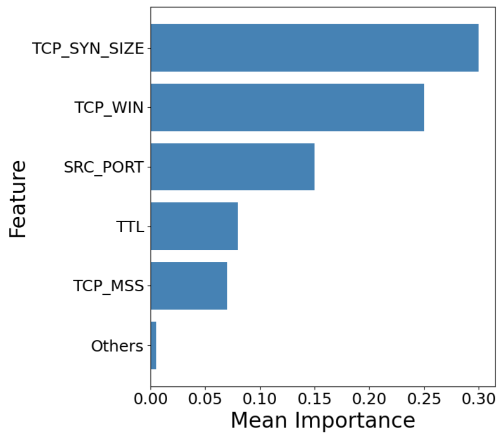

Fig. 3. Visual representation of the feature importance, based on feature permutation, for Random Forest classier.

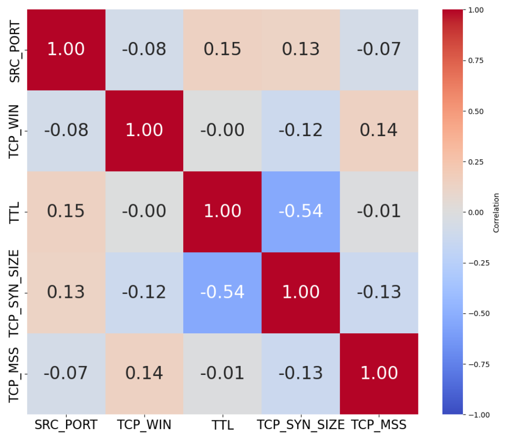

Fig. 4. Feature correlation matrix containing Pearson correlation coefficients.

confidently labelled and its data are dropped from the dataset. The conflicts are logged and later reviewed, which often results in manual label assignment or improvement of the rule-based annotators (and re-running the process). The annotation process is illustrated in Figure 1.

This process also significantly improved our ability to identify IP addresses that exhibit behaviour of multiple different operating systems, which might be caused by multiple devices or virtual machines behind NAT, for example.

Note that dropping samples from a dataset based on their labels (or our inability to assign a label) can introduce a selection bias [3], which may negatively impact the results. However, in this case, it is unavoidable, as it is not feasible to reliably annotate all devices in a large network. To minimize this issue, we employed multiple annotation methods, which increased the fraction of annotated devices. This is a significant improvement over previous works, which mostly relied on a single method.

To validate the annotation process, we utilized the data from the local network, described in III-B. We annotated the data by the multi-annotator method described above and compared its results against the manually assigned labels. The multi-annotator method successfully labelled 79.3 % of the flows, out of which 98.4 % flows were labelled correctly, i.e., matched the manual annotation. Out of the 987 devices, only fifteen were labelled differently, mostly confusing *Android* OS with *Linux* OS. These results demonstrate that the multi-annotator method is capable of high accuracy.

All the created datasets (*subnet1-4*, *local*), including annotations, are published in the Zenodo repository [8].

IV. Experiments

In this section, we performed a series of experiments on the datasets in order to select the optimal settings for the experiments and to review transferability of the ML models across different networks.

A. Dataset preprocessing

To conduct machine learning experiments, some preprocessing of the datasets is necessary. First, network flows that had `TCP_SYN_SIZE` equal to zero were removed, as it signals that the first packet of the connection was not captured (applies to the flow exporter used [5]). We also removed flows with `TCP_WIN` size equal to zero, as this typically signifies that the receiver's buffer was full and could not accept more data. Additionally, all `TTL` values were adjusted to the nearest higher power of two, as recommended in [17] and [9]. This adjustment is crucial to prevent data snooping [3], where classification models learn to distinguish devices based on their distance from the probe. Flows from devices that were not annotated or were labelled as exceptions (such as NATs) were removed[3].

Due to the data being captured from a real-world network, the distribution of classes is highly imbalanced. To address this, random under-sampling was applied to classes with more than 40,000 samples. This approach helped to mitigate class imbalance while maintaining sample diversity. Additionally, we set a maximum number of flows per device to ensure balanced representation. The updated distribution of the

[3]Note that in practical deployment, detection of NAT should be done separately (e.g., by correlating OS classification results based on multiple flows from each IP address, or by dedicated methods, such as [12]). However, this is out of the scope of this paper. Here we focus on evaluating the method in a simplified case where each IP address corresponds to one device.

classes is shown in Figure 2. Furthermore, to limit the impact of class imbalance on evaluation, we exclusively used macro-averaged *Precision* and *F1-score* in this work.

1) Selection of machine learning classifier: To select the ML classifier for the experiments, we performed preliminary tests on the *local* and *subnet1* datasets with ten different types of classifiers (such as Decision tree, Random Forest (RF), Support Vector Classifier, AdaBoost, LightGBM, MLP, and others).

All tree-based classifiers achieved similar results, outperforming other types. Moving forward, we decided to use the RF classifier because of its strong performance and the ability to directly compare the results with those of previous works.

2) Feature selection: Not every network flow feature contains information about OS. Based on previous studies [7], [9], [15], [16], we opted to use the following features: `SRC_PORT, TCP_SYN_SIZE, TCP_WIN_SIZE, TTL` (definitions can be found in *ipfixprobe* documentation [5]).

We also explored various features, with special attention to *TCP options*. We examined seventeen TCP options (`TCP_MSS, WS, SACKPerm, SACK, SKTR, BUBBA, TCHO, SCPS, SNA, RB, SNAP, TCPCF, QSR, TCP_AO, MPTCP, TCPFOC, TCP_ENO` [10]). Among these, only `TCP_MSS` demonstrated a minor improvement. Even though `TCP_MSS` is commonly used by tools like *p0F*, *ettercap* or *Nmap*, recent studies often do not include this feature in the ML feature set. To evaluate the usefulness of the `TCP_MSS` feature, we conducted experiments on the *local* dataset, calculating permutation feature importance for the RF classifier and examining feature correlations. Feature importance based on permutation for the RF classifier is illustrated in Figure 3, and feature correlations are shown in Figure 4.

The results show that `TCP_MSS` does not correlate with other features and demonstrates positive feature importance. Furthermore, usage of the feature improved the classification accuracy on some of the other tested datasets (*subnet1*, *subnet4*). Even though improvements were minor, it is clear that usage of the `TCP_MSS` is beneficial. For this reason, we used this feature in the following experiments when possible.

3) Selection of OS classes: Ideally, we aim to classify the operating systems of all devices within a network. However, this goal is constrained by practical limitations, such as the number of relevant features, the diversity of classification classes, and the challenges of creating and annotating training datasets. Most available datasets contain suitable data for only five operating systems: `Android, iOS, Linux, macOS, Windows`. Preliminary experiments on the *local* dataset revealed that RF models struggled to distinguish *iOS* from *macOS*, as shown in the confusion matrices in Figures 5 and 6.

TABLE II
CLASSIFICATION RESULTS OF RF MODELS TRAINED AND TESTED ON INDIVIDUAL DATASETS. *Precision* AND *F1-score* ARE MACRO-AVERAGED.

	Accuracy	Precision	F1-score
subnet1	96 %	96 %	96 %
subnet2	92 %	92 %	92 %
subnet3	88 %	91 %	87 %
subnet4	97 %	96 %	96 %
local	99 %	99 %	99 %
lasto18	92 %	93 %	92 %
lasto23	93 %	90 %	85 %

To find the cause, we analysed feature statistics and pairwise relationships (*pairplot*) between *iOS* and *macOS* features, which indicated significant similarity. This limitation is likely due to similar implementations of the TCP/IP stack in those systems. Based on these findings, we opted to merge *iOS* into the *macOS* class, reducing OS classes to `Android, Linux, macOS, Windows`.

B. Individual networks

After preprocessing and determining the experiment settings, we performed experiments on the individual datasets.

Data from each dataset were split into training, validation, and testing sets in a ratio of 60:20:20, evenly stratified according to the OS label. All datasets were under-sampled to limit class imbalance (detailed in Section IV-A). The classification model was trained on its respective training split, then validated and tuned on the validation split. The created model was then tested using its testing split. Models were trained using five features: `TCP_SYN_SIZE, TCP_WIN, TCP_MSS, TTL, SRC_PORT`. The older datasets (*lasto**) do not contain `TCP_MSS`, so experiments on these datasets were conducted using only four features.

The results are shown in Table II. All models achieved macro-averaged *F1-score* higher than 85 %, which is an excellent performance. The scores achieved on *lasto** datasets are even higher than those reported in previous works [9], [13], [15], possibly due to the new preprocessing procedure, specifically by filtering flows with `TCP_SYN_SIZE` and `TCP_WIN` equal to zero.

Overall, when we work with data from each network individually, the results are very good and are in line with previous works.

C. Model transferability

Next, we test how a model trained on data from one network behaves when applied to data from another network – which is very important for practical deployment. First, we used the newly created datasets from the *CESNET3* network. A classification model was

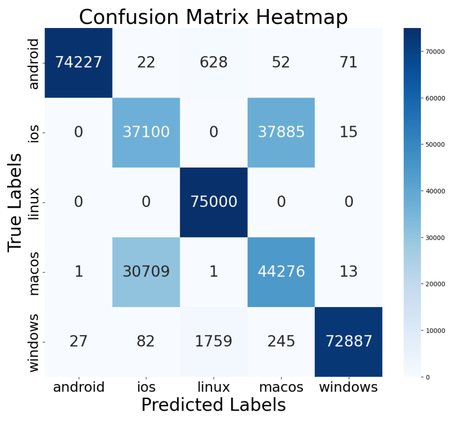

Fig. 5. Confusion matrix for RF model on *local* dataset with five OS classes.

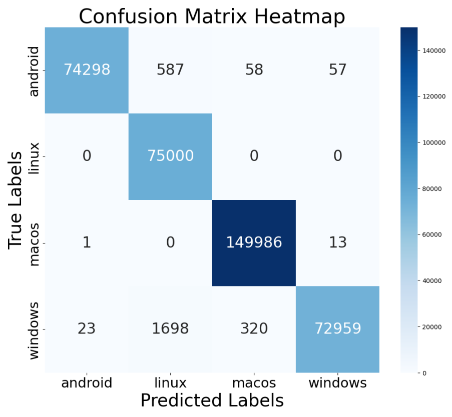

Fig. 6. Confusion matrix for RF model on *local* dataset after merging the *iOS* class into the *macOS* class.

trained on each of the *subnet** datasets and tested on the others. Additionally, we included the *local* dataset, which served as a baseline due to its highest annotation reliability.

As in the previous section, the datasets were split, stratified, and under-sampled. Each model was trained on the corresponding training split, tuned on the validation split, and tested on its testing split as well as on the entirety of other datasets. All models were trained using five TCP/IP features: `TCP_SYN_SIZE, TCP_WIN, TCP_MSS, TTL, SRC_PORT`. The results of these experiments are shown in Table III. Rows indicate the training datasets, while columns represent the testing datasets.

Unsurprisingly, the best-performing model for each test dataset is the one trained on the same dataset. There is a significant drop in classification performance when a model is tested on a different dataset than on which it was trained. The difference ranges from 1 to 22 %, which is a significant improvement compared to the previous work [9], where differences ranged from 7 to 52 %.

This means that the use of better datasets with more reliable annotation helped significantly. On the other hand, the drop caused by testing on data from a different network should be theoretically close to zero, so there is still room for improvement. Overall, the averages reported in Table III are all over 80 %, which is still a good performance, not far from the state of the art.

If we focus on the most reliably annotated dataset, *local*, the model trained on this dataset struggles to reliably classify operating systems in the *subnet1-3* datasets. This could indicate either a lack of diversity in *local* or that the *subnet1-3* datasets, despite the significantly improved annotation, still contain some mislabelled data.

To further explore this issue and assess assumptions about annotation accuracy and dataset diversity, we compared the performance of *local* with two new specially crafted datasets – *sub1-ua*, which is the same as *subnet1*, but annotated by only the *HTTP User-Agent annotator*, and *small*, which contains traffic of only nine devices, but with 100 % confident annotation. The results of these experiments are shown in Table IV.

The model trained on the *local* dataset, which generally achieves the best results, has *F1-score* of only 82 % when tested on *sub1-ua*, worse than on the full *subnet1*. This is probably caused by the lower quality of annotation (i.e. more misclassified samples) of *sub1-ua*. There is no performance drop on the well-annotated *small* dataset. This signifies that the results are heavily affected by the quality of annotation.

However, even with perfect annotation, the results might still be poor if the training dataset is not diverse enough – this is evident from the model trained on the *small* dataset, which achieved an *F1-score* of only 75 % when tested on *local*. In contrast, the model trained on *sub1-ua* dataset exhibits poor results overall, stressing out the advantage of our multi-annotation method.

Finally, using the same methodology, we compared two of the newly created datasets with the publicly available ones. The results are shown in Table V. Only four features were used for this experiment, excluding the `TCP_MSS` feature.

On average, the models trained on our *subnet4* and *local* datasets perform significantly better than those trained on *lasto18* and *lasto23* datasets, which indicates improvement in the ability to perform on unseen data from different networks.

Based on these results, we can confidently state that usage of the better annotation technique and added `TCP_MSS` feature significantly improved the performance of the models on their own dataset as well as on the others, i.e. their transferability to other networks. This enhances the usability of the created models, making them more suitable for practical deployment.

TABLE III
F1-scores ACHIEVED BY RF MODELS ON DIFFERENT DATASETS. ROW SPECIFIES THE TRAINING DATASET, COLUMN THE TESTING ONE. *F1-score* IS MACRO-AVERAGED.

↙ train, test →	subnet1	subnet2	subnet3	subnet4	local	average
subnet1	96 %	77 %	74 %	86 %	84 %	83 %
subnet2	79 %	92 %	86 %	85 %	78 %	84 %
subnet3	70 %	91 %	87 %	86 %	78 %	82 %
subnet4	85 %	87 %	85 %	96 %	98 %	90 %
local	90 %	87 %	84 %	95 %	99 %	91 %
average	84 %	87 %	83 %	90 %	87 %	86 %

TABLE IV
COMPARISON OF PERFORMANCE OF THE *local* DATASET WITH *sub1-ua* (*subnet1* ANNOTATED BY USER-AGENT DATA ONLY) AND *small* (NINE DEVICES, PERFECT ANNOTATION). VALUES REPRESENT MACRO-AVERAGED *F1-score*.

↙ train, test →	local	small	sub1-ua
local	99 %	99 %	82 %
small	75 %	99 %	64 %
sub1-ua	68 %	54 %	73 %

V. CONCLUSION

New machine learning-based monitoring methods show promising results, but their deployability in the real-world networks is often overlooked. This study aimed to fill the gaps in the current state-of-the-art methods to enhance their real-world applicability.

The first contribution is a set of new datasets captured in the same way and on the same date, but in different networks, making them a great resource for testing the transferability of trained models across networks. The datasets were annotated using a combination of multiple annotation methods, which increased both the proportion of devices annotated and the reliability of labels. Although this method is still likely imperfect and may produce a small number of incorrect labels, it is a significant improvement over any of the previous works. By performing experiments on these datasets, we first demonstrated the usefulness of the `TCP_MSS` feature (in addition to the four features used in all previous works). Although the improvement it provides is small, it is not insignificant.

Next, we evaluated the classification performance of Random Forest models trained on individual datasets. In this case, the results are very good and similar (or even better) to those reported in previous works.

However, for practical deployment, it is important that a single model is applicable in many networks. Theoretically, it should not be a problem since the default TCP/IP header values of various operating systems are not expected to change with the network environment. We tested this assumption by training and testing models on data from different networks.

In all cases, the classification models performed worse after transferring to another network, but they are still usable (82 – 91 % macro-averaged *F1-score*). So, we can conclude that a model trained in one network can be used effectively in other networks as well. However, two important requirements must be met – the training dataset must be large and diverse enough and it must be labelled precisely. Which is quite difficult to achieve.

Of course, the requirements for good annotation and a sufficiently large dataset are nothing new. However, in this paper, we demonstrated how important they are in this particular case of OS fingerprinting methods and how testing on just a single dataset might provide misleading results for practical usage.

A. Future work

In our future work, we aim to further enhance the real-world deployability of the TCP/IP flow-based OS fingerprinting approach by creating a more diverse dataset with additional classes, enabling broader OS family coverage. We also plan to apply *Open Set Recognition* methods to classify unseen devices and evaluate possible *Data Drift* by assessing the model performance over time. Finally, in order for this method to work well in practice, it is necessary to reliably recognize cases of multiple devices behind a single IP address, such as NAT.

ACKNOWLEDGMENT

This work was supported by project no. VJ02010024 funded by the Ministry of Interior of the Czech Republic, and by the SOCCER project, which is funded under Grant Agreement No. 101128073 and supported by the European Cybersecurity Competence Centre.

TABLE V
ILLUSTRATES MACRO-AVERAGED *F1-score* ACHIEVED BY RANDOM FOREST MODEL. ROW SPECIFIES THE TRAINING DATASET, COLUMN THE TESTING ONE.

↲ train, test →	local	subnet4	lasto18	lasto23	average
local	99 %	95 %	90 %	71 %	89 %
subnet4	96 %	95 %	79 %	66 %	84 %
lasto18	68 %	59 %	92 %	62 %	70 %
lasto23	77 %	82 %	60 %	85 %	76 %
average	85 %	83 %	80 %	71 %	80 %

REFERENCES

[1] Ahmet Aksoy and Mehmet Hadi Gunes. Operating system classification performance of tcp/ip protocol headers. In *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 112–120, 2016.

[2] S. Alexander and R. Droms. Dhcp options and bootp vendor extensions. RFC 2132, RFC Editor, March 1997. RFC 2132.

[3] Daniel Arp and et. al. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, Boston, MA, August 2022. USENIX Association.

[4] Juan Caballero, Shobha Venkataraman, Pongsin Poosankam, Min G Kang, Dawn Song, and Avrim Blum. Fig: Automatic fingerprint generation. 2007.

[5] CESNET. ipfixprobe - ipfix flow exporter. https://github.com/CESNET/ipfixprobe, 2019.

[6] Subterrain Security Group. Passive os fingerprinting tool. *Available at https://web.archive.org/web/20050205014947/http://www.subterrain.net/projects/siphon/*, 2000.

[7] Desta Haileselassie Hagos, Anis Yazidi, Øivind Kure, and Paal E. Engelstad. A machine-learning-based tool for passive os fingerprinting with tcp variant as a novel feature. *IEEE Internet of Things Journal*, 8(5):3534–3553, 2021.

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

[9] Matej Hulák, Václav Bartoš, and Tomáš Čejka. Evaluation of passive OS fingerprinting methods using TCP/IP fields. In *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–4, 2023.

[10] Internet Assigned Numbers Authority (IANA). TCP parameters, 2023.

[11] Tomáš Jirsík and Pavel Čeleda. Identifying operating system using flow-based traffic fingerprinting. In Yvon Kermarrec, editor, *Advances in Communication Networking*, pages 70–73, Cham, 2014. Springer International Publishing.

[12] Ali Safari Khatouni, Lan Zhang, Khurram Aziz, Ibrahim Zincir, and Nur Zincir-Heywood. Exploring nat detection and host identification using machine learning. In *2019 15th International Conference on Network and Service Management (CNSM)*, 2019.

[13] Martin Lastovicka, Tomas Jirsik, Pavel Celeda, Stanislav Spacek, and Daniel Filakovsky. Passive os fingerprinting methods in the jungle of wireless networks. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2018.

[14] Martin Laštovička, Antonín Dufka, and Jana Komárková. Machine learning fingerprinting methods in cyber security domain: Which one to use? In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 542–547, 2018.

[15] Martin Laštovička, Martin Husák, Petr Velan, Tomáš Jirsík, and Pavel Čeleda. Passive operating system fingerprinting revisited: Evaluation and current challenges. *Computer Networks*, 229:109782, 2023.

[16] Jingzhi Li, Ziling Wei, and Shuhui Chen. Passive os identification in imbalanced dataset. In *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–6, 2023.

[17] Richard Lippmann, David Fried, Keith Piwowarski, and William Streilein. Passive operating system identification from tcp/ip packet headers. In *Workshop on Data Mining for Computer Security*, volume 40. Citeseer, 2003.

[18] B. Littlefield. Vendor-identifying vendor options for dynamic host configuration protocol version 4 (dhcpv4). RFC 3925, RFC Editor, October 2004. RFC 3925.

[19] Petr Matoušek, Ondřej Ryšavý, Matěj Grégr, and Martin Vymlátil. Towards identification of operating systems from the internet traffic: Ipfix monitoring with fingerprinting and clustering. In *2014 5th International Conference on Data Communication Networking (DCNET)*, pages 1–7, 2014.

[20] Alberto Ornaghi, Marco Valleri, Emilio Escobar, Eric Milam, Gianfranco Costamagna, and Alexander Koeppe. The ettercap project, 2015.

[21] Timea Pahi, Maria Leitner, and Florian Skopik. Analysis and assessment of situational awareness models for national cyber security centers. In *ICISSP*, pages 334–345, 2017.

[22] David W Richardson, Steven D Gribble, and Tadayoshi Kohno. The limits of automatic os fingerprint generation. In *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, pages 24–34, 2010.

[23] Shodan. https://www.shodan.io/.

[24] Dominik Soukup, Peter Tisovčík, Karel Hynek, and Tomáš Čejka. Towards evaluating quality of datasets for network traffic domain. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 264–268, 2021.

[25] WhatIsMyBrowser.com. https://www.whatismybrowser.com/.

[26] Michal Zalewski. Passive os fingerprinting tool, 2003. http://lcamtuf.coredump.cx/p0f.shtml.

[27] Daowei Zhang, Qiujie Wang, Ziling Wei, and Shuhui Chen. An operating system identification method based on active learning. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–6, 2022.