Annals of Telecommunications
https://doi.org/10.1007/s12243-025-01114-z

Towards identification of network applications in encrypted traffic

Ivana Burgetová[1] · Petr Matoušek[1] · Ondřej Ryšavý[1]

Received: 6 January 2025 / Accepted: 11 August 2025

Abstract
Network traffic monitoring for security threat detection and network performance management is challenging due to the encryption of most communications. This article addresses the problem of identifying network applications associated with Transport Layer Security (TLS) connections. The evaluation of three primary approaches to classifying TLS-encrypted traffic was carried out: fingerprinting methods, Server Name Indication (SNI)–based identification, and machine learning–based classifiers. Each method has its own strengths and limitations: fingerprinting relies on a regularly updated database of known hashes, SNI is vulnerable to obfuscation or missing information, and AI techniques such as machine learning require sufficient labeled training data. A comparison of these methods highlights the challenges of identifying individual applications, as the TLS properties are significantly shared between applications. Nevertheless, even when identifying a collection of candidate applications, a valuable insight into network monitoring can be gained, and this can be achieved with high accuracy by all the methods considered. To facilitate further research in this area, a novel publicly available dataset of TLS communications has been created, with the communications annotated for popular desktop and mobile applications. Furthermore, the results of three different approaches to refine TLS traffic classification based on a combination of basic classifiers and context are presented. Finally, practical use cases are proposed, and future research directions are identified to further improve application identification methods.

Keywords TLS fingerprinting · JA4 · Encrypted traffic · Application identification · Machine learning

1 Introduction

Identifying and classifying network applications encrypted by Transport Layer Security (TLS) has become an increasingly challenging in modern networks. The widespread adoption of encryption protocols such as TLS has led to a significant proportion of network traffic being encrypted, rendering conventional monitoring tools less effective. As a result, network administrators struggle to gain visibility into network activity, making it difficult to detect security threats, enforce policies, and optimize network performance.

The complexity of identifying applications in TLS communications is due to several factors. First, encryption does not allow for content inspection previously used for application identification. Second, the new version of TLS uses encryption to protect parameters previously used for application identification, such as Server Name Indication (SNI) and certificates. To address these issues, several methods of classifying TLS traffic have been proposed:

- *TLS fingerprinting* relies on unique patterns in encrypted data to match connections with known application behavior. However, TLS fingerprinting often struggles with new or updated applications, where the patterns can be significantly different.
- *Server Name Identification* uses the server name available in the TLS handshake to identify the communicating application. However, the new proposal considers the use of Encrypted Server Name Indication (ESNI) to increase user privacy. ESNI keeps the SNI secret by encrypting the SNI part of the client's hello message.
- *Machine learning classification* uses statistical models trained on various features of TLS traffic, such as negotiated security parameters, to predict the associated application. ML-based methods can adapt to new and

✉ Ivana Burgetová
burgetova@fit.vutbr.cz

Petr Matoušek
matousp@fit.vutbr.cz

Ondřej Ryšavý
rysavy@vutbr.cz

1 Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 61200 Brno, Czech Republic

changing traffic patterns, but require significant training data and computational resources.

The ultimate goal of these techniques is to accurately identify a network application associated with each TLS connection. Due to inherent challenges, it is not always possible to identify every single application. In such cases, an acceptable solution is to provide a (ranked) list of possible applications associated with the connection, allowing network administrators to make informed decisions based on likely candidates.

1.1 Contribution

This article is an extended version of our original work published at the CSNet conference [1], which was selected for journal publication. In response to the invitation, this version includes new content, with notable enhancements in both methodology and presentation. The main addition is a novel multi-level classification approach that combines the outputs of individual classifiers into an ensemble model. This ensemble improves overall performance by aggregating predictions to produce more accurate and robust results. Furthermore, several sections of the original paper have been revised and refined for improved clarity and completeness.

The key contributions of this extended work are as follows: (i) the introduction of a novel, annotated dataset of TLS communications collected from widely used desktop and mobile applications, designed to support further research on encrypted traffic classification; (ii) a comprehensive comparison of three distinct approaches for identifying encrypted application traffic—TLS fingerprinting, machine learning-based classification, and SNI matching—along with an evaluation of their respective accuracy, coverage, and practicality for deployment; (iii) an analysis of the challenges in TLS application detection, particularly those arising from shared TLS properties across different applications, which hinder precise classification; and (iv) a discussion of real-world use cases and future directions to enhance the performance and applicability of these techniques.

1.2 Structure of the paper

This article is structured as follows: Sect. 2 presents related work, providing an overview of previous studies dealing with the identification of encrypted traffic. Section 3 describes the principles of TLS encryption, explaining the key features of the TLS handshake and how they are used for TLS fingerprinting. Section 5 describes the experimental setup, including the process of creating new datasets and the methodology used to evaluate the different identification methods. Section 6 discusses the experiments and results, providing a comprehensive analysis and addressing practical considerations. Section 7 suggests how the individual methods can be combined to achieve better accuracy and presents the experimental results. Section 8 presents the main observations from our experiments and reflects on deployment issues. The final section summarizes the article with concluding remarks and suggestions for future research.

2 Related work

Classification of encrypted traffic and the identification of network applications have been researched since the widespread adoption of encrypted communication protocols. This section gives an overview of the major work on identification methods, primarily TLS fingerprinting, machine learning, and neural networks, which have received significant attention recently.

The use of TLS fingerprints for malware detection in encrypted traffic was addressed by Anderson et al. in [2]. The authors extracted the cipher suites, the TLS extensions, and the length of the client's public key from the TLS client/server hello records. In addition to the TLS information (TLS), they observed the flow statistics (META), the sequence of packet lengths and inter-arrival times (SPLT), the byte distribution (BD), and the server certificate if it is self-signed (SS). By combining these data features on a large dataset of malicious and legitimate TLS flows, they were able to achieve 99.6% accuracy in classifying malware. When using TLS attributes alone, the accuracy ranged from 63 to 100%. Unfortunately, the proposed method was applied to private datasets, limiting comparison with other approaches. The TLS attributes were only obtained from the client hello, which is different from our approach.

Machine learning detection of encrypted malware communications was also investigated by De Lucia and Cotton [3]. The authors applied a support vector machine (SVM) and a convolutional neural network (CNN) to the streams extracted from captured TLS connections. As features, they used record size, type, and direction. Their results show high accuracy, but they do not discuss important issues such as similarity of TLS features and overlap of malware families. In contrast to their approach, we focus on identifying network applications in encrypted traffic rather than malware families.

The application of machine learning and deep learning to the identification of encrypted traffic was also explored by Barut et al. [4]. The authors combined flow metadata (port numbers, payload size, bytes transferred) with TLS features to classify encrypted application traffic. They used random forest and k-NN classifiers to select the features. However, the best RF classifier selected the source port number as the most important feature, which is a value randomly generated by the operating system. Therefore, its use for classification is questionable. For TLS features, both statistical data (the number of cipher suites, extensions, key lengths, etc.) and pre-processed lists of cipher suites with extensions were used. To use the CNN classifier, the authors transformed the input data to overcome the bias due to the imbalance of the dataset. Their conclusion showed the importance of TLS cipher suites for application identification, which is a part of TLS fingerprints.

In [5], Anderson and McGrew examined the evolution of TLS usage in applications over time. They tracked the use of different versions of TLS, cipher suites, and extensions and collected session data such as associated processes, destination IP addresses, and ports. They clustered similar fingerprints using the Levenshtein distance. Although their work focused on general TLS trends, our aim was to address application identification.

Fingerprint overlap was discussed by Anderson and McGrew in [6]. They extended TLS fingerprinting to include destination address, port, and SNI so that their fingerprints were more accurate using the destination context. This was similar to server fingerprinting: JA3S and JA4S hashes. We also use the server attributes along with the SNI values. The authors measured the similarity between fingerprints using the Levenshtein distance and added weights to the attributes based on the information gain ratio. Rather than identifying family of applications as in [6], our work attempts to identify individual applications where possible. In the case of shared fingerprints, we compute the most likely one or present a set of matching applications.

Our article extends the previous work of Matousek et al. [7], which investigated the reliability of TLS fingerprints for mobile application identification. We add JA4 fingerprints with ML techniques and SNI matching to highlight their advantages and limitations.

Another popular direction for identifying encrypted traffic is AI techniques such as deep learning, which uses CNN [8, 9] or RNN classifiers [10]. Unlike TLS fingerprinting or machine learning, deep learning approaches work with the full payload, i.e., they encode incoming packet payloads into vectors or images that are then used for training and testing. Although the authors claim to achieve high levels of accuracy, they ignore many important issues related to the nature of the encrypted data. First, they treat the input data as uninterpreted, without distinguishing between a packet header and a payload. This has a significant impact on the stability of the model, as encrypted traffic typically has a different payload distribution depending on the negotiated algorithm, e.g., HTTPS transmissions between the same hosts would have different characteristics if a different encryption algorithm were chosen. In addition, different applications transmit different amounts of data, e.g., an encrypted file transfer has much more data than an encrypted email. This omission results in an unbalanced training dataset that needs to be artificially normalized. Another problem with modeling full packet payloads is the huge amount of processing data. For example, the popular ISCX dataset contains 21 GB of data (including payload), but only 2436 TLS connections. Therefore, storing and processing the full payload is not feasible for real-world networks, making the TLS-based methods more preferable for practical use.

3 TLS encryption

Transport Layer Security (TLS) [11, 12] is a protocol defined on top of the transport layer that provides encryption, data integrity, and authentication for application protocols. Typically, TLS is implemented on top of TCP. After establishing a TCP connection, the TLS client and the server perform a TLS handshake in which they negotiate security parameters to establish a secure TLS channel; see Fig. 1. The TLS

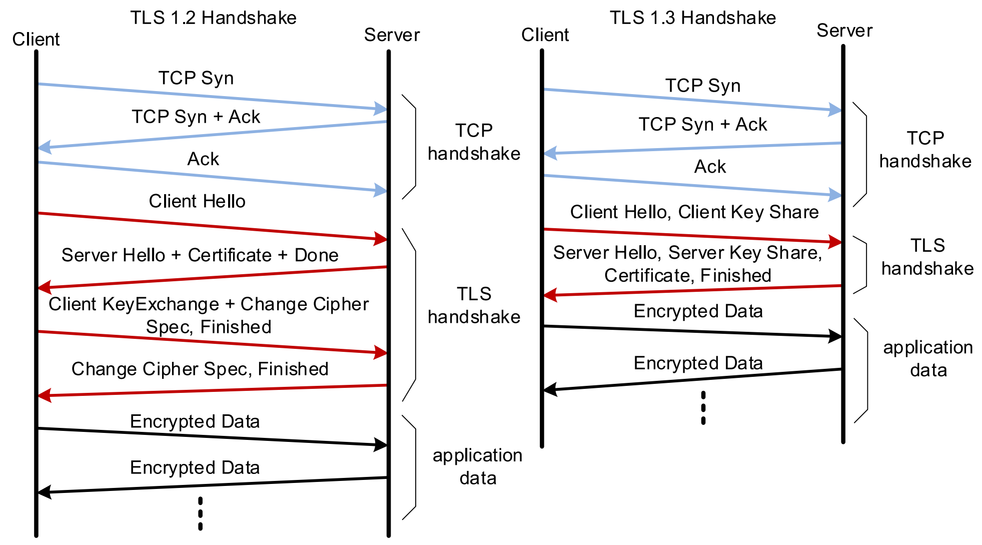

Fig. 1 TLS handshake version 1.2 and 1.3

handshake is an essential part of the TLS fingerprint, specifically the client hello and server hello packets, which contain a list of cipher suites, extensions, and other parameters supported by the client and the server. TLS defines a large number of possible parameter values. Their combinations represent a distinctive feature of the client or server that is used for application fingerprinting. The TLS handshake data is sent unencrypted. Once the TLS handshake is complete, the subsequent packets between the client and the server are encrypted.

We mentioned above that TLS is typically implemented over TCP. This is true, but TLS can also be implemented over UDP when it is part of the QUIC protocol [13, 14]. QUIC (Quick UDP Internet Connection) is a connection-oriented client–server protocol originally developed by Google. It implements its own handshake to negotiate cryptographic and transport parameters before establishing a connection. QUIC integrates the TLS handshake into the CRYPTO frames. It only supports TLS version 1.3 and higher. Thus, extracted TLS parameters can also be used to create TLS fingerprints for applications encapsulated by QUIC.

3.1 TLS attributes

To identify encrypted applications, there are three sources of features for classification models: (a) TLS attributes extracted from TLS headers during the TLS handshake that form so called JA3 and JA4 fingerprints,[1] (b) metadata about the TLS flow (e.g., number of bytes and packets transmitted, duration; see, e.g., [15]), and (c) the full packet payload.

TLS fingerprints are hashes created from TLS Client Hello or Server Hello messages. Client fingerprints are JA3 and JA4, while server fingerprints are JA3S and JA4S. Each hash comprises different TLS attributes (see Table 1). These attributes are concatenated into one string, which is then hashed using MD5 or SHA-1 functions.

Because both TLS fingerprinting and machine learning approaches use TLS attributes obtained from the TLS protocol [11, 12], a brief description of these attributes is provided below.

- *Version* of the TLS handshake protocol.
- *A list of cipher suites* includes possible combinations of key exchange methods, authentication, encryption, and data integrity algorithms. Valid combinations are standardized by IANA.[2] The current IANA cipher suite list contains 351 different combinations. The list of cipher suites may contain random GREASE values [16] for client or server compatibility testing. These values introduce instability into TLS fingerprinting and are therefore excluded from the fingerprint calculation.
- *A list of extensions* defines additional TLS features. There are about 63 different extensions.
- *Supported groups (SG)*. This TLS extension specifies the named groups that the client supports for key exchange. They are ordered from most to least preferred.
- *Elliptic curve point format (EC format)* describes the encoding supported by the client for transmitting EC values.
- *Server Name Indication (SNI)* specifies a domain name of the server that the client is contacting [17]. The SNI is not part of the TLS fingerprint, but plays an important role in annotating the requested service. However, it is only useful for application identification when the client is contacting a fixed service, such as a weather forecast server. In the case of Web browsers, the SNI value changes with each new Web server requested.
- *Application Layer Protocol Negotiation (ALPN)*. When multiple application protocols are supported by a single server, the client and the server must negotiate an application protocol to be used for each connection [18].
- *Supported version* is a list of TLS versions supported by the client, ordered by preference [12].
- *Signature algorithm* is a list of supported hash algorithms used for signatures.

Table 1 TLS attributes used in JA3 and JA4 fingerprints

TLS Attribute	JA3	JA3S	JA4	JA4S
TLS/QUIC protocol			x	x
Handshake version	x	x	x	x
Cipher suites	x	x	x	x
Extensions	x	x	x	x
Supported groups	x			
EC format	x			
SNI				
ALPN			x	x
Supported versions			x	x
Signature algorithms			x	

Table 1 shows which TLS attributes are shared by different types of TLS fingerprints. The TLS cipher suites are ordered for JA4/S fingerprints, while JA3/S fingerprints keep the original order. This plays an important role for application identification.

3.2 Properties of TLS attributes

We have analyzed the importance of the TLS attributes using entropy, which represents the degree of uncertainty of the value in the full range of possible values. This means

[1] See https://blog.foxio.io/ja4+-network-fingerprinting

[2] See IANA TLS Parameters.

Table 2 Entropy of TLS features in the MDA/ISCX datasets

TLS attribute	Uniqueness		Emptiness (%)		Entropy	
	MDA	ISCX	MDA	ISCX	MDA	ISCX
TLS version	1	3	0	0	0	0.09
Client cipher suites (unsrt)	35	22	0	0	0.48	0.81
Client cipher suites (srt)	31	24	0	0	0.41	0.81
Client extensions (unsrt)	8202	28	0	0	0.73	0.79
Client extensions (srt)	59	23	0	0	0.57	0.79
EC format	2	3	1.78	6.09	0.22	0.46
SNI	731	116	0.03	22.56	0.86	0.70
ALPN	10	7	7.94	59.1	0.3	0.70
Client supported versions	48	1	10.08	100	0.83	0
Signature algorithms	17	8	0	1.7	0.45	0.68
Server cipher suites	11	22	0	0	0.62	0.71
Server extensions (unsrt)	53	40	0	0	0.52	0.58
Server supported versions	3	1	22.64	100	0.59	0

that attributes with higher entropy contribute more to the uniqueness of the fingerprint and help to better identify the application. Low entropy means that many TLS connections share the same attribute value, in which case the attribute does not contribute much to distinguishing applications.

The entropy of the attribute X is computed as the sum of the weighted probabilities of the occurrences of its unique values, i.e., $H(x) = -\sum_{x \in X} p(x). \log_n p(x)$ where x is a unique value of the attribute X, $p(x)$ is its probability and n is the number of unique values of attribute X.

Table 2 contains the entropy of TLS attributes in the MDA and ISCX datasets (see Sect. 4.4). The MDA dataset contains 21,301 TLS connections. After filtering out incomplete connections and connections to analytics and advertising servers, we obtained 16,427 connections from 77 different applications. The table shows the number of unique values for the attribute, the percentage of connections containing an empty value for the attribute, and the entropy. For comparison, we have also included values from the ISCX dataset with 1494 TLS connections and 16 applications.

As expected, the most important TLS attributes for fingerprinting are SNI, client extensions and server cipher suites for the MDA dataset, and client cipher suites, client extensions, and server cipher suites for the ISCX dataset.

We can see a difference between the MDA dataset created in 2024 and the ISCX dataset created in 2016. The numbers differ due to the removal of obsolete cipher suites and the addition of new extensions. In addition, no server- or client-supported versions were found in the ISCX.

The client-supported versions show a high entropy, but there are about 10% of TLS connections with the empty value for the MDA. The TLS version also contains only one value for the MDA, and no values in the ISCX datasets, making it useless for application detection.

As shown in the table, the entropy of TLS attributes changes over time. For example, the more important attributes in the older ISCX dataset are the client cipher suites and client extensions, while the new TLS versions captured in the MDA dataset contain more unique values in the SNI and client-supported version attributes.

4 Test environment

Our test environment included an Android emulator for mobile applications and a Windows-based virtual sandbox for desktop applications. Both tools captured network communications in PCAP files and annotated connections based on application processes. We focused on Android and Windows applications due to their market dominance, although applications from other environments can be similarly analyzed.

4.1 Emulating mobile apps

To generate TLS fingerprints for mobile applications, we developed a tool[3] that emulates app behavior using the Android Virtual Device (AVD). The tool automatically downloads the APK file of the target application, installs it on a virtual Android device, and initiates its execution using the ADB shell and the `monkey` command to simulate user interaction. During this emulation, the application typically establishes several connections to its backend servers. These connections are captured using `tshark`, and the resulting TLS traffic is parsed to extract client and server hello messages. From these messages, we generate JA3/S, JA4/S, and

[3] See https://hashapp.netology.sk [Sept 2024].

JA4X fingerprints, which are then stored in the fingerprint database. Moreover, the communication is capture into pcap files. This method was used to capture communication of 35 selected applications (see Appendix A).

4.2 Sandboxing Windows apps

TLS connections from desktop applications were captured by executing each application in a sandboxed Windows environment. All network traffic from the host was recorded, and application-specific connections were isolated by monitoring open sockets associated with running processes. This method enabled automatic labeling of each TLS connection with the corresponding process name. Using this approach, we constructed a dataset of TLS communications from 42 popular Windows applications, all of which are installable via the winget package manager (see Appendix B).

We also found that many desktop applications available in the Microsoft Store, such as Instagram, TikTok, Pinterest, and Facebook, are deployed as Progressive Web Apps (PWAs). PWAs are web applications that provide an app-like experience and can be installed on a device to run in a dedicated window without the traditional browser interface. Because PWAs run inside the web browser process, it is impossible to identify them by their process name. As a result, we focus exclusively on native Windows applications. Analysis of PWA-based applications has been reserved for future work.

4.3 ISCX2016 dataset

For an objective comparison, we used the publicly available dataset ISCXVPN2016[4] created by the Canadian Institute for Cybersecurity. The dataset contains annotated samples of network application communications such as web browser, email, chat, streaming, file transfer (see [19]). The full dataset contains 21 GB of captured communications, covering 2436 TLS connections from 16 different applications (see Appendix C).

[4] See https://www.unb.ca/cic/datasets/vpn.html [Sep 2024].

Table 3 Statistics of the MDA and ISCX2016 datasets

	Mobile MDA	Desktop MDA	ISCX2016
Total TLS connections	6227	15,074	2436
Complete connections	6133	15,047	2422
Filtered connections	4142	12,285	1494
Train part	3095	8144	1063
Test part	1047	4141	431
Number of apps	35	42	16

Table 4 Efficiency of different TLS fingerprints in the MDA dataset

Fingerprint type	Total	Uniqueness	Covered apps	Efficiency
JA3	8208	99.5%	67.5%	1.02
JA4	111	54.1%	41.6%	3.36
JA3S	77	44.2%	20.8%	4.53
JA4S	97	48.5%	27.3%	4.06
JA3+JA3S	8330	99.2%	80.5%	1.02
JA4+JA4S	264	66.7%	70.1%	2.16
JA3+JA4+JA3S+JA4S	8349	99.2%	84.4%	1.02
SNI	728	88.0%	89.6%	1.27

4.4 MDA dataset

The Mobile Desktop Applications (MDA) dataset is an annotated dataset created by our research team by emulating a mobile application and executing selected desktop applications in the Windows sandbox as described above. The dataset contains mostly the encrypted traffic[5] of communications in the form of PCAP files from mobile and desktop applications.

5 Test environment

Table 3 shows the number of TLS connections from MDA mobile and desktop applications compared to ISCX2016. Both datasets were cleaned by removing incomplete connections and filtering out advertising and tracking traffic. We have also computed basic statistics for the MDA dataset. Table 4 provides selected properties of individual fingerprints, their combinations, and the SNI value for mobile and desktop application connections. These properties evaluate different types of fingerprints and their combinations. We provide the total number of distinct values, the percentage of unique fingerprints, the percentage of applications covered by unique fingerprints, and the efficiency of the fingerprint.

The efficiency E expresses the average number of applications sharing the same fingerprint. It is calculated as follows: let $i = 1 \dots n$ be the number of unique fingerprints and the function $f(i)$ maps each fingerprint to an application. For example, if a fingerprint is assigned to two different applications, $f(i) = 2$. Then, the efficiency E of the fingerprint type is calculated as follows:

$$E = \frac{\sum_{i=1}^{n} f(i)}{n} \tag{1}$$

[5] See https://github.com/matousp/tls-fingerprinting [2024].

Table 4 shows that we found 8208 different JA3 fingerprints, of which 99.5% were unique, that is, used by only one application. The rest of the fingerprints were shared by multiple applications. However, JA3 fingerprints only cover 67.5% of the applications, and the rest of the applications do not have unique fingerprints. Although all features containing JA3 fingerprints are imbalanced, we did not apply any transformations to them because the fingerprints-based classification methods we used (Sect. 6.1) are not affected by imbalanced features. The efficiency of 1.02 means that on average, one fingerprint is used for 1.02 applications. Thus, the optimal fingerprint has a coverage of 100% and an efficiency of 1. From this point of view, the combinations of fingerprints or SNI value are more promising for application identification, so we focused on them in our experiments.

5.1 Evaluation methodology

Due to the number of shared fingerprints, as shown in Table 4, it is not easy to assign a specific application to each observed TLS connection. On the other hand, it is helpful for network monitoring to assign a small set of candidate applications. To address the issue of shared fingerprints, we use two different classification approaches:

- *Probabilistic classification*: Assigns each TLS connection to a single application—the one deemed most likely by the classifier.
- *Set-based classification*: Associates each TLS connection with a set of potential applications, based on the selected classification approach. The classification is considered correct if the true application is included in the resulting set.

Regardless of the specific classification method—whether it is fingerprinting or a machine learning approach based on a collection of binary classifiers (each trained to recognize a single application)—the output for each TLS connection may include zero, one, or multiple predicted applications.

To compare the performance of different classification methods, we use the overall *accuracy*, expressed as the percentage of connections falling into one of the following categories:

- *OK*: the application correctly identified
- *Error*: the application misclassified
- *Unknown*: the application not recognized by any classifier

These categories are interpreted slightly differently depending on the classification approach:

In probabilistic classification, each test instance is evaluated by classifiers that assign probability scores to all possible applications. The application with the highest score is selected as the predicted result. The classification is considered *OK* if this predicted application matches the true label, *Error* if the prediction is incorrect, and *Unknown* if none of the classifiers assigns a valid score, indicating that the application is not recognized.

In set-based classification, all classifiers are applied to each test instance to produce a set of matched applications. The result is classified as *OK* if the true application appears in the set, *Error* if the set contains only incorrect applications, and *Unknown* if no classifiers match the instance at all.

6 Experiments

We have performed experiments using different identification methods. All experiments follow the same processing steps as shown in Fig. 2. The input is the labeled dataset with TLS communication, which is filtered in the first step to remove known connections to public ad servers. The filtered dataset is divided into a training part and a test part. The training part is used to train ML-based detectors and to map fingerprints and SNIs to applications. Once the models are created, they are applied to the test data. While more statistically robust evaluation methods, such as repeated experiments with multiple random train/test splits, exist, we deliberately chose this fixed split approach to reflect a realistic deployment scenario. In practice, models are typically trained on historical data and then applied to classify current or future traffic. This setup mirrors real-world usage more closely than randomized cross-validation, making the evaluation more representative of practical performance expectations. During the evaluation, two types of classification (probabilistic and set-based) are considered as described in Section 5.1.

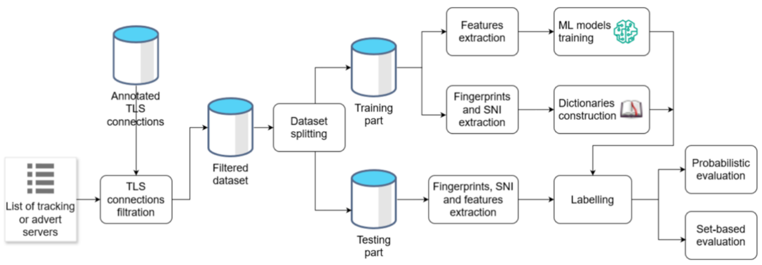
Annotated TLS connections
List of tracking or advert servers
TLS connections filtration
Filtered dataset
Dataset splitting
Training part
Testing part
Features extraction
ML models training
Fingerprints and SNI extraction
Dictionaries construction
Fingerprints, SNI and features extraction
Labelling
Probabilistic evaluation
Set-based evaluation

Fig. 2 The processing pipeline

Table 5 Accuracy of TLS connection classification with different classification types

Classification type	Mobile apps OK	Mobile apps Unknown	Mobile apps Error	Windows apps OK	Windows apps Unknown	Windows apps Error	All OK	All Unknown	All Error	ISCX2016 OK	ISCX2016 Unknown	ISCX2016 Error
Probabilistic classification												
JA3+JA3S	47.6%	13.7%	38.8%	30.4%	61.9%	7.7%	32.3%	52.2%	15.6%	79.4%	6.5%	14.4%
JA4+JA4S	54.4%	1.8%	43.7%	56.8%	0.4%	42.7%	54.0%	0.7%	43.4%	80.0%	6.3%	13.7%
JA3+JA3S+JA4+JA4S	49.2%	13.7%	37.2%	30.4%	61.9%	7.7%	32.6%	52.5%	15.2%	79.4%	6.3%	14.4%
ML-based	46.0%	28.6%	25.4%	47.0%	0.8%	52.2%	44.7%	4.3%	51.1%	77.4%	1.1%	21.4%
SNI	86.3%	5.9%	7.8%	75.0%	1.5%	23.5%	76.6%	1.9%	21.5%	85.9%	5.7%	8.4%
Set-based classification												
JA3+JA3S	79.9%	13.7%	6.5%	37.4%	61.9%	0.7%	46.0%	52.2%	1.9%	86.5%	6.3%	7.2%
JA4+JA4S	90.9%	1.8%	7.3%	98.7%	0.4%	0.9%	97.1%	0.7%	2.2%	86.5%	6.3%	7.2%
JA3+JA3S+JA4+JA4S	78.8%	13.7%	6.6%	37.4%	61.9%	0.7%	45.9%	52.2%	1.9%	86.5%	6.3%	7.2%
ML-based	67.3%	13.6%	19.1%	97.5%	0.2%	2.3%	91.9%	1.4%	6.7%	87.7%	2.5%	9.8%
SNI	88.3%	5.9%	5.8%	98.2%	1.5%	0.3%	96.2%	1.9%	1.9%	91.3%	5.7%	3.0%

6.1 Fingerprints classifiers

Our experiments evaluate the performance of different combinations of fingerprints for application identification. We compare the original version of the fingerprints (JA3+JA3S) with the newer version (JA4+JA4S). We also test the combination of all four fingerprints (JA3+JA3S+JA4+JA4S) (see Table 5).

We use a dictionary-based exact match method for classification. We created a dictionary of fingerprints from the training set and applied it to find the most likely application or a set of applications for each TLS connection in the test set. The fingerprints not seen in the training set were marked *Unknown*.

The experiments yielded some unexpected results. Although the basic properties described in Table 4 show a very high uniqueness and percentage of applications covered for the JA3 + JA3S combinations, its accuracy for connection classification is the lowest (see Table 5) (both probabilistic and set-based methods). This is more obvious for Windows applications, where JA3 fingerprints tend to be more unique due to the different order of the TLS client extensions. This high degree of uniqueness leads to a high percentage of unseen *Unknown* fingerprints in the test set. In this situation, the JA4 fingerprints work better because they use the client extensions differently.

In general, JA4 + J4S fingerprints perform better, but tend to form a larger cluster of associated applications, on average, three more applications than in the case of JA3 (see Table 6). The performance of the combination of all four fingerprints is comparable to the JA3 + J3S fingerprints, as the included JA3 fingerprints significantly increase the percentage of unseen fingerprints.

6.2 SNI classifier

For SNI classification, we use the same dictionary-based and exact match method as for fingerprinting. Unsurprisingly, this classifier achieves the best accuracy for most of our datasets. It is slightly outperformed by the set-based classifier, but at the cost of a higher average number of applications assigned to each connection.

6.3 ML-based classifiers

To facilitate comparison with fingerprinting methods, we also trained ML-based binary classifiers for application identification. A separate classifier was trained for each application, with the primary goal of evaluating whether ML algorithms could achieve greater accuracy in identifying applications. We deliberately did not perform advanced feature engineering or use flow metadata, relying only on the information available in the TLS handshakes. This allowed us to directly compare the accuracy of the ML-based detectors with JA3/JA4 fingerprinting.

The dataset was divided into training and test parts, following the same methodology used for fingerprinting to ensure

Table 6 Average number of predicted applications for one fingerprint (set-based classification)

Fingerprint type	Mobile	Desktop	All	ISCX2016
JA3+JA3S	5.46	2.27	3.85	2.09
JA4+JA4S	5.77	5.03	6.71	2.09
JA3+JA4+JA3S+JA4S	5.29	2.26	3.78	2.09
ML-based	1.79	3.22	3.55	1.57
SNI	1.17	2.12	2.00	1.77

Table 7 Features used for ML-based binary classifiers

No	Feature	Description
F1	TlsVersion	The version of the TLS protocol used during the connection
F2	TlsClientCipherSuites	An array of the cipher suites supported by the client
F3	TlsClientExtensionsSet	An ordered array of TLS extensions supported by the client
F4	TlsClientSupportedGroups	An array of supported elliptic curve groups by the client
F5	TlsClientAlpns	An array of ALPN protocols supported by the client
F6	TlsClientSupportedVersions	An array of TLS protocol versions supported by the client
F7	TlsClientSignatureAlgorithms	An array of signature algorithms supported by the client
F8	TlsServerExtensions	An array of TLS extensions accepted by the server
F9	TlsServerCipherSuite	The cipher suite selected by the server

comparable conditions across methods. Specifically, two-thirds of the captured TLS handshakes were used as training data, with the remaining one-third reserved for testing. All applications present in the test set were also included in the training set. This split reflects a realistic scenario where models are trained on historical data and then applied to classify future traffic; therefore, cross-validation techniques such as *k*-fold are not employed. For fingerprinting, repeating the experiment yields identical results due to the deterministic nature of the method. In contrast, for machine learning approaches, the reported results represent the best performance achieved after hyperparameter tuning. This procedure was applied consistently across all datasets.

6.3.1 Features and encoding

The input features represented categorical data,[6] which required appropriate encoding. We used a one-hot encoding, which, while potentially generating a large number of columns, avoids introducing unintended relationships between values. The input features are listed in Table 7. String values were encoded directly using one-hot encoding. Lists of strings were converted into a single string by concatenating the individual values in their original order. The combined string was then one-hot encoded. In the case of extension types, these were first converted to an ordered list and then concatenated into a single string using the same encoding process as for lists. We chose this approach after analyzing the dataset and finding that ordering the information kept the number of different values manageable without losing much information. We also remove the GREASE values from the list before encoding.

Encoding categorical data results in many boolean columns. Table 8 shows the size of the boolean vector after encoding categorical columns for each dataset. The total number represents the size of the input vector for classifier training, which is obtained by concatenating the individual one-hot encoded source features. The table also shows the size for each source feature after applying one-hot encoding.

As can be seen, the largest vectors in all datasets are the client and server extension features (F3 and F8), which contain the most diverse values within the source datasets. The total size of the feature vector varies slightly between the datasets, reflecting the difference between the Windows and mobile datasets and the older ISCX dataset. For example, in the mobile dataset, we observed only a single TlsVersion (F1), but a richer variation of extension values (F3 and F8).

We also calculated the correlation between the encoded features. The graph in Fig. 3 shows the distribution of Pearson's correlation coefficients between pairs of features in the datasets. The histogram shows a distribution of correlation coefficients between -1 and 1, with most values centered around 0, indicating that most pairs of features are uncorrelated. The low positive correlation is likely due to the nature of one-hot coding, as some categories may rarely co-occur, resulting in very low positive correlations between pairs of features that are activated together in some cases. The histogram also shows a tail towards correlation coefficients close to 1, meaning that some pairs of features are highly correlated. We further examined these correlations and found that specific client cipher suites are always used together with certain extension sets (F3), signature algorithms (F7), and supported groups (F4).

As part of feature selection, we evaluated two dimensionality reduction methods—principal component analysis (PCA) and autoencoder-based compression—by comparing classifier performance on original one-hot-encoded vec-

Table 8 Data dimensions for features F1 to F9

Dataset	Total	F1	F2	F3	F4	F5	F6	F7	F8	F9
Windows	136	3	17	33	10	3	3	17	42	8
Mobile	142	1	23	46	10	8	5	10	30	9
ICSX	123	3	20	25	4	6	1	7	37	20

[6] Although most attributes are numeric representing constant values (TLS versions, cipher suites), we treat them as strings because they have categorical rather than ordinal meaning.

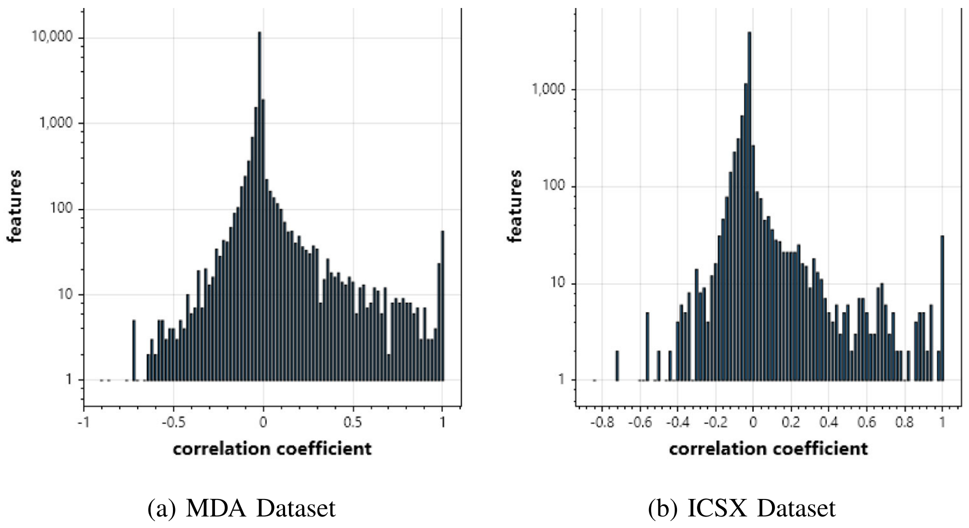

Fig. 3 Correlation among features

tors and their reduced forms. PCA projects data onto a lower-dimensional linear space to maximize variance, while autoencoders use neural networks to learn compact representations. However, neither method significantly improved classification metrics or reduced computation time. In fact, the autoencoder approach introduced substantial training overhead, increasing the overall computational cost.

The subsequent experiments were conducted with a nearly balanced dataset, meaning the ratio of positive to negative samples was close to 1. Nevertheless, Fig. 4 shows that similar results can be achieved with a lower ratio of 0.8.

6.3.2 Dataset balancing

For each application, there are significantly fewer positive class elements (connections of the application) because the connections of all other applications form a negative class. To address this issue, a random oversampling technique was applied. In this method, instances of the minority class (positive samples) are randomly duplicated and added to the training dataset until the class distribution becomes more balanced. This simple duplication increases the presence of

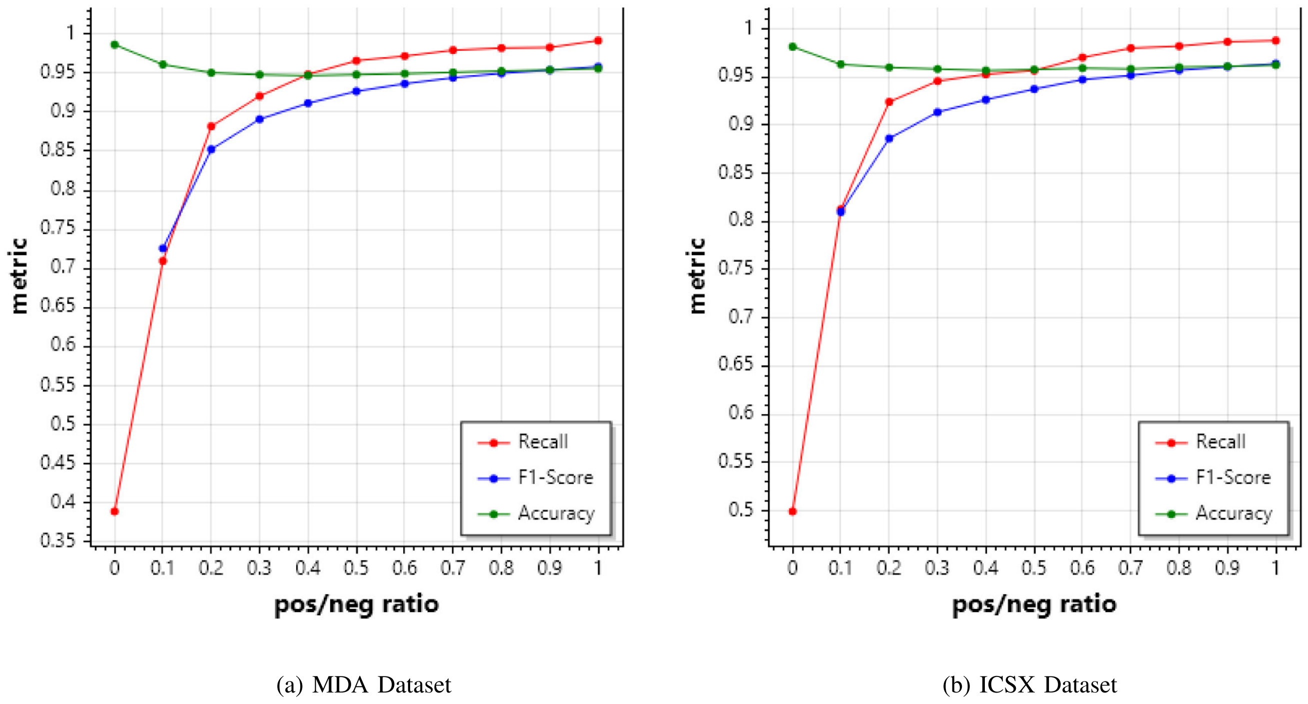

Fig. 4 Impact of the unbalanced dataset on the classification

the minority class during training, helping the classifier better learn its characteristics.[7]

The graphs in Fig. 4 illustrate the impact of dataset imbalance on key performance metrics of the trained classifiers. The presented values represent the mean measured values across all classifiers applied to the datasets. As shown, both the recall and the F1-score improve when the dataset is more balanced. As expected, the accuracy decreases slightly in the balanced case. This is because, in an unbalanced dataset, accuracy is often inflated—the classifier can predict the majority class correctly most of the time, while failing to detect instances of the minority class. In contrast, balancing the dataset forces the classifier to consider both classes more equally, providing a more realistic assessment of its true performance.

6.3.3 Application detectors

For each application, we trained a set of classifiers using a variety of machine learning algorithms provided by the ML.NET framework.[8] Prior to training, the input data was balanced with respect to the target application using the random oversampling technique described earlier, ensuring equal representation of positive and negative classes. The classifiers were trained on the training portion of both datasets, and model performance was evaluated using the F1-score, which provides a balanced measure of precision and recall. The best-performing model for each application was selected as its application-specific detector.

To optimize both model choice and configuration, we employed ML.NET's AutoML framework,[9] which supports joint model selection and hyperparameter tuning through randomized or grid-based search over predefined parameter spaces. This is crucial, as the values of hyperparameters—such as the number of leaves in decision trees, learning rate in boosting algorithms, or regularization strength in linear models—have a significant impact on classification performance. Each algorithm was wrapped in a SweepableEstimator, enabling AutoML to explore different configurations by evaluating multiple parameter combinations on the training set. The candidate algorithms included:

- **FastTree**: A gradient-boosted decision tree implementation optimized for speed and scalability. Hyperparameters such as learning rate, number of leaves, and number of iterations were explored.
- **LightGBM**: A high-performance gradient boosting algorithm that uses histogram-based learning. AutoML tuned the number of iterations, learning rate, number of leaves, and minimum data per leaf.
- **LBFGS Logistic Regression**: A linear classifier trained using the L-BFGS optimizer, with AutoML exploring L2 regularization parameters.
- **Field-aware Factorization Machines (FFM)**: A model that captures feature interactions in sparse data. Hyperparameters included the number of latent factors and the learning rate.

Each classifier was evaluated based on its F1-score, and the configuration yielding the best performance for a given application was selected. Consequently, different applications were associated with different optimal models. Among the evaluated classifiers, the random forest model was most frequently selected (177 times), followed by Field-aware Factorization Machines (4 times) and LBFGS Logistic Regression (5 times). The selected application detectors were subsequently validated using the methodology described in Sect. 5.1.

7 Multi-level classification

Individual base classifiers often struggle to achieve high accuracy on their own due to inherent limitations. However, combining these classifiers into an ensemble can improve performance by aggregating their outputs to produce a more accurate and robust model. In this section, we explore different ensemble methods for combining classifiers. Despite the improvements offered by this approach, challenges remain, particularly in scenarios where application fingerprints in TLS connections are not unique. Such fingerprints are shared by multiple applications, leading to potential misclassification. Contextual dictionary-based classification offers a solution by using information from surrounding TLS connections to improve predictions for ambiguous cases. By extending information from unique fingerprints to related connections, errors are reduced and classification accuracy is improved in complex datasets.

7.1 Classifier combination

The individual application classifiers described in Sect. 6 are taken as base classifiers and by combining them into an ensemble we aim to produce a more accurate and robust model. Three simple ensemble methods have been investigated:

Set intersection model: The set-based classifiers differ in their accuracy and the size of the candidate application sets they generate. Each classifier produces a set of potential appli-

[7] While more sophisticated methods such as SMOTENC could be considered, even this simple approach sufficiently improved the false negative rate.

[8] https://learn.microsoft.com/en-us/dotnet/machine-learning/mldotnet-api

[9] https://learn.microsoft.com/en-us/dotnet/machine-learning/automated-machine-learning-mlnet

Table 9 Accuracy of prediction for combined classifiers

Classification type	Mobile apps				Windows apps				All			
	OK	Unknown	Error	AvgApp	OK	Unknown	Error	AvgApp	OK	Unknown	Error	AvgApp
Set-based classification												
Intersection	83.1%	7.2%	9.7%	0.95	85.1%	1.0%	13.8%	1.14	84.0%	3.1%	12.8%	1.08
Majority voting	88.4%	0%	11.6%	1.15	86.1%	0%	13.9%	1.16	86.7%	0%	13.3%	1.17
Meta-model	85.0%	0%	15.0%	1.01	85.2%	0%	14.8%	1.11	84.6%	0%	15.4%	1.08

cations, and these sets can be refined by computing their intersection. By intersecting all the non-empty sets produced by each classifier, the resulting set of candidate applications is typically smaller, narrowing the focus without necessarily increasing predictive accuracy. This reduction simplifies the application discovery process, making the approach valuable in scenarios where fewer candidates are required for practical usability.

Majority voting model: This method aggregates predictions by selecting the application labels most frequently identified by all classifiers. Majority voting is a well-known ensemble method that relies on the consensus of the base classifiers to produce the final prediction. It effectively captures the dominant preferences of each model.

Meta-model: In this approach, the outputs of each set-based classifier are transformed into feature vectors that serve as input to a meta-model. The output set of each classifier is coded as follows: 0 indicates the absence of an application in the set; 1 indicates the presence of a single application in the set; and for sets containing multiple applications, each application is assigned a value of $1/n$, where n is the size of the set. These feature vectors are concatenated across all classifiers to form the input to the meta-model. The meta-model, employing LightGBM algorithm and trained using supervised learning on labeled data, learns to map these inputs to the correct application labels. This approach is similar to stacking, a common ensemble method in which a meta-model integrates the predictions of multiple base models.

The results of the classifier combination methods are given in Table 9. It can be seen that all the models give fairly similar results. Considering the differences, the *set intersection model* is best at reducing the number of candidate applications, making it suitable for tasks requiring a narrower focus. The *majority voting model* achieves the highest accuracy for mobile applications while maintaining a reasonable error rate. The *meta-model* provides a robust alternative by integrating the strengths of the basic classifiers through stacking, offering competitive accuracy with a slightly higher error rate. While the models have comparable performance in terms of accuracy, the choice of method depends on the specific requirements of the application, such as minimizing candidate sets or optimizing accuracy.

7.2 Contextual dictionary-based classification

Another way to improve the accuracy of detecting network applications in TLS traffic is to observe not only a single TLS connection, but also its neighbors. The idea behind this is that an application typically opens multiple TLS connections that are related and share common properties. This means that successfully identifying one TLS connection helps to identify related connections.

The classification based on the context of the surrounding TLS connections provides more accurate results. The information about running applications obtained by the unique fingerprints (occurring only for one application) can be spread to the surrounding TLS connection to help select the correct application for ambiguous fingerprints. We implemented this context idea as follows:

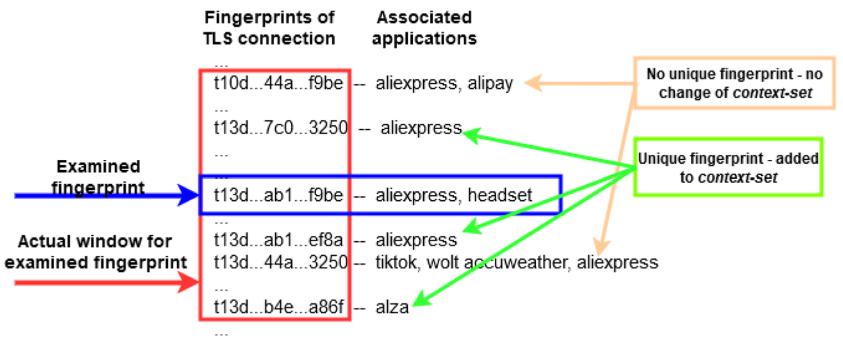
Fingerprints of TLS connection
Associated applications
t10d...44a...f9be -- aliexpress, alipay
t13d...7c0...3250 -- aliexpress
t13d...ab1...f9be -- aliexpress, headset
t13d...ab1...ef8a -- aliexpress
t13d...44a...3250 -- tiktok, wolt accuweather, aliexpress
t13d...b4e...a86f -- alza
No unique fingerprint - no change of context-set
Unique fingerprint - added to context-set
Examined fingerprint
Actual window for examined fingerprint

Fig. 5 Context-set generation

Table 10 Accuracy of prediction for context-based classification, with the best achieved values in bold

	Windows size					
Fingerprint type	11	21	31	41	51	61
Probabilistic classification						
JA3+JA3S	35.7%	35.9%	35.9%	36.2%	**36.4%**	36.2%
JA4+JA4S	64.6%	66.3%	**66.5%**	66.3%	65.6%	64.6%
JA3+JA4+JA3S+JA4S	35.9%	36.1%	36.1%	36.3%	**36.5%**	36.4%
SNI	**85.1%**	84.7%	84.4%	84.3%	84.0%	83.9%
Set-based classification						
JA3+JA3S	44.2%	43.9%	43.4%	42.9%	42.2%	41.3%
JA4+JA4S	93.7%	90.1%	88.1%	87.0%	85.6%	84.3%
JA3+JA4+JA3S+JA4S	44.3%	44.1%	43.6%	43.1%	42.5%	41.8%
SNI	95.3%	95.1%	95.0%	95.0%	94.8%	94.8%

1. For each TLS connection with an ambiguous fingerprint, we create a *context-set*. To create this set, we look for unique fingerprints in the neighborhood within the given time window and add associated applications to the context-set.
2. We find the intersection between the context-set and the set of applications associated with the ambiguous fingerprint.
3. If the intersection contains at least one application, we select the most likely application from this intersection (probabilistic classification) or return the entire intersection (set-based classification). If the intersection is empty, we cannot refine our prediction with the context and use the original set of applications stored in our dictionary to select the most likely application or the entire set of applications.

Figure 5 shows the generation of the context-set for the fingerprint in the blue rectangle. The context-set for this fingerprint contains two applications: AliExpress and Alza. Since the intersection with the set of associated applications contains only the aliexpress application, we can refine the results and select the most likely application in terms of the context of the given TLS connection.

Table 10 summarizes the results of the context-based classification for the MDA dataset for window sizes ranging from 11 to 61 with a step equal to 10. The addition of the context improved the results by 3.9 to 12.5% for the probabilistic classification. The highest improvement was achieved for the combination of JA4 and JA4S fingerprints for the window equal to 31 (accuracy increased by 12.5%). However, the accuracy of the set-based method is reduced by the addition of context. The results show that the larger the context window, the worse the accuracy, but the efficiency increases (see Table 11). In this case, the context can serve as a trade-off between accuracy and efficiency.

A limitation of the context-based approach is the requirement of handling TLS connections from individual devices independently. TLS connections can be sorted by source, so context-based classification is applicable. Furthermore, our context-based approach assigns a final application with respect to both previous and subsequent unique TLS connections. This introduces a small delay in the classification of individual connections when the user is actively using the application. In the case of termination of activity, a timeout can be used, after which only the previous connections are used for classification.

In general, adding context improves the classification accuracy but introduces additional computational complexity and delay, the extent of which depends on the size of the window. The optimal window size is not necessarily the one with the highest accuracy; rather, it should balance overall accuracy with the efficiency of the entire identification process.

Table 11 Average number of predicted applications for one fingerprint for context-based classification (set-based classification)

	Windows size					
Fingerprint type	11	21	31	41	51	61
JA3+JA3S	2.76	2.46	2.32	2.21	2.11	1.97
JA4+JA4S	4.5	3.67	3.28	3.15	3.06	3.00
JA3+JA4+JA3S+JA4S	2.72	2.41	2.28	2.18	2.10	1.99
SNI	1.42	1.39	1.40	1.40	1.40	1.40

8 Conclusion

8.1 Discussion

The aim of our experiments was to compare different approaches to identifying the application based on TLS features. Although the experiments were limited to the MDA and ISCX datasets, there are some interesting observations:

- The SNI approach provides consistent performance across the different datasets, supporting the observation that applications tend to use only a limited set of SNIs to communicate with their application servers. In some cases, the SNI is not available, or the applications contact the share services, making such connections ambiguous.
- In fingerprinting, the different combinations of JA3 and JA4 fingerprints give different levels of accuracy. The best solution is the combination of JA4 and JA4S, which proves the correct design of the hash calculation and supports the claim of the JA4+ authors about the usefulness of this fingerprinting method for identifying encrypted communications. The performance of the widely used JA3 fingerprint is low, suggesting that it is becoming obsolete. This is mainly due to the random order of TLS parameters, which creates multiple fingerprints for an application.
- Finally, we tried to build ML-based classifiers to identify applications. However, using only TLS attributes resulted in classifiers that performed worse than the previous methods (see Tables 5 and 9). When we analyzed the results, we found that the source data contained many overlapping samples (see Sect. 3.2), which negatively affected the metrics of the trained classifiers.

Despite the limited amount of data available for the experiments, it is apparent from the results that the traditional fingerprinting approach can achieve reasonable performance in the task of identifying the candidate application for the TLS connection. This is due to the fact that the method is based on the exact matching of known fingerprints. Due to the characteristics of the input data, the ML-based approach does not help in the case of previously unseen samples, and due to the overlapping samples, it is even worse overall than the JA4+JA4S fingerprinting.

8.2 Comparison with other studies

The main objective of this paper was to compare traditional TLS fingerprinting with ML-based methods and SNI matching for identifying network applications that communicate in TLS tunnels. This comparison was conducted using two datasets: the MDA and the ISCX 2016 (VPN-nonVPN) (see Table 5).

It is not easy to provide an objective comparison with other studies because of proprietary datasets or datasets containing statistical data (flows) that cannot be used for TLS fingerprinting. Therefore, we selected a few studies for the classification of network applications using the ISCX dataset (see Table 12).

The first column shows our results (F1-score) for the following three methods: JA4+JA4S, ML-based classification, and SNI matching. Results are shown for both probabilistic classification (one application selected) and set-based classification (a set of possible applications selected). The second column contains the results of Barut et al. [4], who tested three methods: random forest (RF), k-nearest neighbor (k-NN), and convolution neural networks (CNN). Their classifiers either worked with all available flow metadata (121 features) or just the top ten important features (e.g., source port, TLS cipher suite count). The third column includes the results of Bu et al. [20], who compared a network-in-network (NIN) classification model with a CNN. Depending on the number of NN layers, they used either a large or a small NN model. The final column shows the results of Zhou et al. [21], who compared traditional ML methods with their combined classifier.

As the results show, the accuracy of our TLS-based, ML-based, and SNI classifiers are comparable to that of others. The advantage of our approach is that we use data from the first two packets of the TLS handshake for classification, whereas flow-based statistical classifiers require data from the complete communication. Bu et al. [20] even analyze each individual byte of the packet which is extremely demanding.

Table 12 Comparison of TLS classification on the ISCX dataset

Our approach		Barut et al. [4]		Bu et al. [20]		Zhou et al. [21]	
Method	F1	Method	F1	Method	F1	Method	F1
JA4+JA4S prob	0.889	RF metadata	0.843	NIN_large	0.974	SVM	0.564
ML-based prob	0.873	RF top 10	0.929	CNN_large	0.97	RF	0.971
SNI prob	0.924	k-NN metadata	0.774	NIN_small	0.969	NN	0.925
JA4+JA4S set-based	0.928	k-NN top 10	0.893	CNN_small	0.961	Naive Bayes	0.769
ML-based set-based	0.934	1D CNN meta+TLS	0.633	CNN_deep_pkt	0.965	LR	0.794
SNI set-based	0.955	2D CNN meta+TLS	0.767			Combined	0.954

8.3 Deployment issues

While achieving high accuracy in identifying individual applications through encrypted traffic analysis is challenging—particularly due to the issue of shared fingerprints across multiple applications—identifying a set of likely candidate applications remains feasible and still valuable for network monitoring. Shared fingerprints reduce the discriminative power of TLS-based features alone, as different applications may exhibit similar handshake characteristics. To address this limitation and improve classification precision, incorporating additional contextual features such as IP addresses, port numbers, or flow-level metadata can be highly effective. For example, IP-based heuristics can help associate traffic with specific service providers or endpoints, reducing ambiguity when fingerprints are not unique. By combining these contextual signals with fingerprint-based classification, network administrators can more accurately narrow down potential applications and services, which is especially beneficial in complex, high-traffic environments.

In practice, all of the proposed methods require regular updates to the fingerprint database or retraining of the models. This ongoing maintenance task is non-trivial and requires significant resources, especially as applications are frequently updated and evolve. Ensuring that the fingerprinting method remains accurate and up-to-date is critical to maintaining its effectiveness in identifying encrypted traffic. Regular updates require a robust ecosystem similar to that used for antivirus updates or IDS signatures.

All of the methods evaluated are based on TLS connection information, which can be achieved in extended flow-based monitoring environments through approaches such as IPFIX [22], where network probes analyze and extract selected information from TLS handshakes. Once this information is captured, any of the presented identification methods can be effectively applied. Although the machine learning approach is more computationally intensive during training, the performance of all the methods discussed is efficient enough to allow real-time deployment in operational environments. This makes them suitable for use in live network monitoring systems where timely detection of application traffic is critical for security monitoring.

Identifying network applications is one of the most requested features of network monitoring tools. We have tested three methods based on fingerprinting techniques such as JA3/JA4, SNI-to-application mapping, and ML-based classification. Our experiments showed that accurate identification of individual applications is challenging. However, identifying a set of possible applications provides administrators with valuable insight. Identification of transmitted applications can serve several purposes, such as for network management (monitoring of frequently used applications), cyber security (detecting transmitted malware), and security policy (detecting forbidden applications, e.g., TikTok and WeChat).

When an application is properly identified in the network traffic, based on the security policy, an alarm can be raised or communication can be dropped by the IDS system.

Among the fingerprinting methods, the combination of client and server fingerprints proved to be effective in distinguishing between a variety of applications, with the JA4 and JA4S methods achieving accuracy rates above 90%. In contrast, older JA3/JA3S fingerprints showed lower performance, mainly due to limitations in the way their hashes are computed, rendering them inapplicable. The accuracy of ML-based detectors varies with the input data set.

Further, feature engineering and the use of larger datasets can improve performance. However, ML-based classifiers offer the advantage of reducing the number of unknown results, which is an inherent advantage of their underlying principles compared to traditional fingerprinting techniques. For comparison, we also evaluated the SNI-based method, which takes advantage of the fact that applications often use unique SNI values to identify their servers. As expected, the SNI method gave the best results for single-application identification tasks.

8.4 Future work

In the future, we plan to test the methods on a wider range of datasets, including those with different network environments and applications, to provide a more comprehensive evaluation.

Future work may include the development of more sophisticated methods for annotating connections in real-world scenarios that capture a wider range of application activities and interactions. This would result in richer datasets that more accurately reflect actual patterns. The ML-based classification approach leaves significant room for improvement, particularly through the use of advanced feature engineering techniques that better capture the nuances in the TLS characteristics of applications. In addition, alternative techniques to improve classification accuracy can be explored to address the issue of class imbalance.

Finally, while the methods have shown promise in controlled environments, their effectiveness in real-world operational networks remains to be thoroughly evaluated. Deploying these techniques in live environments would provide valuable insights into their practical utility, including how well they handle diverse traffic patterns, background noise, and incomplete or evolving data. Such deployment would also help assess the robustness of the models under realistic constraints, such as limited visibility, resource limitations, and the presence of previously unseen applications or obfuscation techniques. Evaluating the methods in operational settings is essential for validating their reliability, scalability, and overall readiness for integration into production network monitoring and threat detection systems.

Appendix A: The mobile applications

Application	Description
AccuWeather	Weather forecasting app with real-time alerts
AliExpress	Online retail platform for global shopping
Alipay	Mobile payment and digital wallet service
Alza	Shopping app for electronics and consumer goods
CAPCUT	Video editing app with social media integration
ChatGPT	AI chatbot interface by OpenAI
Discord	Voice, video, and text communication platform
Disney Plus	Streaming service for Disney-owned media
Facebook	Social networking platform for connecting with others
foodora	Food delivery and ordering service
Gmail	Email service by Google with Android integration
Google Play	Official app store for Android apps
Instagram	Photo and video sharing social network
Mapy-cz	Czech map and navigation app
Messenger	Facebook's standalone messaging app
Muj vlak	Czech railway travel planning app
Netflix	Streaming service for movies and TV shows
Packeta	Parcel tracking and delivery app
Reddit	Forum-based social news aggregation platform
RegioJet	Bus and train ticket booking app
Shein	Online fashion retail platform
Signal	Privacy-focused encrypted messaging app
Snapchat	Multimedia messaging app with temporary content
Spotify	Music streaming platform with personalized playlists
Telegram	Cloud-based instant messaging app
Temu	E-commerce app for affordable goods
Tiktok	Short-form video creation and sharing app
Trello	Project and task management tool
Twitter	Social media platform for microblogging
Viber	Messaging and VoIP app with encryption
Waze	Community-driven GPS navigation app
WeChat	Chinese multi-purpose messaging and social app
WhatsApp	End-to-end encrypted messaging platform
Wolt	Food and goods delivery service
Youtube	Online video sharing and streaming platform

Appendix B: The desktop applications

Application	Description
4K_Stogram	Windows desktop app for downloading photos and videos from Instagram
4K_Tokkit	Desktop software to download TikTok videos in bulk
ADAMANT_Messenger	Secure, decentralized messaging app with blockchain integration
AirDroid	Desktop app to manage Android devices remotely
Asana	Project management tool with a desktop interface for task tracking
Beeper	Unified chat app for integrating multiple messaging platforms
Brave	Privacy-focused web browser with built-in ad blocker
Caprine	Desktop wrapper for Facebook Messenger
Cozy_Drive	Cloud storage client with synchronization features for desktop
Deezer	Music streaming app with a desktop version
Electorrent	Remote control interface for qBittorrent clients
eM_Client	Full-featured email client with calendar and chat integration
Evernote	Note-taking app with a Windows desktop version
Google_Chrome	Popular web browser with cross-platform support
Headset	Desktop music player built for streaming from YouTube
Inssist	Instagram assistant and scheduler as a desktop wrapper
LINE	Messaging app with voice and video call support
Mailbird	Unified email client with customizable interface
MEGAsync	Desktop sync tool for the MEGA cloud service
Microsoft_Edge	Web browser developed by Microsoft
Mozilla_Firefox	Open-source web browser for privacy-conscious users
Mozilla_Thunderbird	Desktop email client developed by Mozilla
Nextcloud	Client for private cloud file storage
Notion	All-in-one productivity app with notes, tasks, and databases
Notion_Calendar	Time management and scheduling desktop app
Opera_Stable	Feature-rich web browser with built-in VPN
pCloud_Drive	Cloud storage client with virtual drive functionality
Proton_Drive	Encrypted cloud storage with desktop sync
Send_Anywhere	File transfer tool across devices and platforms
Signal	Privacy-focused messaging app with desktop client
Slack	Team collaboration tool with messaging and file sharing
Sonarr	TV series management and automation app

Application	Description
Spotify	Music streaming app with a native desktop client
TeamDrive	Secure team cloud collaboration software
TeraBox	Cloud storage service with a Windows sync app
TIDAL	High-fidelity music streaming desktop client
Trillian	Multi-protocol instant messaging client
Tweeten	Desktop Twitter client based on TweetDeck
Viber	Messaging and calling app with desktop support
Yandex_Messenger	Messaging client from Yandex with desktop support
Zoom_Workplace	Video conferencing app with chat and collaboration tools

Appendix C: The ISCX applications

Application/service	Description
AIM	AOL Instant Messenger, a legacy text and file messaging service
BitTorrent	Peer-to-peer protocol for distributing large amounts of data
Email	General category for sending and receiving messages (SMTP, IMAP, POP3)
Facebook	Social networking platform with messaging, video, and content sharing
FTPS	FTP over SSL/TLS for secure file transfer
Gmail	Web-based email service developed by Google
Hangout	Google's legacy messaging and video conferencing platform
ICQ	One of the earliest instant messaging services
Netflix	Video streaming service offering movies, series, and original content
SCP	Secure file transfer protocol using SSH
SFTP	SSH File Transfer Protocol for secure file transfer
Skype	Microsoft's VoIP and messaging application
Spotify	Music streaming service with personalized recommendations
Vimeo	Online video hosting and streaming platform for creators
VoipBuster	VoIP service for low-cost international calls

Acknowledgements The authors would like to thank to Kristián Kičinka for generating TLS fingerprints.

Author contribution P.M. wrote the main part of sections 2 and 3. O.R. and I.B. performed the experiments for Sections 5 and 6 and prepared the corresponding texts. All authors jointly prepared the texts for the remaining sections.

Funding Open access publishing supported by the institutions participating in the CzechELib Transformative Agreement. This work was supported by project "PREACT: Privacy-respecting Explainable Assessment and Collection of Threats" (2024-2025), no. TM05000014, funded by the Technological Agency of the Czech Republic.

Data availability Generated dataset is available at https://github.com/matousp/tls-fingerprinting.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Burgetová I, Ryšavý O, Matoušek P (2024) Towards identification of network applications in encrypted traffic. In: 2024 8th Cyber Security in Networking Conference (CSNet). pp 213–221
2. Anderson B, Paul S, McGrew D (2018) Deciphering malware's use of TLS (without decryption). J Comput Virol Hack Tech
3. de Lucia MJ, Cotton C (2019) Detection of encrypted malicious network traffic using machine learning. In: 2019 IEEE military communications conference. pp 1–6
4. Barut O, Zhu R, Luo Y, Zhang T (2021) TLS encrypted application classification using machine learning with flow feature engineering. In: 10th In. Conf. on Communication and Network Security (ACM). pp 32–41
5. Anderson B, McGrew D (2019) TLS beyond the browser: combining end host and network data to understand application behavior. In: Proc. of the internet measurement conference. pp 379–392
6. Anderson B, McGrew DA (2020) Accurate TLS fingerprinting using destination context and knowledge bases. CoRR abs/2009.01939
7. Matoušek P, Burgetová I, Ryšavý O, Victor M (2021) On reliability of JA3 hashes for fingerprinting mobile applications. Digital forensics and cyber crime, vol 351. Springer, Cham, pp 1–22
8. Lotfollahi M, Zade RSH, Siavoshani MJ, Saberian M (2018) Deep packet: a novel approach for encrypted traffic classification using deep learning
9. Wang W, Zhu M, Wang J, Zeng X, Yang Z (2017) End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: Conf. on intelligence and security informatics. pp 43–48
10. Yao H, Liu C, Zhang P, Wu S, Jiang C, Yu S (2022) Identification of encrypted traffic through attention mechanism based long short term memory. IEEE Trans Big Data 8(1):241–252
11. Dierks T, Rescorla E (2008) The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF
12. Rescorla E (2018) The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF

13. Iyengar J, Thomson M (2021) QUIC: a UDP-based multiplexed and secure transport. RFC 9000, IETF
14. Thomson M, Tuner S (2021) Using TLS to Secure QUIC Transport. RFC 9001, IETF
15. Wang Z, Thing VL (2023) Feature mining for encrypted malicious traffic detection with deep learning and other ml algorithms. Comput Sec 128
16. Benjamin D (2020) Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS extensibility. RFC 8701, IETF
17. Eastlake D (2011) Transport Layer Security (TLS) extensions: extension definitions . RFC 6066, IETF
18. Friedl S, Popov A, Langley A, Stephan E (2014) Transport Layer Security (TLS) application-layer protocol negotiation extension. RFC 7301, IETF
19. Draper-Gil G, Lashkari AH, Mamun MSI, Ghorbani AA (2016) Characterization of encrypted and VPN traffic using time-related features. In: Information systems security and privacy. pp 407–414
20. Bu Z, Zhou B, Cheng P, Zhang K, Ling ZH (2020) Encrypted network traffic classification using deep and parallel network-in-network models. IEEE Access 8:132950–132959
21. Zhou K, Wang W, Wu C, Hu T (2020) Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks. ETRI J 42(3):311–323
22. Claise B, Trammel B, Aitken P (2013) Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, IETF