A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web

IMAN AKBARI, MOHAMMAD A. SALAHUDDIN, LENI VEN, NOURA LIMAM, and RAOUF BOUTABA, University of Waterloo, Canada
BERTRAND MATHIEU, STEPHANIE MOTEAU, and STEPHANE TUFFIN, Orange Labs, France

Traffic classification is essential in network management for operations ranging from capacity planning, performance monitoring, volumetry, and resource provisioning, to anomaly detection and security. Recently, it has become increasingly challenging with the widespread adoption of encryption in the Internet, e.g., as a de-facto in HTTP/2 and QUIC protocols. In the current state of encrypted traffic classification using Deep Learning (DL), we identify fundamental issues in the way it is typically approached. For instance, although complex DL models with millions of parameters are being used, these models implement a relatively simple logic based on certain header fields of the TLS handshake, limiting model robustness to future versions of encrypted protocols. Furthermore, encrypted traffic is often treated as any other raw input for DL, while crucial domain-specific considerations exist that are commonly ignored. In this paper, we design a novel feature engineering approach that generalizes well for encrypted web protocols, and develop a neural network architecture based on Stacked Long Short-Term Memory (LSTM) layers and Convolutional Neural Networks (CNN) that works very well with our feature design. We evaluate our approach on a real-world traffic dataset from a major ISP and Mobile Network Operator. We achieve an accuracy of 95% in service classification with less raw traffic and smaller number of parameters, out-performing a state-of-the-art method by nearly 50% fewer false classifications. We show that our DL model generalizes for different classification objectives and encrypted web protocols. We also evaluate our approach on a public QUIC dataset with finer and application-level granularity in labeling, achieving an overall accuracy of 99%.

CCS Concepts: • **Networks** → **Network management**; **Network measurement**; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Encrypted traffic classification; HTTP/2; QUIC; TLS; deep learning

ACM Reference Format:
Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam,, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2021. A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 1, Article 4 (March 2021), 26 pages. https://doi.org/10.1145/3447382

1 INTRODUCTION

Traffic classification is quintessential for network operators to perform a wide range of network operation and management activities. This includes capacity planning, security and intrusion

Authors' addresses: Iman Akbari, iakbariazirani@uwaterloo.ca; Mohammad A. Salahuddin, mohammad.salahuddin@uwaterloo.ca; Leni Ven, shwen@uwaterloo.ca; Noura Limam, noura.limam@uwaterloo.ca; Raouf Boutaba, rboutaba@uwaterloo.ca, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, Canada, N2L 3G1; Bertrand Mathieu, bertrand2.mathieu@orange.com; Stephanie Moteau, stephanie.moteau@orange.com; Stephane Tuffin, stephane.tuffin@orange.com, Orange Labs, 2 Avenue Pierre Marzin, Lannion, France, 22300 .

2476-1249/2021/3-ART4 $15.00
https://doi.org/10.1145/3447382

detection, quality of service (QoS), performance monitoring, volumetry, and resource provisioning, to name a few. For example, an enterprise network administrator or Internet service provider (ISP) may want to prioritize traffic for business critical services, identify unknown traffic for anomaly detection, or perform workload characterization for designing efficient resource management schemes to satisfy performance and resource requirements of diverse applications. Depending on the context, misclassification on a large scale may result in failure to deliver QoS guarantees, incur operational expenses, security breaches or even disruption in services.

To preserve the privacy of Internet end-users, encrypted communication between clients and servers is becoming the norm. Most prominent web-based services are now running over Hypertext Transfer Protocol Secure (HTTPS). Furthermore, to improve security and quality of experience (QoE) for end-users, new protocols (e.g., HTTP/2 [9] and QUIC [19]) have emerged, which overcome various limitations of HTTP/1.1. For example, around 32% of all HTTPS sessions with identifiable HTTP version use HTTP/2 as their underlying protocol (based on a real-world mobile traffic dataset, cf. Section 5.2.1). This suggests a high adoption rate of HTTP/2 in the Internet. However, HTTP/2 features, such as payload encryption, multiplexing and concurrency, resource prioritization, and server push, add to the complexity of traffic classification.

Machine Learning (ML) and Deep Learning (DL) have undoubtedly become evermore influential with their application extending over a wide range of domains, including traffic classification. Over the past two decades, a large body of literature has been created to harness the power of ML for different traffic classification objectives, such as *service-level* classification (i.e., classification into coarse-grained service categories, e.g., video streaming, chat and webmail), *application-level* classification (i.e., fine-grained classes, e.g., YouTube, Gmail and WhatsApp), QoS-level and malicious traffic detection. However, there are still various limitations that need to be addressed for its real-world, practical usage.

For instance, numerous works (e.g., [17, 25, 41]) in traffic classification pick their labels somehow arbitrarily, which are often inconsistent in granularity. For example, authors in [25] use labels such as Network Time Protocol (NTP), a protocol, and YouTube, an application, at the same time. This allows for gerrymandering the dataset to report higher accuracies, and at the same time it is not conducive to real-world traffic analysis use-cases. Some other works (e.g., [26, 28, 39, 43]) use datasets with a mixed set of protocols that are often easily distinguishable using header signatures. Realistically, there is no practical interest in DL-based traffic classification, when the classification can be achieved with high accuracy via deterministic non-ML approaches (e.g., header matching). We address this issue by using a dataset of real-world traffic (cf. Section 5.2.1) with consistent labeling at the service and application levels. We have made the dataset publicly available.

Furthermore, some works in traffic classification leverage clever techniques to guide the models based on expert domain-specific knowledge. For example, authors in [13] use the position of non-MTU-sized packets to finger-print content inside an HTTP/2 session. While these solutions might show performance advantages in the short-term, small variations in the protocol can jeopardize their entire approach. Thus, future research should move towards generic data-driven methods that can adapt to different protocols and different traffic classification objectives, by simply changing the training data. We discuss this issue further in Section 3.1.

More importantly, extensions such as the Server Name Indication (SNI) in Transport Layer Security (TLS) can essentially reveal the server's identity, allowing for trivial classification of many traffic flows based on the server name. It has been showcased [28] that current DL models trained for traffic classification rely heavily on these extensions, and the model performance degrades if they are removed. This has severe implications for the state-of-the-art methods. For one, expensive and complex models are being used to learn a relatively simple logic, similar to that of a server name to label look-up table, that can be implemented deterministically. We argue that the true power

of DL is exploited when the models actually learn about the *nature* of traffic in different classes, and identify complex patterns to distinguish between them. We realize this by occluding these extensions from the training data and eliminate the model's reliance on them, while leveraging a carefully crafted feature engineering approach (cf. Section 3.1 & Section 4.2).

In this paper, we leverage DL for service classification (e.g., video streaming, social media, web mail) with a focus on new encrypted web protocols, i.e., HTTP/2 and QUIC, and overcome the above limitations. Unlike many works in this area, we focus exclusively on encrypted web traffic, and explore the challenges of unleashing the full potential of DL to find complex patterns in traffic that are innate to each traffic class. We occlude parts of the input that allow the DL model to learn a lazy and unsophisticated logic, and instigate how encrypted traffic should be treated differently from general raw ML input, e.g., images. We also place emphasis on a well-generalizable feature set that can be utilized by a diverse variety of future works in encrypted web traffic classification. Our main contributions are:

- We propose a novel feature engineering approach for encrypted traffic classification that focuses on protocol-agnostic aspects of the encrypted web traffic. We leverage standard flow statistics, the traffic shape with respect to size, timing, and direction, along with raw bytes from the TLS handshake packets. This is in contrast to most DL approaches for traffic classification, where the full raw traffic is fed to the DL model. We justify why the proposed feature set is a better fit for classification of encrypted traffic.
- We develop a neural network architecture based on Convolutional Neural Network (CNN) and Stacked Long Short-Term Memory (LSTM) layers that is highly effective in leveraging the extracted features for distinguishing between different traffic classes. We exploit the full potential of DL in identifying and correlating useful traffic traits, while being lighter in the number of trainable parameters and less likely to overfit.
- We leverage real-world mobile traffic dataset from an ISP, and demonstrate that our approach has an edge over the state-of-the-art in service classification of encrypted web traffic. We achieve an average accuracy of over 95% for classification exclusively over HTTPS (i.e., HTTP/1.1 and HTTP/2 over TLS), outperforming [28] by a significant margin of nearly 50% fewer false classifications. We have made the corresponding pre-processed dataset available to the public.
- We showcase that our DL model generalizes for a finer classification granularity, i.e., application classification. We also show that our model adapts to a different encrypted web protocol, i.e., QUIC, by simply changing the training data. We achieve an accuracy of 97% in application-level classification and an accuracy of 99% on a public QUIC dataset [29].

The rest of the paper is organized as follows. In Section 2, we provide some background on the HTTP/2 and QUIC protocols, and the challenges they bring to traffic classification, as well as typical feature categories used by traffic classification approaches. We then review the literature that inspired our work. We delineate our methodology, feature design and neural network architecture in Section 3. In Section 4, we discuss data processing and our DL model training. We evaluate our approach in Section 5, providing insights into what affects its performance and its advantage over existing methods. We conclude in Section 6 with a brief summary of our work and instigate future research directions.

2 BACKGROUND

Traffic classification has attracted significant attention in the past two decades. In this section, we start by briefly introducing the HTTP/2 and QUIC protocols and the challenges they bring to traffic classification. We shed light on the broad categories of features used for traffic classification, followed by a review of the related works that have inspired this paper.

2.1 New Web Protocols: An Overview

2.1.1 HTTP/2. The HTTP protocol has become a fundamental component in most web-based services. Based on Google's SPDY [8] protocol, HTTP/2 is the successor of HTTP/1.1. and has been widely adopted by the biggest providers on the internet. It offers performance improvements and addresses various limitations in HTTP/1.1 with features including synchronous communication (i.e., pipelining), multiplexing of streams, server push, and binary message framing. In HTTP/2, the smallest unit of communication is a frame consisting of a header and a sequence of bytes depending on the frame type. This allows for header compression, as well as prioritization of objects by the web-browser, to avoid the head-of-line (HOL) blocking. Furthermore, HTTP/2 uses encryption as a de-facto standard. The protocol offers reduced latency, better bandwidth, and improved QoE. However, HTTP/2's multiplexing, compression, and encryption, introduce new challenges for traffic classification [12].

2.1.2 QUIC. QUIC is a transport-layer protocol, recently proposed by Google. It is already in use by most Google services, supported by most modern web-browsers, and submitted to IETF [20] for standardization. QUIC provides an alternative to TCP and brings the key exchange and encryption to the transport layer. By providing multiplexing and pushing congestion control to the user space, QUIC provides enhanced performance compared to HTTP over TCP and works hand-in-hand with HTTP/2 to address HOL blocking.[1] The significance of QUIC for traffic classification is two-fold: First, it reinforces the fact that future web communications will use more encryption, i.e., a larger proportion of each session's information will be encrypted. Second, similar to HTTP/2, enhancements such as multiplexing add complexity to traffic classification.

2.2 Deep Learning Architectures

CNN is a class of neural network model with the assumption of spatial invariance built into its architecture. Compared to a dense neural network, the invariance assumption massively cuts down the number of parameters in the model. A convolutional layer has a set of *filters*, which are convolved against the inputs to produce the outputs. Each filter has very few parameters compared to the number of inputs and outputs. Commonly, convolutional layers are interleaved with pooling layers, which reduces the dimension of the output by down-sampling.

A LSTM unit is a recurrent neural network, which is capable of remembering features from earlier positions in a sequence. LSTM is a common element in neural networks for natural language processing (NLP). Multiple LSTM units can be stacked together to improve the information flow between positions in the input sequence. An example of three stacked LSTMs is shown in Figure 1.

2.3 Related Works

Traffic classification using ML started in the early 2000s to distinguish between protocols (e.g., DNS, SMTP and HTTP) in a network trace. This involved lightweight classic ML models, such as Naïve Bayes and Decision Trees [40], typically trained on a set of flow statistics picked by domain experts as the model's input. Soon, the attention shifted toward more challenging traffic classification tasks, such as classification of encrypted Skype traffic [4, 11]. This is particularly interesting as it operates on non-standard port numbers. Deep Learning introduced new opportunities in traffic classification by making it possible to feed large fine-grained feature vectors such as raw traffic to models, as opposed to aggregated statistics over entire sessions that required manual feature extraction efforts. The capabilities of Multilayer Perceptron (MLP), Stacked Autoencoders (SAE), CNN, and LSTM for traffic classification have been extensively explored in recent literature [16, 18, 30, 39, 43].

[1]HTTP-over-QUIC is often referred to as HTTP/3, as the protocol is aware of the semantics used in HTTP/2 and binds them to the wire protocol. This name is used to underline the integration and separate it from the general QUIC protocol.

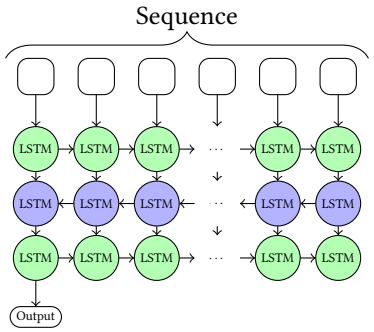


Fig. 1. Part of our proposed model uses three stacked LSTM's to process the flow information.

We can broadly categorize the typical features employed in traffic classification literature for modeling traffic into the following groups:

(1) **Flow Statistics:** A standard flow-meter, such as CICFlowMeter [22], yields the mean, standard deviation, minimum, maximum, of packet lengths, inter-arrival times, TCP flag counts, flow durations, number of packets, number of bytes, etc. These statistics constitute a feature vector for each flow and have been employed since the early traffic classification literature.
(2) **Raw Bytes:** The actual flow bytes from packet headers and payloads, which have grown in popularity with the advent of DL. Their appeal is leveraging data in the rawest form, as done in more conventional applications of DL such as computer vision.
(3) **Time-series:** Following a fixed-size, packet-level feature through all packets in a flow can yield a dynamic-sized time-series feature representing the flow. For example, the sizes of the packets in a flow is a valid time-series feature.

Lotfollahi et al. [26] use raw packet bytes (i.e., feature type 2), with certain adjustments in pre-processing, as the feature vector to CNN, and SAE. The resulting model is an automated fingerprinting approach as it tries to classify single packets and not the flows. They leverage a corpus of mixed-protocol ISCXVPN2016 dataset [22], where they achieve an accuracy of 97%. Bu et al. [15] extend the CNN with a MLP after each convolution (referred to as Network-in-Network or NIN) to boost its local abstraction capability. Furthermore, they leverage global average pooling rather than fully connected layers before final layers, to reduce model complexity. The authors process raw packet headers and payloads (i.e., feature type 2) through two parallel NINs, where the final classification is the weighted sum of the individual classifications. They achieve an F1-score (cf. Appendix A.2) of 98.3% and 98.5% for service- and application-level classification, respectively.

While the ISCX dataset has been widely used in the traffic classification literature, due to its mixed-protocol nature, it is a much more straightforward classification task than classifying homogeneous fully encrypted traffic, as the model can learn an easy header-matching logic. For instance, the traffic in the mail class includes SMTP and POP3 traffic, while the file transfer class includes SFTP and FTPS traffic, which are easy to distinguish based on their header patterns. Even models receiving packets individually and outside the context of the flow (e.g., [26]) can have good accuracies. In contrast, in TLS traffic classification, the performance of such models degrades, as there is simply not enough information in a single encrypted packet alone.

Aceto et al. [2, 3] evaluate numerous DL approaches for classification of mobile application traffic, using a proprietary dataset. They argue that there is no silver bullet, when it comes to the choice of a neural network for traffic classification. However, one-dimensional CNN and LSTM networks typically perform well due to the sequential nature of network traffic. Lopez-Martin et al. [25]

combine LSTM and CNN layers for service classification on a time-series (i.e., feature type 3), and achieve an accuracy of 96% on their labeled dataset. However, their classes are inconsistent in granularity and the model is essentially classifying protocols for some labels. This is a problem we discuss in Section 1. The authors also show that traditional methods (i.e., based on lightweight ML models) are inferior to DL models in accuracy, by a significant margin. The high accuracies reported for traditional models in other works often pertain to different classification tasks (e.g., QoS) or mixed-protocol datasets (e.g., [40]).

Liu et al. [23] explore the utility of representation layers and an "embedding layer" over the traffic flows in an initial unsupervised step. The authors use a stack of bi-directional Gated Recurrent Unit (GRU) layers connected to an encoder network. Autoencoder-based models have also been explored in other literature [3, 26]. However, it is difficult to identify a clear advantage in Autoencoder architectures over state-of-the-art CNN models in their evaluations. Therefore, we partly adopt the use of stacked recurrent units in our models, while proposing a lighter feature engineering that is more apt for general encrypted web traffic classification.

The authors' analogy between the initial encoding layer to embedding layers in NLP (i.e., used to motivate their approach) is inaccurate. The NLP embedding layers capture the semantics of how words of a corpus appear in correlation to each other and complex linguistic semantics, while encoding layers find compressed representations of individual flows. The exact transfer of NLP methodology to traffic classification has been done in [32], but the research in this area is in its infancy, and has not received much attention due to the restrictiveness of such features.

Wang et al. [38] combine CNN and LSTM layers for learning both spacial and temporal features of the traffic, with raw one-hot encoded bytes (i.e., feature type 2) making up the input to the model. Although the objective of their training is intrusion detection, the effectiveness of their method is still significant to service classification. The authors show that though they might be outperformed by rival methods in detecting specific attacks, they are the most consistent in detecting attacks across all categories, with a 99% accuracy on the ISCXIDS2012 dataset [33].

Anderson et al. [5] augment the typical features used in classic ML techniques (i.e., feature type 1), with: (1) values of certain fields from the TLS handshake headers, picked by a domain expert, and (2) packet length for the first fifty packets. Their evaluation shows a clear advantage of including these additional features, as the results of all models (i.e., Support Vector Machine, Linear Regression, Logistic Regression, Random Forest, Decision Trees, MLP) show higher accuracy at 0.001% False Discovery Rate (FDR)[2]. More recently [6], the authors propose a TLS client fingerprinting approach based on similar hand-picked features from the ClientHello message, and contextual information such as IP address, port number, and autonomous system. A weighted Naïve Bayes (NB) model is used for classification. While the approach is sound for client software fingerprinting, it is in contrast to our work which focuses on identifying regularities in traffic patterns rather than particular servers. Their approach relies on exclusively hand-picked features and data from all relevant clients. Moreover, while we appreciate the focus on simpler algorithms when they perform well, in our experiments lightweight traditional models, especially NB, perform very poorly for service classification. Lastly, in many use cases of service classification (e.g., volumetry), early detection, which is an advantage of the approach in [6], is not a high priority.

Schuster et al. [31] also bring the importance of traffic shape to spotlight, showing that even the actual traffic content can be finger-printed based on traffic shape. The authors model the traffic of Netflix, YouTube, and other streaming platforms over Dynamic Adaptive Streaming over HTTP (DASH) protocol as a time-series of down/up/all bytes-per-second, down/up/all packets-per-second,

[2]Accuracy at .001% FDR means the accuracy of the model, when it is only allowed one false positive for every 100,000 true positives.

and down/up/all average packet length. Using these features and a standard CNN, they identify the video being streamed with an accuracy of over 98%, on a dataset that constitutes up to a 100 titles. The authors show that by analyzing the "burstiness" of traffic at different points in time, the actual title being watched under encryption can be determined with high accuracy.

Bronzino et al. [14] explore classification and regression models for inferring important QoS metrics of encrypted video traffic. The authors make use of traditional ML models such as Linear Regression, Ridge Regression, Support Vector Regression, Decision Tree, and Random Forest (RF) regressors, as well as RF classifiers. They leverage a carefully crafted set of statistical features (i.e., feature type 1) to predict the target metrics, effectively achieving 93% precision for detecting video resolution. While this approach is effective in predicting particular metrics (i.e., playback startup delay and resolution), the feature engineering is tailored for a specific task and does not generalize to various traffic classification objectives.

Rezaei et al. [28, 29] leverage CNN and CNN-LSTM architectures with certain adjustments to achieve high performance. Their focus, like ours, is on encrypted web traffic such as HTTPS. However, their dataset also contains non-encrypted traffic. In [28], for their CNN-LSTM model, the authors model the traffic sessions as a series of flows. From each flow, the first six packets are fed raw to the flow-level model (i.e., feature type 2). Their dataset is comprised of real-world mobile traffic, labeled with applications and 45% of the flows use SSL or TLS. The authors report an overall accuracy of 94.22%, but the accuracy for HTTPS alone is 75.43%. In their post-hoc analysis, the authors identify the importance of different parts of the TLS headers to their model by masking different portions of the input and evaluating its impact on the model accuracy. They find that the model does in fact heavily fit to cipher info and the SNI field, to the point that the accuracy of the model drops to 38% when SNI records are occluded. We discussed this as one of the pitfalls of applying DL to traffic classification in Section 1. Due to its high relevance, we use [28] for comparison of our model to the state-of-the-art.

In [29], Rezaei et al. propose a semi-supervised approach to address scarcity of labeled data. In pre-training, the convolutional part of the CNN-LSTM model is trained on unlabeled data, with statistical features of traffic (i.e., feature type 1) used as the target of training. In the subsequent supervised step, the authors connect the pre-trained CNN to the LSTM part and perform the training on labeled data. They demonstrate the effectiveness of their approach on both a small QUIC dataset they have made public, and the Ariel dataset [27] for HTTPS with an accuracy of 96% and 84%, respectively. The authors show that without the pre-training step, their accuracy is lower.

3 SYSTEM ARCHITECTURE

In this section, we discuss our traffic engineering strategy that unifies the three categories of traffic features explained in Section 2.3 and is highly suitable for encrypted traffic classification. We also present our deep neural network model based on Stacked LSTM and convolutional layers, which is designed to work with our feature engineering and realize its potential.

3.1 Feature Engineering

Most works in DL-based traffic classification feed raw traffic bytes to the neural network model (cf. Section 2.3). Indeed, DL models are powerful enough to extract meaningful features from raw input on their own, provided a sufficiently large dataset. The notion of leveraging raw traffic bytes as model input is inspired from more conventional domains of DL, such as computer vision. Some works in traffic classification, such as Seq2Img [16], have gone as far as modeling the traffic as a two-dimensional image. However, as with the adoption of DL in any new domain, there are important considerations in traffic classification based on domain-specific knowledge of the task and the nature of data.

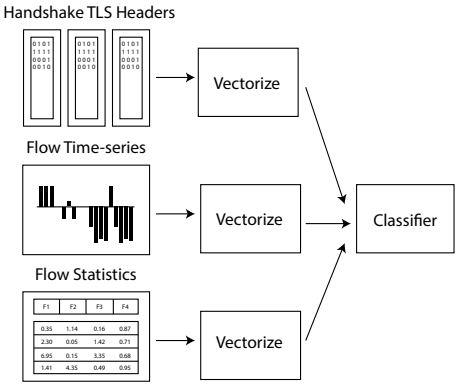


Fig. 2. TLS headers from the handshake, flow time-series, and standard flow statistics as the DL model input.

An important distinction between network traffic and images is encryption, which is becoming the norm in ordinary web usage. A traffic flow or packet is often almost completely encrypted, except for the initial handshake and some of the header fields that are transmitted in plain-text. Therefore, in the computer vision analogy, a traffic flow is like an image that is completely obfuscated except for a small area in it. Any effort to consume the encrypted portions of the traffic as the classification model input is essentially an attack on established encryption algorithms, such as Advanced Encryption Standard (AES), which is unrealistic.

Furthermore, it is crucial to consider what the DL model is exactly learning during training. For example, in an insightful post-hoc analysis, Rezaei et al. [28] show that the accuracy of their DL model completely degrades when the SNI field or TLS cipher info is masked. This implies that typical neural network models trained on raw traffic basically implement a look-up table which predicts a class based on the server's identity exposed by certain TLS extensions. We refer to the parts of the traffic that expose the server's identity as *canary features*. There are three major drawbacks in relying on canary features: (i) An expensive deep neural network is used for implementing a relatively simple logic, which can be performed deterministically with a very low computational overhead. (ii) The performance of the DL model is highly dependent on seeing large amounts of traffic from all relevant servers in a service category (e.g., traffic from all video streaming platforms) in training. In other words, the model is not really learning anything about the nature of video flows in general. (iii) The availability of these identifiers of the server (e.g., plain-text SNI field) in-the-clear is crucial for the utility of the DL approach. If the SNI field becomes outdated or encrypted in the future versions of TLS, which is not unlikely with the advent of Encrypted SNI, the entire DL method can lose its effectiveness.

Our input to the DL model combines all three types of features, described in Section 2.3. As summarized in Figure 2, it is comprised of: (i) handshake header bytes, (ii) flow time-series, and (iii) flow statistics.

First, we include raw bytes from the handshake in our input to the model. However, we remove the canary features such as SNI and cipher info in our pre-processing, to diminish the model's reliance on that information. Also, due to our focus on encrypted protocols, we assume that L5-7 payloads contain very little information as they are expected to be encrypted. Therefore, there is no utility in including entire packets in the DL model input and the aforementioned payloads only create more ways for the model to overfit. Besides, packets other than the handshake packets (i.e., ClientHello and ServerHello messages) are redundant and expose virtually no meaningful

information to the model. Thus, the raw traffic data for our DL model input, is truncated after the TLS headers of the handshake packets.

Secondly, we steer our DL model's focus on traffic aspects that are hardly affected by encryption. While the TLS records and extensions evolve over time and new encrypted protocols emerge with radically different characteristics, the traffic shape would always be available regardless of the underlying protocol. These kinds of traffic features can be quite effective for different traffic classification objectives, such as detecting a video being streamed using DASH [31] or distinguishing between malware and non-malware traffic [5]. We hypothesize that the traffic flow time-series of packet sizes, directions, and inter-arrival times (IATs) contain useful information for service classification as well. We discuss the flow time-series in more detail in Section 3.2.

Lastly, traditional flow statistics measured by standard flow-meters can also assist the model in traffic classification. Examples of traditional features include mean, standard-deviation, and median of packet sizes, number of different TCP flags, duration of the flow, etc. These features have been used for a variety traffic classification tasks for over two decades, and continue to be a simple yet powerful tool for distinguishing between different classes of traffic. This also allows for our overarching methodology to generalize for works such as [14], where a set of features are picked by domain experts for a particular traffic classification or regression task. Recall that Rezaei et al. [29] propose a semi-supervised traffic classification approach based on flow statistics, where the neuron weights from the unsupervised pre-training are used as a starting point for supervised learning. Their CNN model is initially trained on a large unlabeled dataset to compute standard statistical features. Subsequently, the pre-trained CNN is attached to an LSTM layer with time dimension over the different flows of a session. The new composite model is then trained on a smaller labeled dataset. While we appreciate the shift towards semi-supervised learning and the use of unlabeled datasets, steering the DL model towards statistical aggregations is not ideal to harness the full potential of DL. Their approach guides the DL model to extract meaningful statistical insights from the traffic. However, if the weights learnt in pre-training are expected to stay relatively constant, one can deterministically compute the statistical features used in pre-training and offer them as the DL model's input. Moreover, DL is effective in detecting much more nuanced patterns in traffic, i.e., patterns that cannot be easily aggregated into high-level scalar metrics of the flow.

Note that the combination of handshake features and flow time-series was first proposed by [7] to detect malicious traffic. Aside from the use of statistical features as a third input, another key distinction between our approach and [7] is the use of DL to extract useful features of the handshake, while they require a domain expert to cherry-pick them for the TLS protocol. Though our research is focused on encrypted web protocols and mostly revolves around TLS, our feature engineering methodology is protocol-agnostic. Regardless of a protocol's implementation details, it is expected to have a negotiation or handshake segment, while the rest of the traffic would be fully encrypted. This segment will make up the only raw inputs to the model. The flow time-series, i.e., traffic shape and timing, as well as flow statistics will always be available in IP. Therefore, the model will have to be retrained and specialized for new protocol versions as they evolve, but our overarching feature engineering methodology will still be applicable.

3.2 Flow Time-series

Previously, we discussed the drawbacks of relying only on raw traffic bytes for traffic classification, which include learning over-simplified logic and poor transferability to future use-cases. Indeed, the goal of employing highly intricate DL models for traffic classification is to find distinctive and complex patterns pertaining to the nature of a traffic class. Therefore, we divert our attention to the aspects of traffic that are not used to identify a known server, but are rather innate to the service class itself.

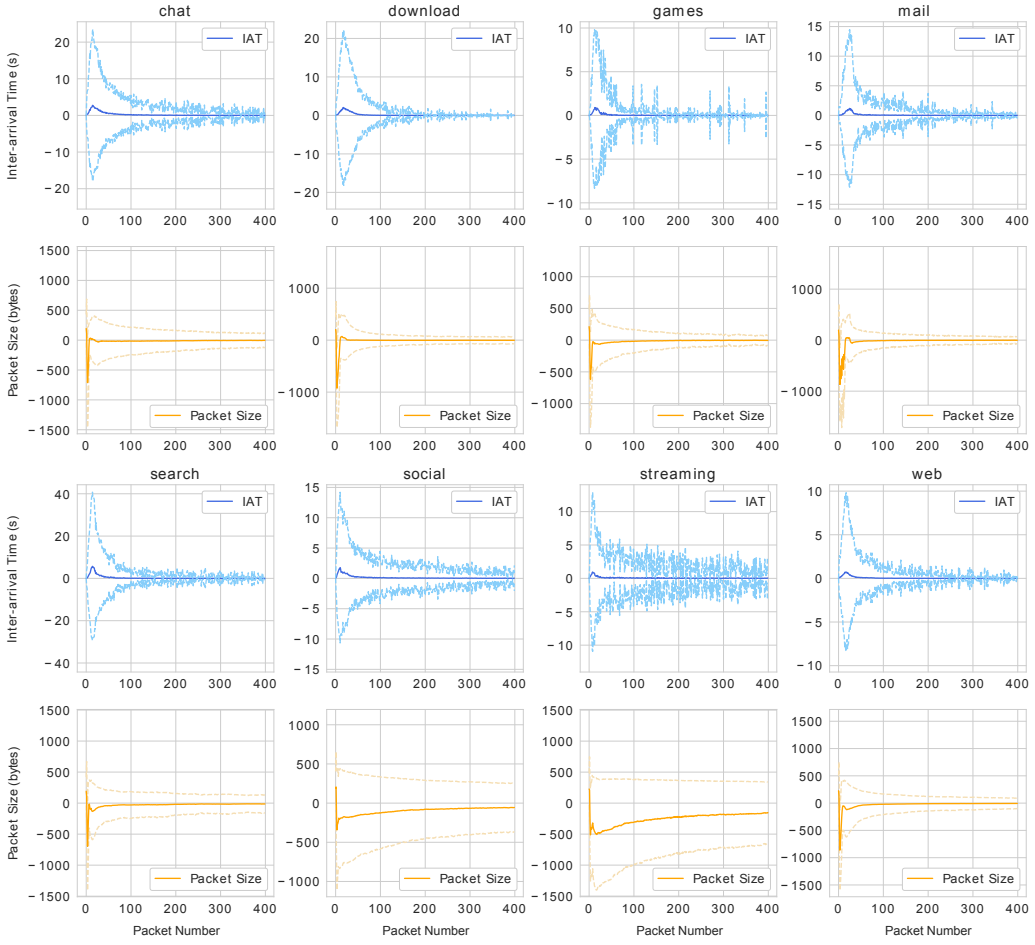


Fig. 3. The average of flow time-series extracted from the Orange'20 dataset. The blue lines follow the IATs of the packets in the traffic flows. The orange lines show the size of the packets sent in each direction, where negative values indicate packets from flow's destination to its origin. The dashed lines delineate error bars of one standard deviation for each graph.

One advantage of relying on the flow time-series features (cf. Section 3.1), is their relative independence from the implementation details of the protocol. While different extensions may be added to and removed from the protocol making it more difficult to classify traffic, characteristics such as the traffic shape and timing of the packets will always be present, unless a protocol is intentionally designed to obfuscate such information. Though this is a possibility, designing such obfuscation strategies would have a negative impact on bandwidth, latency, and QoS, as it entails sending redundant traffic or delaying packets in order to manipulate the time-series. Therefore, it is unrealistic that there would be enough motivation for introducing such measures in ubiquitous web protocols.

Figure 3 shows the difference of flow time-series between various classes in our dataset (cf. Section 5.2.1), where the time-series of packet IATs and sizes are averaged for each class and the error

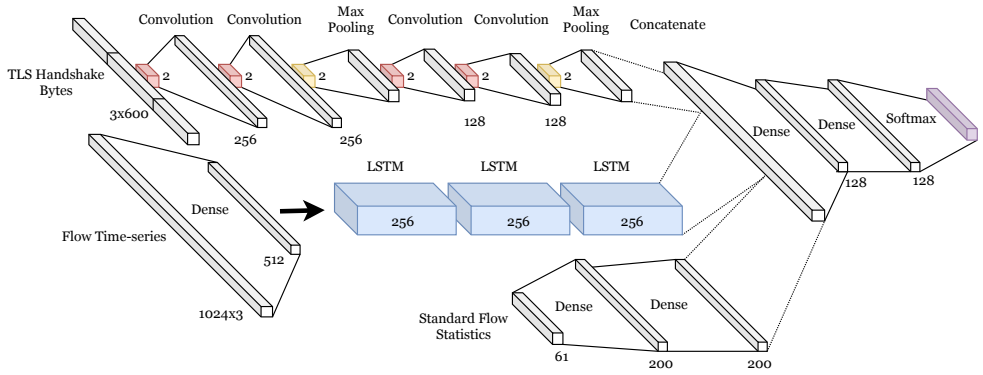


Fig. 4. Tripartite neural network architecture

is plotted with dashed lines. At a glance, it is easy to see subtle but visible differences between the traffic shapes of different classes. Note that the packet size time-series has a positive value when the packet goes from the flow's origin to its destination, and negative otherwise. In other words, we multiply the packet size by -1 when it's going from flow destination to flow origin. As revealed later in our experiments, deep neural networks (e.g., CNN and LSTM layers) are highly effective in detecting these nuanced patterns for each class, and traffic classification can be enhanced by including the time-series in the input. By combining these features with raw bytes, we can create a powerful feature set that can be used for learning the nature of traffic, as well as identifying useful parts of the secure protocol's headers for identifying applications.

3.3 Model Architecture

Our neural network architecture reflects the structure of the features presented in Section 3.1. As shown in Figure 4, our neural network model separately processes the flow time-series, the handshake TLS headers, and the standard flow statistics as input. Each of the three inputs is fed to a separate set of neural network layers, and the output of those layers is later concatenated and passed through additional fully-connected layers to produce the final prediction.

The raw handshake bytes are fed to a deep one-dimensional CNN with max-pooling layers in between. The structure of these layers is quite standard and a one-dimensional equivalent of commonly used computer vision models, which has proven effective on network traffic [2, 26, 28]. We only feed the first C bytes of up to three ClientHello and ServerHello packets from the flow to the model. In our evaluations in Section 5.4, it is proven that there is no real disadvantage in omitting the rest of the traffic, which has strong implications for future research in this area. The value for C in our experiments is 600, which is picked through a hyper-parameter search described in Section 5.4.1.

The flow time-series has three channels: (i) IAT, (ii) size, and (iii) direction. In our experiments, we found that a stacked LSTM architecture preceded by a dense layer, is extremely effective in processing the flow time-series features, while one-dimensional CNNs are also viable (cf. Section 5.4). In our implementation, we use a stack of three LSTM layers going through the flow time-series in both directions (cf. Section 2.2). We also include flow statistics extracted using CICFlowMeter [22]. Since these features do not have a natural ordering or sequentiality, a fully-connected network is used to ingest them.

One of the major advantages of our feature engineering is the ability to include information about a large number of packets without substantially increasing the model size. In a classic raw

input approach, it is normal to include the first b bytes of the first k packets of a flow to the model. The size of the input grows linearly by increasing k, which can create a super-linear increase in the number of model parameters and quickly lead to overfitting. In contrast, our model limits the raw traffic to the handshake packets, and uses a lightweight representation with only three channels for the other packets of the flow. This allows the model's scope to grow and consider hundreds of packets without a significant impact on its complexity. The outputs from these three parts (i.e., flow time-series and statistics, and TLS headers) are concatenated and passed through multiple additional dense layers, which yields the output of the network as a softmax layer.

Our early experiments showed that the models are highly prone to overfitting. This is not surprising considering the fact that the number of parameters in the model to be trained is in the order of millions. It is not uncommon for traffic classification models to be trained on datasets which have an order of magnitude less entries than the number of trainable parameters in the model. To overcome the problem of overfitting, we use very high drop-outs (i.e., up to 50% at some layers), especially in the final dense layers. As showcased in Section 5.4, our feature engineering itself has a tremendous effect in lowering the chance of overfitting when compared to the conventional raw traffic input.

4 LABELING & TRAINING

Pre-processing is a very important step in traffic classification, which is sometimes overlooked despite the fact that it can significantly impact the accuracy of the model. In this section, we describe how our data is gathered, labeled, and pre-processed for the training in practice.

4.1 Labeling based on Server Names

A primary challenge in the application of ML to network systems is the scarcity of labeled real-world packet traces. Due to privacy concerns, administrators are reluctant to publish real-world network data to the public. An alternative is to use synthetic datasets. However, they often fail to capture the true distribution and patterns of real-world network traffic, which significantly affects model generalization and makes their evaluations questionable. It is also generally difficult to label traffic at its origin, as it requires users to actively participate in the process and log their activities.

In contrast, raw unlabeled network traces are available in abundance to the service providers. We leverage the SNI field of TLS records to extract the server names. A look-up table is used to match server names to their respective classes. Each domain name from the SNI field is matched against a set of regular expressions that are either carefully handpicked (i.e., by monitoring the traffic of prominent websites and mobile apps, e.g., Netflix, YouTube, and AppStore), or gathered from a dataset of categorized domain names, such as the Blacklists UT1 dataset [1]. For certain providers (e.g., Google and Facebook) extra care must be taken, as similar sub-domains may be shared between multiple service categories. The granularity of classification can be set by simply modifying the look-up table. For instance, based on whether we would like to perform application-level or service-level classification, we can change how server names are mapped to labels.

4.2 Pre-processing

We design our pre-processing to be performed in a distributed fashion using Apache Spark. As shown in Figure 5, the following steps are taken in the pre-processing of the data:

(1) Extract flows (i.e., 5-tuples of src/dst IP/port and protocol) via standard flow-meters like YAF.
(2) Filter flows with TLS packets, as we are particularly interested in encrypted web traffic.
(3) Extract basic flow information, such as the flow start and end times, packet count, byte count, flow time-series, etc.

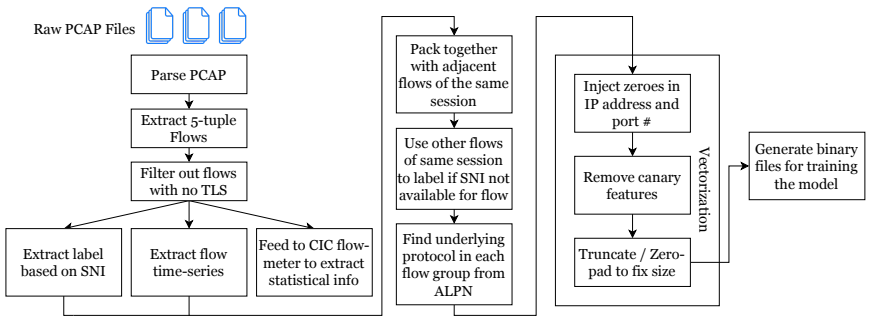


Fig. 5. Overview of the pre-processing steps

(4) Extract statistical features for the flow using CICFlowMeter, and store it as metadata.
(5) Extract SNI domain name and assign label based on a look-up table.
(6) Group flows from the same TLS session ID together. If TLS session ID does not exist, time proximity and NAT-aware source and destination IP & port numbers are used.
(7) For each unlabeled flow f, check other flows in the same session as f, and use their label for f. Often in multi-flow TLS sessions, only the first flow contains the SNI record.
(8) Vectorize the flow into a time-series of binary information as follows: (i) mask IP addresses by injecting zeroes even if they are already randomized, (ii) remove TLS cipher information (iii) mask the SNI record (iv) truncate to MTU size or zero-pad the packet bytes—ensure fixed vector size.
(9) Write raw traffic bytes to binary files, with each entry having an array of vectorized bytes from up to three handshake packets. Include flow statistics and a time-series of maximum length 1024 with the three channels for packet sizes, directions (±1), and inter-arrival times for each entry as well.

4.3 Training strategy

In addition to the model itself, the training strategy (i.e., the optimization process) has an impact on the end result as well. In our approach, we use an Adam optimizer [21] that performs stochastic gradient descent based on adaptive estimates of lower-order moments. We have learned that one effective strategy is to manually lower the learning rate as the training progresses.

Since the learning rate decides the granularity of the parameter search, there is an inherent trade-off in the choice for its value. Small values for the learning rate can lead to slow training and over-exploration of local optima in the loss function's optimization. Excessively high values for the learning rate can prevent enough granularity in the parameter search and lead to low accuracy in the final model, as we leap over optimal configurations.

By manually reducing the learning rate after every few epochs, we can balance this trade-off. Our strategy is to start from a default learning rate (e.g., 10^{-3}) and decimate the learning rate every 10 to 20 epochs when the overall validation loss starts to flatline. This way, we essentially increase the resolution of the search as the training progresses. The optimizer will start with making large leaps in the parameters and gradually decrease the step size in order to learn a highly optimal set of weights for the DL model.

Due to the imbalance in the number of dataset entries for each class, we also employ an up-sampling strategy, which increases the impact and weight of entries from smaller classes during

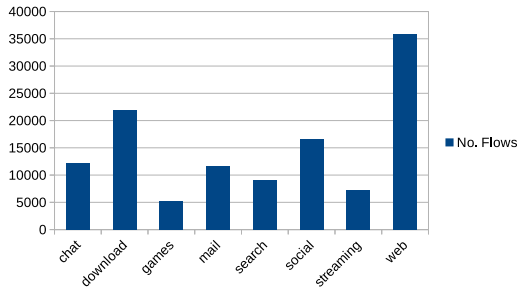


Fig. 6. Breakdown of the approximately-labeled portion of the ISP network dataset per class

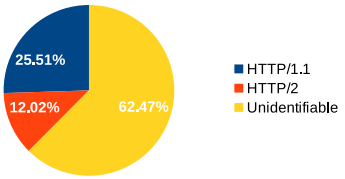


Fig. 7. The percentage of TLS sessions by the overlying protocol. Note that not all TLS sessions can be identified, as the ALPN and NPN records may not be available (cf. Section 5.4.2).

training, proportionate to the size of the classes. Although quite simple, the upsampling strategy considerably enhances the model's overall performance.

5 EVALUATION

5.1 System Specification

We conduct our experiments on a machine with one NVIDIA Tesla P40 GPU 24GB GRAM, 56 Intel(R) Xeon(R) Gold 5120 2.20GHz CPUs and 376 GBs of RAM. The software stack for training and pre-processing includes Centos 7, CUDA 10.1, Tensorflow 1.15 and PySpark 2.4.4. The code base is developed with Keras for convenience.

5.2 Dataset Description

5.2.1 Orange'20 Dataset. The primary dataset used in our work is provided by Orange S.A., a major ISP in Europe. The dataset was collected on July 11, 2019 for about 80 minutes, from the ISP's mobile network. For privacy concerns, the IP addresses are masked and the packet payloads are removed with the exception of TLS headers. The entire dataset has more than 800K unlabeled flows, where about ~300K are TLS flows and of interest to us. We use the SNI field to label the TLS flows (cf. Section 4.1) with the following service categories: (i) chat, (ii) download, (iii) games, (iv) mail, (v) search, (vi) social, (vii) streaming, and (viii) web. A total of 119,565 out of the 343,228 TLS flows are labeled, using our approximate labeling scheme, with both manually picked URLs and the UT1 dataset [1]. The distribution of service categories in the labeled dataset is shown in Figure 6. In the spirit of transparency and openness in research, we have released the pre-processed dataset (cf. Section 4.2) with certain adjustments for privacy compliance to the public.[3]

5.2.2 UC Davis QUIC Dataset. The QUIC dataset has been recently released by the authors in [29]. It comprises of 3,637 flows, classified into Google Docs, Google Drive, Google Music, Google Search and YouTube. This is natural as Google is currently the primary advocate for the QUIC protocol's adoption in the industry. The dataset is relatively balanced, with no class being twice as large as the others. Furthermore, it is partly generated by human users and partly via automated agents.

5.3 Evaluation Methodology

Evidently, the accuracy of a traffic classification model by itself can not be thought of as a global KPI for a model's strength, as the model performance can significantly vary from one dataset

[3]The dataset is available for download at http://bit.ly/UW-Orange-2020

to another. Hence, the significance of these results can only be verified when compared against existing models in the literature.

For comparison, we implement the CNN and CNN-LSTM models proposed by Rezaei et al. [28], which have shown good performance in application-level traffic classification. The authors model the input as a series of flows, which can be thought of as a *user session*. The CNN model operates on the *flow* level, i.e., the first 256 bytes of the first 6 packets of a single flow in the series are fed to a deep CNN. The CNN model is very similar to the header part of our model (cf. Figure 4). On the other hand, the CNN-LSTM model receives the session (i.e., a time-series of flows) as its input, and essentially makes the CNN model time-distributed over the flows of each session and labels the entire session. Similar architectures have been employed over the years for encrypted traffic classification [3, 39], making it an ideal baseline to compare our model against.

5.4 TLS Classification at Service-level

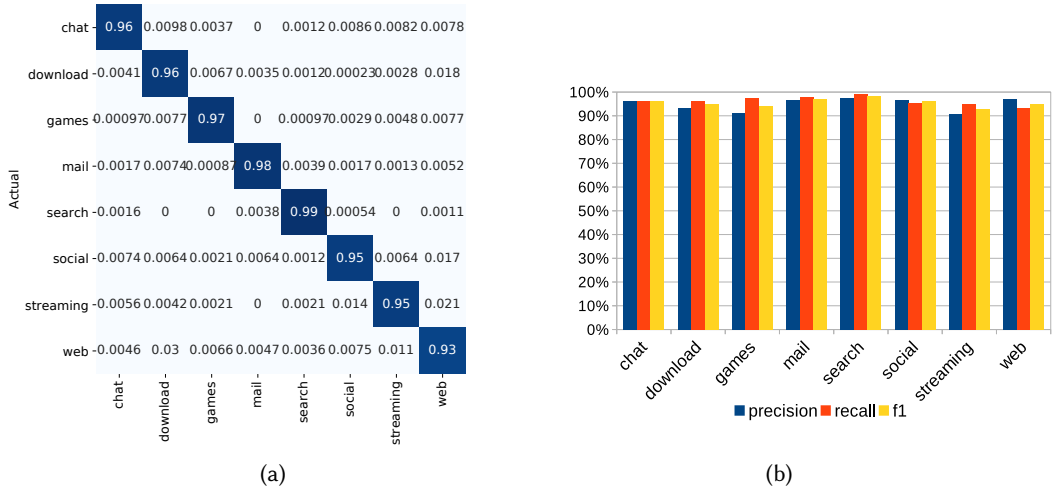
We start by evaluating the performance of our feature engineering approach and DL model architecture on the Orange'20 dataset. The data is pre-processed according to Section 4.2 and comprises of TLS flows only. We compare our DL model against the state-of-the-art CNN and CNN-LSTM architectures, showing a clear advantage and asserting our contributions.

Our model is trained using the Adam optimizer for 40 epochs, with 20% of the dataset used for validation. The learning rate is set to 0.001 at first and reduced every 10 epochs. The results of the experiment are shown in Figure 8, with an overall accuracy and weighted average F1-score of 95.56% and 95.58%, respectively. Figure 8b shows the per-class precision, recall, and F1-score of our model. The F1-score is over 94% for all classes, which implies very good stability despite the highly imbalanced classes in the dataset. This can be attributed to the upsampling strategy employed during training. As a baseline, a C4.5 model is trained on statistical flow features. C4.5, among other DT-based algorithms, is a popular choice in traditional traffic classification [4, 12, 35] but only achieves an accuracy of 81.39%, as shown in Table 1. We attribute the disadvantage of the traditional approach in part to its sole reliance on high-level statistics and not being able to make distinctions based on more fine-grained details of the traffic shape.

The advantage of our model is clear when compared against the UCDavis CNN model, as shown in Table 1. When evaluated on the Orange'20 dataset, the UCDavis CNN model in [28] achieves an accuracy of 91.09% after 20 epochs, which is 4.5% lower than our three-part model in Figure 4. This is a significant gain in performance with 50.39% reduction in false classifications.

In Figure 9, we highlight the training progress to reason about this performance gain. The figure shows validation loss, training loss and accuracy at the end of each training epoch. Evidently, the UCDavis CNN model quickly overfits to the dataset. This is primarily due to a larger raw traffic input, only a part of which is actually useful to the model. After 12 epochs, the training accuracy is perfect, while the validation performance fails to improve. In contrast, the validation and training accuracies converge very well in the case of our model[4], as our feature engineering only provides the useful information for encrypted flows (i.e., handshake and flow shape). The implications of these results are significant. Despite having access to twice the number of packets, the competitor UCDavis CNN model is far less effective, as all meaningful information lies in the handshake. Exposing a larger chunk of the raw traffic to the UCDavis CNN model, simply confuses the model and provides more ways to overfit.

[4]The validation accuracy reported is actually higher than training accuracy. This is due to the *high dropout* strategy discussed in Section 4. During training, the dropouts are active, which in the case of very high dropout values can cause the training accuracy to suffer.

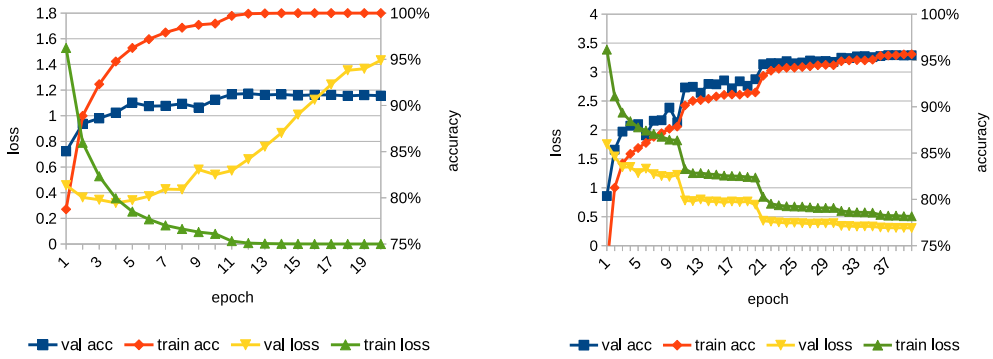


(a) (b)

Fig. 8. Confusion matrix (a) and per-class precision, recall, and F1-score (b) for our model's evaluation on the Orange'20 dataset. Our classifier consistently achieves +94% F1-score across all classes.

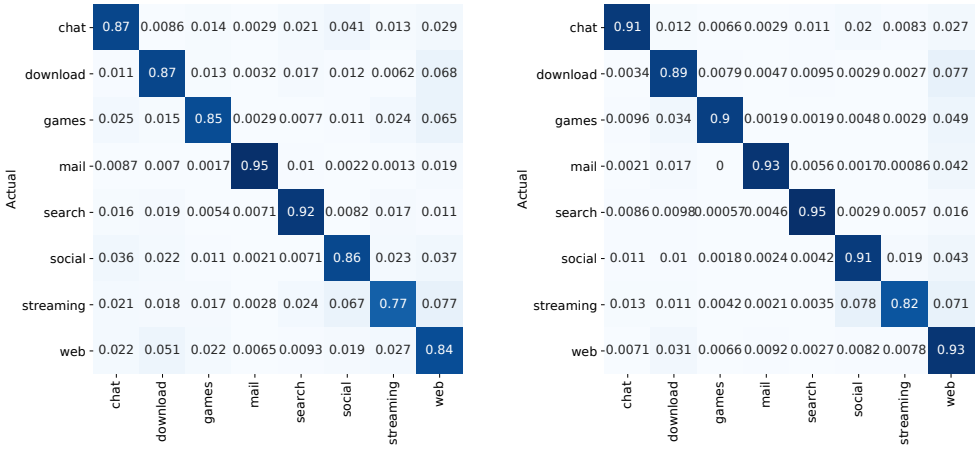
	W Avg precision (%)	W Avg recall (%)	W Avg F1-score (%)	Accuracy (%)	Epoch Time (s)
Full Model (SLSTM)	95.62	95.56	95.57	95.56	2584
Full Model (CNN)	94.54	94.42	94.37	94.43	232
Flow-only Model (SLSTM)	86.71	86.51	86.56	86.51	1814
Flow-only Model (CNN)	76.77	73.17	73.76	73.17	211
UCDavis CNN [28]	91.09	91.06	91.04	91.05	168
UCDavis CNN-LSTM [28]	89.74	89.72	89.73	89.72	245
Traditional Baseline (C4.5)	81.56	81.39	81.41	81.39	18*

Table 1. Performance comparison of TLS flow classification models (*the training time reported for the C4.5 model is for the entire training)



(a) UCDavis CNN model (b) Our model

Fig. 9. The progression of training and validation accuracy, and loss during the training of the UCDavis CNN model [28] and our model



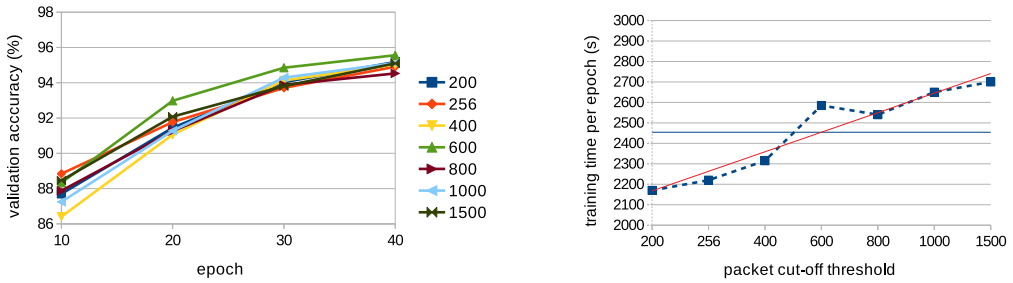
(a) Flow-only Stacked LSTM (b) UCDavis CNN

Fig. 10. Performance of UCDavis CNN model [28] and our Flow-only Stacked LSTM model

Table 1 depicts the performance of other variations of our model. We replace the Stacked LSTM (SLSTM) layers in the original model (cf. Section 3.3) with a deep one-dimensional CNN (cf. Appendix A.1), which is also a reasonable network for consuming a one-dimensional time-series. Though inferior to Stacked LSTM, the accuracy and F1-score of the resultant model is 94.43% and 94.37%, respectively. Nevertheless, it has a clear advantage over the UCDavis CNN model, which only processes raw traffic, with 37.77% less false classifications. The advantage of employing CNNs on the flow time-series side of the model, however, is in higher training speed (232 vs. 2,584 seconds/epoch), which is close to our competitor model despite being much more accurate. LSTM networks are notorious for being computationally expensive to train, and Stacked LSTM layers are even more so. Nevertheless, as model training is often a one-time investment and with the rapid advancement of computational hardware, this is a reasonable cost for higher classification accuracy.

We attribute the superior performance of our model to the flow aspect of our feature engineering and the Stacked LSTM layers. In fact, the flow time-series, despite being a simple feature set to model the traffic, is quite effective by itself. In Table 1, we also showcase results of our model variation with all the other inputs and their corresponding layers removed, except the flow time-series. We refer to these models as *"Flow-only"*. In this case, the Stacked LSTM and deep one-dimensional CNN architectures achieve an accuracy of 86.51% and 73.17%, respectively. Although being inferior in performance to models that also include raw traffic as input, it is important to note that the flow time-series features will always be available regardless of how the encrypted protocols evolve. These features enable a model to learn about the nature of traffic categories themselves, rather than fingerprinting a particular set of servers. Therefore, for all future research, we instigate the use of these flow features as a baseline for evaluations.

It is important to note that the UCDavis CNN model depicts a high misclassification between the streaming and social classes, where mutual providers such as Facebook and Twitter exist, as shown in Figure 10. This is a reoccurring issue with DL models for traffic classification that only rely on raw TLS bytes and are more tuned towards identifying a *server* rather than a *service*. The Flow-only



(a) Accuracy per cut-off value

(b) Epoch time

Fig. 11. Our model's performance for different values of packet truncation cut-off (C)

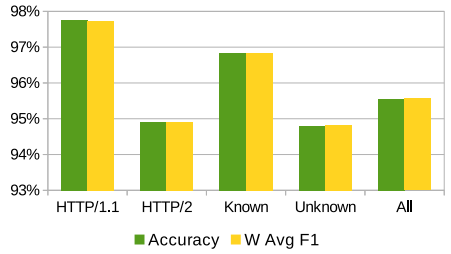
model, despite having access to very simplistic traffic features, makes fewer misclassifications between these classes, and when used together with the handshake bytes in our full model, is able to alleviate the difficulties in distinguishing between the two classes.

A rather surprising result in Table 1 is the performance of the UCDavis CNN-LSTM model [28]. As mentioned in Section 5.3, the UCDavis CNN-LSTM model is time-distributed over the flows of a "session", and theoretically has access to more information in comparison to our model that processes flows individually. However, the model's access to full traffic bytes does not work in its interest, and though having more parameters and capacity than the UCDavis CNN model, it overfits more severely to the data achieving a slightly less accuracy of around 90%. In fact, similar to the UCDavis CNN model, it quickly rises and achieves perfect training accuracy at around the seventeenth epoch, but fails to increase the validation accuracy any further.

5.4.1 Packet cut-off. We discussed in Section 4.2 that when the traffic bytes are pre-processed, we only truncate them to MTU for the sake of having fixed sizes. However, when loading the data for training, we additionally truncate the packets of the handshake to a cut-off of C bytes. Figure 11a shows the validation accuracy of the models with different values for the hyper-parameter C, at each epoch of training.

There is clearly a trade-off when increasing the value of C. Exposing a larger portion of the packet headers can give the model more information to work with. However, it also increases the capacity and size of the model, making it more likely to overfit to the training data. Based on the results, we found $C = 600$ to be a good value for the parameter. The final accuracy varies between 94.53% and 95.56% for all the models we considered. Note that the input is zero-padded at the end if the packet's header is smaller than C. Therefore, given that not all packets have useful information in the first C bytes, it is not surprising that a middle value for C works better on average in the aforementioned trade-off. Training time per epoch is displayed in Figure 11b. As expected, the training time increases almost linearly with C, since the number of parameters on the raw bytes side of the model increases.

5.4.2 HTTP Versions. The final insight here pertains to the HTTP version and how it affects the performance of our model. In Sections 1 and 2, we mentioned that HTTP/2 brings new features to the web, but at the same time complicates traffic classification. We evaluated our model's performance on different subsets of the validation set, based on the HTTP version. Not all flows captured in the Orange'20 dataset have the Next Protocol Negotiation (NPN) or ALPN records available to identify the protocol used over TLS. The "known" and "unknown" in Figure 12 elude to this fact.



(a)

	W Avg precision (%)	W Avg recall (%)	W Avg F1-score (%)	Accuracy (%)	% of dataset
HTTP/1.1	97.79	97.75	97.74	97.75	25.51
HTTP/2	95.12	94.91	94.91	94.91	12.02
Known	96.88	96.84	96.83	96.84	37.53
Unknown	94.94	94.80	94.83	94.80	62.47
All	95.62	95.56	95.57	95.56	100.00

(b)

Fig. 12. The impact of different HTTP versions on the performance of our model. Note that not all TLS sessions can be identified as the ALPN and NPN records may not be available. The flows without next protocol information are denoted as *"unknown"*.

W Avg precision (%)	W Avg recall (%)	W Avg F1-score (%)	Accuracy (%)
97.24	97.08	97.11	97.08

Table 2. Model performance in application-level classification

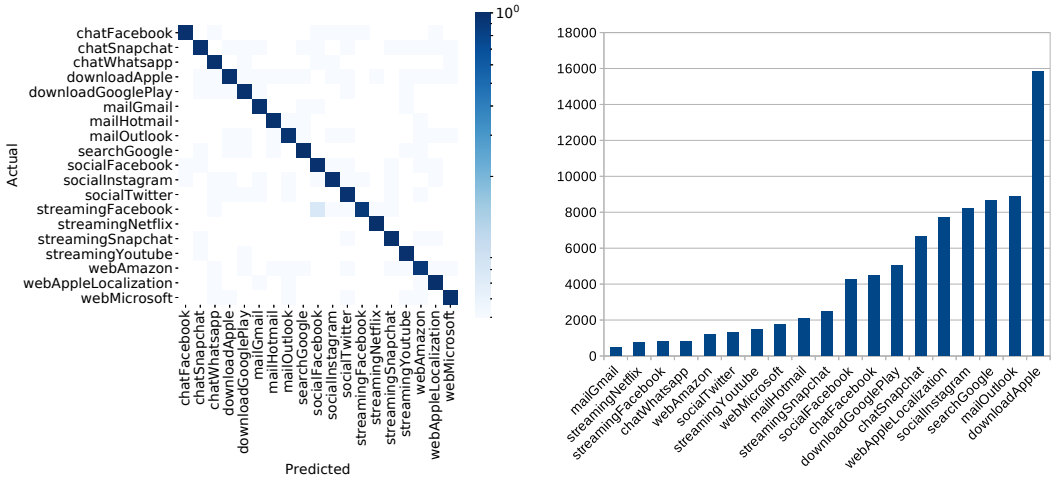
As expected, the performance of the model on HTTP/1.1 is higher than HTTP/2, i.e., 97.75% vs. 94.91%, due to the latter being a more complex protocol with features such as multiplexing, which make traffic classification more difficult. More importantly, the flows captured without the ALPN/NPN records are also generally harder to classify than the rest. This is due to the fact that there is a higher likelihood that these flows are captured from the middle of a session. Hence, they contain less information in their beginning for the model to leverage. It should also be noted that there is a disparity between the number of HTTP/2 and HTTP/1.1 flows in the training set. This reinforces the model's better performance on HTTP/1.1.

5.5 TLS Classification at Application-level

Many works in traffic classification (e.g., [3, 28]) focus on application-level classification which is at a finer granularity than our labels in Section 5.4. While counter-intuitive, application-level classification is often an easier task, especially when model is closed-world (i.e., dataset entries strictly belong to one of the n known applications) or canary features are not occluded. If the DL model is trained with the objective of a look-up table for identifying servers themselves (cf. Section 3.1), application-level classification is generally easier for the model, as it does not need to learn what behaviors are *shared* between different applications of the same service category.

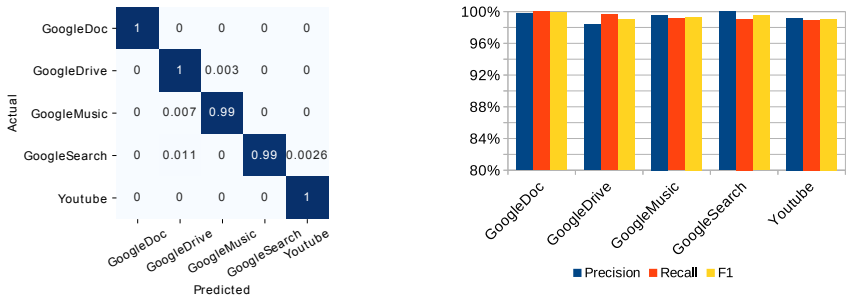
In order to evaluate our model in application-level classification, we identified 19 fine-grained labels that have enough representative flows for the training to be consistent. The distribution of these labels that make up for 82,776 flow entries of the dataset (i.e., ~ 70%) is shown in Figure 13b. Figure 13a depicts the result of employing our model by simply modifying the last softmax layer to accommodate 19 fine-grained classes instead of the 8 service categories. The overall accuracy of the model is 97.08%, as shown in Table 2. Despite having more classes, the accuracy of the model is higher than service-level classification, due to the raw bytes part of the model being extremely effective in fingerprinting specific servers. One side effect is that the different services from the same provider (e.g., Facebook video and Facebook social) have higher cross misclassifications, as evident in Figure 13a.

A key take-away from this experiment is that a good feature engineering approach for encrypted traffic classification can be adapted to different classification tasks, as it is good in capturing the "nature of traffic". It will be an interesting part of future research to leverage the feature engineering



(a) Application-level model accuracy (b) Label distribution

Fig. 13. Distribution of application-level labels in Orange'20 dataset and the model's performance



(a) Confusion matrix (b) Per-class precision, recall, and F1-score

Fig. 14. Model performance on the UCDavis QUIC traffic dataset

presented in Section 3.1 in areas other than service- and application-level classification, such as QoS classification and security.

5.6 QUIC Classification

We evaluated our model on the UC Davis QUIC Dataset (cf. Section 5.2.2), which only includes the traffic shape time-series and not the actual network traces. In order to adapt to the dataset structure, we modified our approach by only activating the flow time-series part of the model and conducted the training for 20 epochs.

In evaluation, our model achieve a high validation accuracy (i.e., 99.37%), which is higher than the best one reported by the authors for their CNN model in [29] (i.e., ~98%), regardless of whether their semi-supervised scheme (i.e., pre-training on unlabeled data first) is carried out or not. Figure 14a shows the result of the classification, which re-affirms that our proposed feature engineering is indeed a good indicator of the traffic class, and can adapt well to different encrypted web protocols.

Our model achieves high accuracies despite the fact that QUIC is a more challenging protocol with a larger encrypted portion. These results also validate the utility of the Stacked LSTM architecture used in the flow time-series part of our model.

6 CONCLUSION & FUTURE WORK

Traffic classification has become increasingly challenging with the widespread adoption of encryption in the Internet. Moreover, encrypted protocols are bound to evolve, rendering protocol-specific approaches futile in the future. In this paper, we propose a DL approach for encrypted traffic classification that focuses on protocol-agnostic aspects of the encrypted web traffic. Our feature set comprises of a time-series of packet size, direction and inter-arrival times, flow statistics, and raw bytes from only the TLS handshake, while the DL model is based on CNN and Stacked LSTM layers. We show that raw traffic apart from the TLS handshake does not contribute to the DL model's performance, but rather adds to its complexity and increases overfitting. Thus, the feature engineering method makes use of concepts that are applicable to most encrypted protocols.

To ensure our DL model robustness to future versions of TLS, we obfuscate parts of the TLS handshake (e.g., SNI field and cipher info) that give away the server identity. Instead, we focus on the traffic shape and timing of packets, which show high potential in learning the complex nature of traffic among different classes. We show that our DL model generalizes for different classification objectives, i.e., service- and application-level classification, and adapts to different encrypted web protocols (i.e., HTTP/2 and QUIC) by simply changing the training data. We evaluate our approach for service-level classification on a real-world mobile traffic dataset from an ISP, and show that by leveraging less raw traffic and a smaller number of parameters, our model outclasses a state-of-the-art approach [28, 29].

Some recent works [36, 37] have leveraged generative models for addressing the class imbalance problem in deep traffic classification. These data augmentation approaches can facilitate DL models and should be explored more in the future. Furthermore, Attention Networks [34] have gained popularity in the past few years, especially in the NLP literature. Recently, they have also been leveraged in traffic classification [24, 42]. Despite our initial experiments showing no real advantage in replacing LSTM layers in our architecture with Attention Networks, we will continue to experiment with the combination of recurrent and convolutional layers with Attention Networks. We also plan to investigate domain adaptation and transfer learning [10], which are important yet often overlooked in traffic classification literature, and were not in the scope of this paper. For instance, it is interesting to study how well a model trained under certain network conditions (e.g., subnet, packet drop rate, latency and QoS level) would work under different conditions. Similarly, measuring the accuracy of a model through time and studying how well it translates to completely different sets of applications and servers should be explored in depth. Another important open problem is re-purposing models for different traffic classification tasks. For instance, a service classification model trained on millions of entries might be transferable to malware detection using relatively few data points, by reusing the weights of its initial and middle layers. Lastly, the robustness of traffic classification models against adversarial attacks and privacy leaks is of utmost importance and remains an open research question for future work.

ACKNOWLEDGMENTS

We thank our shepherd Athina Markopoulou and the anonymous reviewers for their valuable feedback, Yann Meyer for his help in the preparation of the dataset, and Ezzeldin Tahoun for his help in the preliminary stages of the project.

REFERENCES

[1] Université Toulouse 1. 2020. Blacklists UT1. http://dsi.ut-capitole.fr/blacklists/index_en.php. [Online; Accessed 01-October-2020].

[2] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. 2018. Mobile encrypted traffic classification using deep learning. In *IEEE Network Traffic Measurement and Analysis Conference (TMA)*. 1–8.

[3] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. 2019. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management* 16, 2 (2019), 445–458.

[4] Riyad Alshammari and A Nur Zincir-Heywood. 2009. Machine learning based encrypted traffic classification: Identifying ssh and skype. In *IEEE symposium on computational intelligence for security and defense applications*. 1–8.

[5] Blake Anderson and David McGrew. 2017. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1723–1732.

[6] Blake Anderson and David McGrew. 2020. Accurate TLS Fingerprinting using Destination Context and Knowledge Bases. *arXiv preprint arXiv:2009.01939* (2020).

[7] Blake Anderson, Subharthi Paul, and David McGrew. 2018. Deciphering malware's use of TLS (without decryption). *Springer Journal of Computer Virology and Hacking Techniques* 14, 3 (2018), 195–211.

[8] Mike Belshe and Roberto Peon. 2012. *SPDY Protocol*. Technical Report. Network Working Group. 1–51 pages. https://tools.ietf.org/pdf/draft-mbelshe-httpbis-spdy-00.pdf

[9] Mike Belshe, Roberto Peon, and Martin Thomson. 2015. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. IETF RFC 7540. 1–96 pages.

[10] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning* 79, 1-2 (2010), 151–175.

[11] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. 2007. Revealing skype traffic: when randomness plays with you. In *ACM SIGCOMM Computer Communication Review*, Vol. 37. 37–48.

[12] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. 2018. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Springer Journal of Internet Services and Applications* 9, 1 (2018), 16.

[13] Pierre-Olivier Brissaud, Jérôme Francçis, Isabelle Chrisment, Thibault Cholez, and Olivier Bettan. 2019. Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 842–856.

[14] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)* 3, 3 (2019), 1–25.

[15] Zhiyong Bu, Bin Zhou, Pengyu Cheng, Kecheng Zhang, and Zhen-Hua Ling. 2020. Encrypted Network Traffic Classification Using Deep and Parallel Network-in-Network Models. *IEEE Access* 8 (2020), 132950–132959.

[16] Zhitang Chen, Ke He, Jian Li, and Yanhui Geng. 2017. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *IEEE International Conference on Big Data (Big Data)*. 1271–1276.

[17] Ramin Hasibi, Matin Shokri, and Mehdi Dehghan. 2019. Augmentation scheme for dealing with imbalanced network traffic classification using deep learning. *arXiv preprint arXiv:1901.00204* (2019).

[18] Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. 2017. Unsupervised traffic flow classification using a neural autoencoder. In *IEEE Conference on Local Computer Networks (LCN)*. 523–526.

[19] Janardhan Iyengar and Ian Swett. 2015. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. Technical Report. Network Working Group. 1–30 pages.

[20] Jana Iyengar and Martin Thomson. 2018. QUIC: A UDP-based multiplexed and secure transport. *Internet Engineering Task Force, Internet-Draft* (2018).

[21] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[22] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *International Conference on Information Systems Security and Privacy (ICISSP)*. 253–262.

[23] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE Conference on Computer Communications (INFOCOM)*. 1171–1179.

[24] Xun Liu, Junling You, Yulei Wu, Tong Li, Liangxiong Li, Zheyuan Zhang, and Jingguo Ge. 2020. Attention-based bidirectional gru networks for efficient https traffic classification. *Elsevier Information Sciences* 541 (2020), 297–315.

[25] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access* 5 (2017), 18042–18050.

[26] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Springer Soft Computing* 24, 3 (2020), 1999–2012.

[27] Jonathan Muehlstein, Yehonatan Zion, Maor Bahumi, Itay Kirshenboim, Ran Dubin, Amit Dvir, and Ofir Pele. 2017. Analyzing HTTPS encrypted traffic to identify user's operating system, browser and application. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 1–6.

[28] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. 2019. Large-scale mobile app identification using deep learning. *IEEE Access* 8 (2019), 348–362.

[29] Shahbaz Rezaei and Xin Liu. 2018. How to achieve high classification accuracy with just a few labels: semi-supervised approach using sampled packets. *arXiv preprint arXiv:1812.09761* (2018).

[30] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2017. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376* (2017).

[31] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security Symposium (USENIX Security 17)*. 1357–1374.

[32] Yan Shi, Dezhi Feng, and Subir Biswas. 2019. A Natural Language-Inspired Multi-label Video Streaming Traffic Classification Method Based on Deep Neural Networks. *arXiv preprint arXiv:1906.02679* (2019).

[33] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security* 31, 3 (2012), 357–374.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[35] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. 2015. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management* 25, 5 (2015), 355–374.

[36] Ly Vu, Cong Thanh Bui, and Quang Uy Nguyen. 2017. A deep learning based method for handling imbalanced problem in network traffic classification. In *International Symposium on Information and Communication Technology*. 333–339.

[37] Pan Wang, Shuhang Li, Feng Ye, Zixuan Wang, and Moxuan Zhang. 2020. PacketCGAN: Exploratory study of class imbalance for encrypted traffic classification using CGAN. In *IEEE International Conference on Communications (ICC)*. 1–7.

[38] Wei Wang, Yiqiang Sheng, Jinlin Wang, Xuewen Zeng, Xiaozhou Ye, Yongzhong Huang, and Ming Zhu. 2018. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* 6 (2018), 1792–1806.

[39] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*. 43–48.

[40] Nigel Williams, Sebastian Zander, and Grenville Armitage. 2006. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review* 36, 5 (2006), 5–16.

[41] Haipeng Yao, Pengcheng Gao, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Zhu Han. 2019. Capsule network assisted IoT traffic classification mechanism for smart cities. *IEEE Internet of Things Journal* 6, 5 (2019), 7515–7525.

[42] Haipeng Yao, Chong Liu, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Shui Yu. 2019. Identification of Encrypted Traffic Through Attention Mechanism Based Long Short Term Memory. *IEEE Transactions on Big Data* (2019).

[43] Zhuang Zou, Jingguo Ge, Hongbo Zheng, Yulei Wu, Chunjing Han, and Zhongjiang Yao. 2018. Encrypted traffic classification with a convolutional long short-term memory neural network. In *IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 329–334.

A APPENDIX

A.1 Model Details

Table 3. Architecture of the tripartite model with convolutions in the flow side. By default a layer connects to its predecessor.

Type	Shape	Connection
In	(3, 600)	Header Input
Reshape	(1800, 1)	
Convolution1D	(1799, 256)	
ReLU		
Convolution1D	(1799, 256)	
ReLU		
MaxPooling1D	(899, 256)	
Convolution1D	(898, 128)	
ReLU		
Convolution1D	(897, 128)	
ReLU		
MaxPooling1D	(448, 128)	
Flatten	(57344)	
In	(61)	Flow Meter Input
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.2)		
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.5)		
In	(1024, 3)	Flow Input
Convolution1D	(1024, 128)	
BatchNorm		
ELU		
Convolution1D	(1024, 128)	
BatchNorm		
ELU		
MaxPooling1D	(512, 128)	
Convolution1D	(512, 64)	
BatchNorm		
ELU		
Convolution1D	(512, 64)	
BatchNorm		
ELU		
MaxPooling1D	(256, 64)	
Flatten	(16384)	
Concatenate	(73928)	
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(8)	
Softmax		

Table 4. Architecture of the tripartite model with LSTM's in the flow side. By default a layer connects to its predecessor.

Type	Shape	Connection
In	(3, 600)	Header Input
Reshape	(1800, 1)	
Convolution1D	(1799, 256)	
ReLU		
Convolution1D	(1799, 256)	
ReLU		
MaxPooling1D	(899, 256)	
Convolution1D	(898, 128)	
ReLU		
Convolution1D	(897, 128)	
ReLU		
MaxPooling1D	(448, 128)	
Flatten	(57344)	
In	(61)	Flow Meter Input
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.2)		
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.5)		
In	(1024, 3)	Flow Input
Dense(Distributed)	(1024, 512)	
BatchNorm		
LeakyReLU		
LSTM(Forward)	(1024, 256)	
LSTM(Backward)	(1024, 256)	
LSTM(Forward)	(256)	
Dropout(0.5)		
Concatenate	(57800)	
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(8)	
Softmax		

A.2 Performance Metrics

Each prediction from a binary classifier model can be either a true-positive (TP), false-positive (FP), true-negative (TN), or false-negative (FN). When evaluating a model's performance, the count of each of these four metrics, can help calculate other useful metrics.

Precision denotes what percentage of positive instances from the model, are correct. It is defined as follows:

$$precision = \frac{TP}{TP + FP} \times 100\% \tag{1}$$

Recall indicates what percentage of the instances that have a positive label in ground truth, have indeed been classified as positive by the model:

$$recall = \frac{TP}{TP + FN} \times 100\% \tag{2}$$

However, these two metrics by themselves can not be used to evaluate a model. For instance, a model that always gives a positive prediction will have a perfect recall (i.e., $recall = 100\%$). Similarly,

a model that always gives negative predictions, would never have a false-positive and will have a perfect precision (i.e., $precision = 100\%$). That is why, F1-score (or F1) is defined as a combination of the two metrics:

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \times 100\% \tag{3}$$

This is called a "harmonic mean" of precision and recall. It is more useful than the arithmetic mean, since if either metric falls to zero, so would the F1-score.

Accuracy is a more familiar metric, which essentially indicates what percentage of all predictions are correct:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{4}$$

In our evaluations, we use weighted average recall, precision, and F1-score of the models, which is an average of those metrics over all classes and weighted by the class sizes (i.e., number of entries with each class label in the dataset). This is denoted by "W Avg" precision, recall, and F1-score throughout the paper.

Received October 2020; revised December 2020; accepted January 2021