Data Drift in DL: Lessons Learned from Encrypted Traffic Classification

Navid Malekghaini*, Elham Akbari*, Mohammad A. Salahuddin*, Noura Limam*, Raouf Boutaba*,
Bertrand Mathieu†, Stephanie Moteau†, and Stephane Tuffin†
*David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
{nmalekgh, eakbaria, mohammad.salahuddin, noura.limam, rboutaba}@uwaterloo.ca
†Orange Labs, Lannion, France
{bertrand2.mathieu, stephanie.moteau, stephane.tuffin}@orange.com

***Abstract*—Deep learning models have shown to achieve high performance in encrypted traffic classification. However, when it comes to production use, multiple factors challenge the performance of these models. The emergence of new network traffic protocols, especially at the application-layer, as well as updates to previous protocols affect the patterns in input data, making the model's previously learned patterns obsolete. Furthermore, proposed model architectures are usually tested on datasets collected in controlled settings, which makes the reported performances unreliable for production use. In this paper, we study how the performances of two high-performing traffic classifiers change on multiple real-world datasets collected over the course of two years. We investigate the changes in traffic data patterns showing the extent to which these changes reduce the performance of the two models. Furthermore, we propose architectural adaptations to a flow time-series based traffic classifier, showing that they improve accuracy by 4.8%.**

***Index Terms*—Data Drift, Encrypted Traffic Classification, Deep Learning, Web Traffic, HTTP/2, QUIC**

I. Introduction

Deep learning (DL) models have shown superior performance in encrypted traffic classification [1]–[3]. However, when it comes to deploying a DL model in production, there is more to consider than model performance, which is dependent on the target (*i.e.*, test) dataset. In practice, the model performance is tightly coupled with the target dataset properties. The effect of the target dataset on model accuracy has been highlighted when comparing the performances of different traffic classification models [2], [4].

The need for datasets with sample distributions that reflects real-world data is a known issue in traffic classification. The fact that network traffic datasets are often collected under controlled settings or generated synthetically is not due to a dismissal of this principle but rather, it is a reflection of the difficulty of labeling real-world network traffic. Even if perfectly collected and labeled data existed at some point, it is likely to be considered irrelevant six months later, due to the dynamic nature of network traffic. Over time, traffic patterns are affected by the protocols, software, and devices that generate them. This pattern evolution is known as *data drift*, or *concept drift*, in the machine learning (ML) literature.

Data drift is a phenomenon in which the distribution of input data over classes changes over time. For example, a service may switch to another transport protocol leading to a different flow time-series shape or traffic shape. A flow time-series-based classifier is then likely to decay in identifying the new traffic. Hence, data drift refers to a change in the distribution of real-world data caused by its dynamic nature, which affects model performance.

In this work, we study the effect of data drift on the performance of two state-of-the-art encrypted network traffic classifiers [1], [2]. Using several datasets of real-world network traffic collected from a major ISP's network, we show that model performance degradation does indeed occur in a production setting, *i.e.*, when a model trained on old data attempts at classifying new data. We offer an explanation for the degradation, based on the portions of the traffic that the models struggle on. We also analyse the architecture of the models, offering guidelines for designing architectures that we empirically show are more robust to data drift. We study the effect of dataset size on model performance, guided by the observation that in practice, several factors in the data collection process affect the number of possible labeled samples, and the datasets on which the models train can be of various sizes. Our main contributions can be summarized as:

- To the best of our knowledge, we are the first to address the problem of data drift in real-world encrypted traffic classification. We study this phenomenon using five real-world network traffic datasets collected over a course of more than two years from a major ISP's network.
- We provide insights into the type of data drift that happens in network traffic. These insights are useful to practitioners working with traffic classification models in production.
- We provide guidelines for designing models that are robust to a change of dataset and encryption protocol. Our guidelines have the distinction of being empirically tested on real-world data.

This paper is organized as follows. Section II presents the closely related works, while Section III presents the datasets and models used in the paper. In Section IV, we explain our experiments with the models trained on one dataset and tested on one or more other datasets. We further investigate and explain the obtained results. In Section V, we present

ISBN 978-3-903176-48-5© 2022 IFIP

our insights and guidelines on designing a robust model architecture, along with the supporting experiment results. Section VI concludes the paper and outlines directions for future work.

II. Related Work

In light of the obfuscation of previously reliable features by encryption such as application-layer payload, the traffic classification literature turned to features (*e.g.*, packet size, timestamp, direction and the statistics derived from them) that were difficult to tweak without affecting quality of service. Before the advent of DL, the performance of several supervised models such as Naïve Bayes, AdaBoost, Support Vector Machines for encrypted traffic classification using these features were studied in [5], [6]. Furthermore, semi-supervised approaches based on Gaussian Mixture Models, k-Means and k-Nearest Neighbour clustering and Multi-Objective Genetic Algorithms were studied in [7]–[9] for real-time encrypted traffic classification. A survey of classical ML approaches can be found in [10], [11].

The capacity to automatically extract feature vectors from raw data in DL provided new opportunities for encrypted traffic classification. These opportunities were explored using various deep models including MultiLayer Perceptrons, Stacked Autoencoders, Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) ([1], [3], [12]). These models were evaluated using open mixed-protocol datasets such as ISCXVPN2016 [13] and ISCXIDS2012 [14]. The work in [4] uses a proprietary dataset to evaluate numerous application-level classification methods that use DL models.

A survey of DL models used for network traffic analysis can be found in [15]. In this section, we focus on models that deal with network traffic data and consider the problem of real-world deployment of a trained classifier with respect to robustness across different datasets. Since the literature of encrypted traffic classification is not extensive in this specific area, we consider other domains such as anomaly detection.

Ma *et al.* [16] propose a framework to detect and adjust to concept drift in an anomaly detection system. The authors define concept drift as a sudden change in the distribution of the key performance indicator (KPI) stream. Since the number of KPIs in their base anomaly detector is large, they especially focus on automatic threshold setting for the concept drift detection algorithm to free operators from manually tuning per-KPI parameters. Their concept drift adaptation algorithm is based on linearly transforming the new concept to the old concept in each time window. Their work differs from ours, as they deal with a different domain where input data is in the form of a continuous stream, so applying standard concept drift algorithms to their domain is rather straightforward.

Saurav *et al.* [17] consider the problem of an anomaly detection model losing its relevance when trained on historical data and used in a dynamically changing and non-stationary environment, where the definition of normal behavior changes. Their proposed model, a recurrent neural network (RNN) trained incrementally on a data stream, is used to make predictions while continuously adapting to new data when prediction errors increase. They show that their model is able to adapt to different types of concept drift, *e.g.*, sudden, gradual and incremental.

Taylor *et al.* [18] study the effect of training on one dataset and testing on another, building up on their previous work AppScanner, an automatic tool for fingerprinting smartphone apps from encrypted data. They collect five datasets of app generated traffic, four of which were collected six months after the first one and differ from the first one in a subset of three factors: time of collection, app device, and app version. The authors test the effect of each factor on the accuracy of the model when trained on the base dataset and tested on the target dataset and find that mere time passing has the least effect on the model's accuracy, whereas the model's accuracy drops from around 70% to 19% when tested on the dataset with new app versions and devices.

Although the work in [18] is based on traditional ML models, it relates to ours in the recognition of the effect of ambiguous flows in confounding the classifier, as well as confirming the phenomenon of model decay in mobile app fingerprinting. As opposed to the synthetic datasets employed in [18], our work is based on real-world datasets.

III. Methodology

A. Deep Learning models

1) UW Tripartite Model: The University of Waterloo (UW) Tripartite model is a DL model proposed in [2] and reported to achieve above 90% accuracy on a dataset of purely encrypted TLS network traffic. It is a three-part model, each part designed to operate on one of three different types of input data. The model consists of a series of CNNs operating on header bytes from the first three packets of the TLS handshake. CNNs are useful for extracting shift-invariant information which makes them suitable for header bytes. The model further contains a series of LSTM layers operating on flow time-series data, which includes a three-dimensional array of packet sizes, packet directions, and packet inter-arrival times for each flow. LSTMs are renowned for relating useful information in a time-series data. The output of the LSTMs passes through a dropout layer before being concatenated to other parts' outputs. Lastly, a series of dense layers in the model is designed to work on statistical flow data, which includes 77 features. The statistical features are called auxiliary features in this paper, as experiments suggest that they have the least effect on the model's performance. The outputs of the three parts are then concatenated and passed through two dense layers and a softmax layer to obtain the end result, as shown in Fig. 1.

The UW Tripartite model is used as a baseline in this work. To the best of our knowledge, it obtains the highest accuracy on a fully encrypted dataset, for service-level classification, which aligns with the work in this paper.

2) UCDavis CNN Model: The authors in [1] propose a CNN model for early classification of network traffic flows. Their CNN model operates on the first six packets of a flow, for

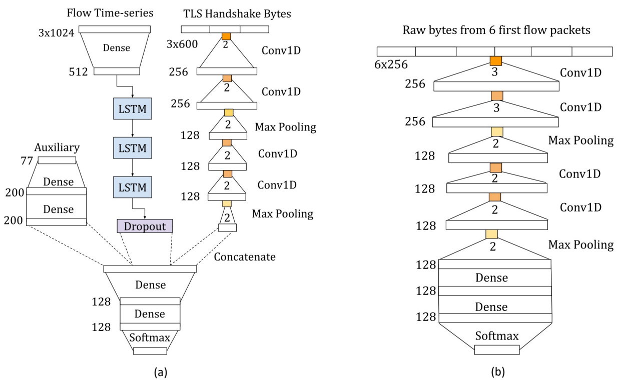

Fig. 1. (a) UW Tripartite model architecture [2] (left), and (b) UCDavis CNN model architecture [1] (right). Orange and yellow boxes depict convolution and max-pooling layer kernels, respectively. Each layer's output vector is depicted by a white box accompanied by its size.

each of which, the first 256 raw bytes from layer 3 and above are extracted and concatenated together to form the input vector. The model consists of convolutional, max-pooling and dense layers as shown in Fig. 1. UCDavis CNN model is also used as a baseline, as it was shown in [2] that after the UW Tripartite model, this model obtains the best accuracy on their fully encrypted dataset among a number of evaluated models.

B. Datasets description

We use a total of six datasets in this paper which consist of TLS and QUIC traffic traces collected from a major ISP's network. The source and destination IP addresses are obfuscated and the packets are truncated after 400 bytes, except for the TLS handshake packets.

Preprocessing and labeling modules are used to turn the packet captures into labeled datasets of traffic flows. Both modules are implemented as in [2]. The preprocessed data includes TLS raw header bytes from the flows, as well as flow time-series data containing an array of packet sizes, packet inter-arrival times, and packet directions for each flow. Moreover, it consists of 77 auxiliary features for each flow, extracted using CICFlowMeter [13]. The auxiliary features include statistical information about flows, *e.g.*, mean, median, minimum, and maximum of packet sizes in each direction. The labeling module labels the flows according to the Server Name Indication (SNI) field and consists of 8 classes each representing a service category, namely, chat, download, games, mail, search, social, streaming, and web.

Our datasets can be categorized into two types based on encryption protocol: *TLS* and *QUIC*.

(i) TLS datasets: We leverage five datasets encrypted with the TLS protocol, each containing one to two hours of packet traces. The datasets are captured chronologically and named in the format of MM-YYYY as: 07-2019, 09-2020, 04-2021, 05-2021, and 06-2021.

(ii) QUIC dataset: The QUIC dataset, QUIC-05-2021, is extracted from a packet trace of QUIC traffic captured at the same time as the TLS 05-2021 dataset. The dataset only consists of flow time-series data since TLS handshake bytes are tightly coupled to the TLS protocol and thus such data is irrelevant to QUIC. Auxiliary data was not added to this

TABLE I
DATASET PROPERTIES

Protocol	Name	Total flows (K)	Labeled flows (K)	Labeled flows (%)
TLS	07-2019	762.7	119.8	15.7
	09-2020	411.7	89.9	21.8
	04-2021	284.8	42.3	14.8
	05-2021	124.0	17.5	14.1
	06-2021	261.2	51.2	19.6
QUIC	QUIC-05-2021	37.8	26.0	68.0

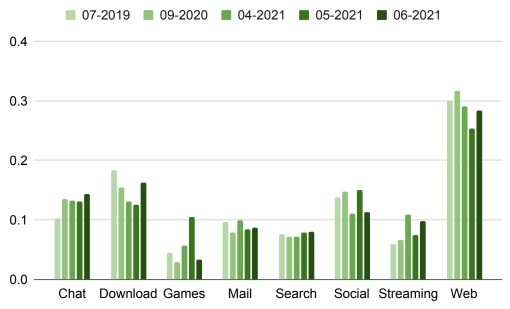

Fig. 2. Class distribution of the TLS datasets

dataset as the effect of such data on the performance was negligible in our experiments. The QUIC dataset is used to show that our architecture adaptation best practices, which are centered around the flow time-series part of the UW Tripartite model, generalize to non-TLS encrypted data (*cf.*, Section V).

Table I shows the total number of flows and labeled flows in each dataset, along with the percentage of labeled flows. The number of labeled flows shows the size of each dataset. We examine our labeling module's performance for the TLS flows by calculating the percentage of labeled flows in each dataset, which is shown in Table I. We can see that the percentage of the labeled flows across the datasets are more or less in line with each other. Additionally, the labeled distribution of service classes for TLS flows are depicted in Fig. 2. There is insignificant difference in class distribution across the datasets. Therefore, we use the accuracy as the primary performance metric for evaluating the models across the datasets. Moreover, to deal with the class imbalance we adopt a weighting strategy, *i.e.*, we up-sample classes with a smaller number of flows.

For labeling the QUIC dataset, we changed the classes from the TLS dataset. Since QUIC is still not widely adopted by services across the web, not all classes from the TLS dataset have enough samples in the QUIC dataset. For instance, QUIC is known for enhanced security and faster connections, which makes it more suitable for time-sensitive applications, *e.g.*, streaming services. Therefore, it makes sense that we did not see any flows labeled as the "Download" class. Hence, we keep the Games, Social, Streaming, and Web classes, while adding new classes of E-commerce and Resources. The Resources class corresponds to the flows that are essentially shared among different websites that mostly deliver tools, such as JS APIs or design content for websites. The new labeling module can label up to 68% of the flows, a large improvement from the $<20\%$ labeling performance on the TLS dataset. This is probably because fewer services use QUIC, so the SNIs are not as varied in this dataset as they were in the TLS datasets.

C. Software stack and performance metrics

The software stack for data pre-processing, model training, and evaluation includes Tensorflow with Keras API, CUDA, PySpark, SCAPY, and TShark. Training was conducted on 80% of each dataset, while the remaining 20% was used for validation. A multi-class classification problem can be seen as a set of many binary classification problems, one for each class. Each binary classification task may result into True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). The performance of each binary classifier can be measured in terms of:

$$Precision = \frac{TP}{TP+FP} \times 100 \quad Recall = \frac{TP}{TP+FN} \times 100$$

$$F1-score = \frac{2 \times Precision \times Recall}{Precision + Recall} \times 100$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \times 100$$

In this work, we measure the performance of the multi-class classifiers in terms of accuracy, and weighted average F1-score, recall, and precision, where weighted average is the average of the corresponding metric across all classes weighted by the number of data points that we could label for each class.

IV. Investigation

In this section, we study the performance of the UW Tripartite traffic classification model when trained on a baseline dataset and used on a different target dataset. We investigate the time decay aspect by further decomposing the model and experimenting with different datasets.

A. Baseline performance

We start by highlighting the performance of the UW Tripartite model [2] on the 07-2019 dataset. This dataset is the closest in time of capture to the dataset used in [2]. Therefore, we expect to see similar model accuracy. To have an insight into the performance of each part of the UW Tripartite model separately, we also conduct experiments on the decomposed model. Table II shows the performance of the full Tripartite model (denoted FHA) as well as its decomposed parts, *i.e.*, flow time-series part (denoted F), TLS header part (denoted H) and the auxiliary part (denoted A).

The results on the baseline 07-2019 dataset are very similar to the results reported in [2]. In particular, for the full model and the flow time-series part, the difference in accuracy is around 1% and 0.2%, respectively. We notice that, when we train the three parts of the model separately on the 07-2019 dataset, the TLS header part shows the highest accuracy, which is 0.3% higher than the accuracy of the full model on the same dataset, while the auxiliary part shows the lowest accuracy of 43.8%. With such a low accuracy, it is likely that the auxiliary input to the model is not helping but rather confusing the full UW Tripartite model, resulting in a lower performance than the TLS header part alone.

TABLE II
UW Tripartite model performance on the 07-2019 and [2] datasets

Dataset	Accuracy (%)			
	FHA	H	F	A
07-2019	94.5	94.8	86.3	43.8
Akbari *et al.* [2]	95.5	N/A	86.5	N/A

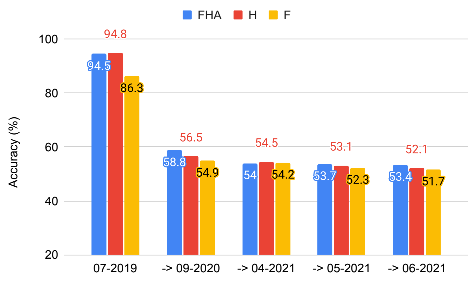

Fig. 3. Accuracy of the UW Tripartite model when trained on baseline 07-2019 dataset and used (notation $\rightarrow$) on target datasets

B. Model robustness to time decay

We evaluate the performance of the UW Tripartite model on different target (*i.e.*, test) datasets, *i.e.*, 09-2020, 04-2021, 05-2021, 06-2021, after training it on the baseline 07-2019 dataset. The target datasets were collected at different points in time within two years from the 07-2019 dataset. More precisely, we study the performance of the full model, as well as the decomposed model parts when trained on the baseline dataset and used on the target datasets. Experiments have shown that the performance of the auxiliary part of the model has little or rather negative impact on the performance of the full model. In particular, when we exclude the auxiliary part from the Tripartite model, we achieve 94.9% accuracy on the 07-2019 dataset, which is even better than what we achieve with the full model. For this reason and due to space limitations, we focus our study on the TLS header and flow time-series parts of the UW Tripartite model.

The results of the first set of experiments is shown in Fig. 3. Evidently, the prediction ability of the model decays over time, which is quantified in Table III. We see that the decay of the full model is at its lowest on the 09-2020 dataset (*i.e.*, 35.7%) and at its highest on the 06-2021 dataset (*i.e.*, 41.1%). Note that the 07-2019 dataset and the 06-2021 dataset are about two years apart.

Model decay over time is an expected phenomenon. Nevertheless, we see that it does not have an equal impact on the TLS header and flow time-series parts of the model. In fact, the model performance on the header part decays 7% more on average than the performance on the flow time-series part (*i.e.*, 40.75% compared to 33%). This suggests that using the traffic shape features, which is captured by the flow time-series input, make the classifier more robust to decay over time. This also suggests that the TLS headers contribute more to the drop in accuracy over time for the UW Tripartite model.

The previous experiments also highlight that performance decay correlates with the time difference between the training dataset and the target datasets. Therefore, we run experiments

TABLE III
DECAY IN MODEL'S PREDICTIVE ABILITIES WHEN TRAINED ON THE BASELINE 07-2019 DATASET

Model	Target datasets				Avg. accuracy drop (%)
	09-2020	04-2021	05-2021	06-2021	
FHA	35.7	40.5	40.8	41.1	39.52
H	38.3	40.3	41.7	42.7	40.75
F	31.4	32.1	34.0	34.6	33.02

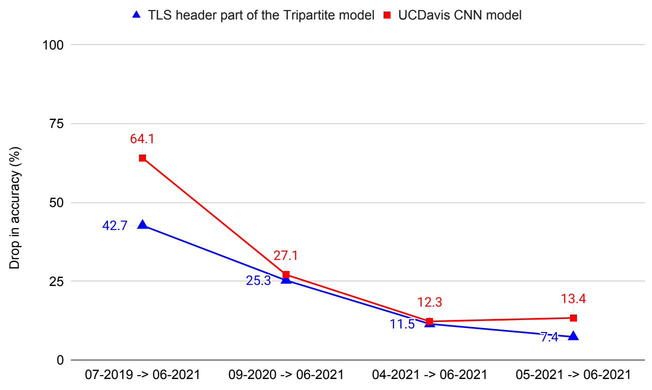

Fig. 4. Model decay over time for the TLS header part of the UW Tripartite model [2] and UCDavis CNN model [1]. Notation X → Y: model was trained on dataset X and used on Y.

to further investigate this observation. In particular, we train the TLS header part of the model using different datasets and test it on the most recent 06-2021 dataset. We conduct the same experiments for the UCDavis CNN model [1] and compare the performance of both models.

The performance decay in decreasing order of time span between the training and target datasets are shown in Fig. 4. It is evident that the closer the datasets are in time of capture, the lower the performance decay. For example, the TLS header part of the UW Tripartite model decays by 42.7%, 25.3%, 11.5%, and %7.4 when the training and target datasets are roughly 2 years, 1 year, 2 months, and 1 month apart. We attribute this to a discrepancy in data distribution between the training and target datasets, *i.e.*, data drift, which we will investigate in the next subsection.

The same trend can be seen on the UCDavis CNN model. In fact, time changes seem to affect the UCDavis CNN model even more. For instance, when the training and the target datasets are 2 years apart, the accuracy of the UCDavis CNN model decays by 64.1%, whereas the TLS header part of the UW Tripartite model decays by 42.7%. Two aspects of the TLS header part of the UW Tripartite model could be contributing to its higher robustness to data drift: (i) more regularization layers, which prevents the model from overfitting to the training dataset, and (ii) feature engineering, in which the TLS handshake header bytes are used as input as opposed to any header bytes, reducing the noise in the model's input.

C. Traffic data drift

The considerable drop in both model's performances when they are trained on 07-2019 and tested on 2021 datasets, as well as the fact that the performance drop correlates with the

TABLE IV
ADOPTION OF HTTP/2 AND SPDY PROTOCOLS OVER TIME [19]

Protocol	2018-2019	2019-2020	2019-2021
HTTP/2	+ 40.6%	+ 31.0%	+ 53.5%
SPDY	- 93.4%	- 66.6%	- 83.3%

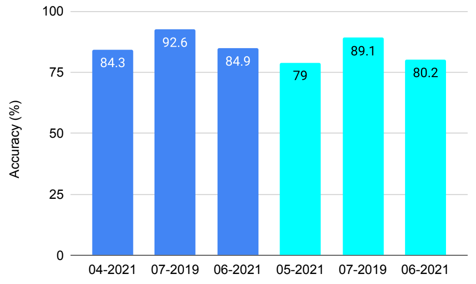

Fig. 5. Model's performance on datasets with similar sizes (dark blue: 04-2021 size, light blue: 05-2021 size)

difference in time of capture between datasets, could indicate that the valid data distributions that the models are learning may be changing with time, thus making the learned patterns obsolete. To investigate this, we take a closer look at the layer-5 protocol distributions in the datasets, looking for any time-related changes that we could recognize. Note that since our data is encrypted, only some application-layer protocols could be identified.

Table IV shows the result of this investigation. From 2018 to 2021, the adoption of HTTP/2 rose while the usage of SPDY, which is the predecessor to HTTP/2, drastically decreased. From 2019 to 2021, we can see that there is a 83.3% decrease in the usage of SPDY and a 53.5% rise in the adoption of HTTP/2. This may in part explain the drift in the datasets and, in particular, the different patterns in the raw header bytes, from 2019 to 2021. Unlike HTTP/2, SPDY uses a dynamic compression algorithm in the headers that makes it more vulnerable to chosen plain text attacks. Therefore, SPDY leads to more information leakage than HTTP/2 and is easier to classify.

To analyze this issue further in our datasets, we expect to see a change on the accuracy of the model even when it is trained and tested on the newest datasets. This is because we expect to see better results on the 07-2019 datasets where there could be considerably more SPDY flows than the 2021 datasets.

We know that in DL models the size of the datasets has a direct impact on the overall classification accuracy. Therefore, in order to have a fair comparison, we reduced the number of flows in the 07-2019 dataset to the number of flows in the 04-2021 and 05-2021 datasets. Since we reduce the dataset using random sampling, we perform multiple experiments and report the average accuracy. The results are depicted in Fig. 5. As can be seen, the accuracy of the model on the reduced 07-2019 datasets is around 8% to 10% higher than on the other datasets. This suggests that the TLS headers in the 07-2019 dataset are easier to classify than the TLS headers in the newer datasets.

To confirm our hypothesis about the impact of the application-layer protocols, we conduct experiments based on

TABLE V
DISTRIBUTION OF ALPN FIELD VALUES FOR DIFFERENT DATASETS

ALPN filter	Dataset		
	07-2019	09-2020	Merged-2021
HTTP/2	0.12	0.09	0.09
HTTP/1	0.25	0.15	0.14
Missing ALPN	0.62	0.76	0.77

the Application-Layer Protocol Negotiation (ALPN) field of the TLS protocol. Table V shows the distribution of ALPN field values for different datasets. Note that all the 2021 datasets are merged. There are two main reasons for doing this: (i) 07-2019 and 09-2020 datasets consist of roughly 119K and 89K flows, respectively. In contrast, the 2021 datasets are considerably smaller and merging them results in 98.9K flows, which is comparable in size to the 09-2020 and 07-2019 datasets; (ii) 2021 datasets are captured in a closer time frame, which makes their data patterns rather similar as shown in Fig. 4.

From Table V, it is evident that between 62%-77% of flows in the considered datasets do not have an ALPN field value (*i.e.*, missing ALPN). Moreover, around 10%-20% of flows consist of HTTP/1 and HTTP/2 application-layer protocols, which are only a small portion of flows in each dataset. Therefore, we evaluate model performance in three different scenarios, where the flows in the datasets are either HTTP/1, HTTP/2, or Unknown. It is unknown for a flow with no ALPN if it uses HTTP/1, HTTP/2 or neither HTTP/1 nor HTTP/2. Table VI illustrates the performance of the TLS header part of the UW Tripartite model on each dataset based on the ALPN field value. For HTTP/1 and HTTP/2, model performance across the datasets is more or less the same. However, the performance gap between the 07-2019 dataset and other datasets on flows with missing ALPN is considerable. Specifically, the model achieves around 93.2% accuracy on the flows with missing ALPN extracted from the 07-2019 dataset, while the performance is around 81% on the other datasets. This further substantiates that the TLS headers in the 07-2019 dataset are easier to classify, and the majority of this ease comes from flows with missing ALPN.

By examining the ALPN of all the datasets, we found a few flows with application-layer protocols other than web protocols (*e.g.*, Apple push-notification). Interestingly, the 07-2019 dataset is the only dataset that contains flows with the ALPN fields indicating the SPDY protocol. Recalling from the Table IV, in the time frame corresponding to the 07-2019 dataset SPDY was still highly used, which we speculate as the reason for superior classification performance on the missing ALPN portion of this dataset. Additionally, from Table IV it can be seen that from 2019 to 2021 the adoption of HTTP/2 has increased by more than 83.3%, which substantiates previous findings.

For a fair comparison, we reduce the number of HTTP/1, HTTP/2, and missing ALPN flows in each dataset to the smallest across all the datasets (*i.e.*, the number of HTTP/2 flows in the 05-2021 dataset). The results are depicted in Fig. 6. It is evident that the TLS header part of the UW

TABLE VI
PERFORMANCE OF THE TLS HEADER PART OF THE UW TRIPARTITE MODEL BASED ON ALPN

Dataset	Accuracy (%)		
	HTTP/2	HTTP/1	Missing ALPN
07-2019	93.5	97.5	**93.2**
09-2020	94.6	94.8	80.7
Merged-2021	91.6	96.9	81.1

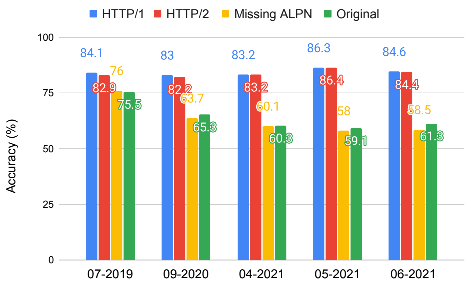

Fig. 6. Model performance with ALPN filter on the same size datassets

Tripartite model yields similar performance on HTTP/1 and HTTP/2 protocols. The accuracy is over 80% for all datasets on either HTTP/1 or HTTP/2 flows. However, the model shows inferior performance, *i.e.*, around 60% average accuracy on the missing ALPN portion of the datasets, except for the 07-2019 dataset which has a relatively higher accuracy of around 75%. Additionally, for the 09-2020 dataset, the performance of the model is lower than 07-2019 and higher than 04-2021 datasets. These results are all in line with the increase in the adoption of HTTP/2 and decrease in SPDY usage over time in Table IV. This further substantiates our hypothesis that the missing ALPN portion in the 07-2019 dataset is easier to classify. As the majority of the original flows (*i.e.*, no filter on the ALPN) are from the missing ALPN (*i.e.*, Unknown) portion, the performance of the model on the original flows is similar or slightly better than the missing ALPN flows alone. It is better because of the small portion of HTTP/1 or HTTP/2 flows available in the original dataset compared to missing ALPN flows.

Now we investigate whether the model is biased on the ALPN field or not. Indeed, this could lead to better model performance when the ALPN field value is either HTTP/1 or HTTP/2. To investigate this, we obfuscate the ALPN field in the raw traffic bytes (*e.g.*, replace with random bytes) and re-pre-process the data. We re-evaluate the TLS header part of the UW Tripartite model with the obfuscated ALPN field on HTTP/1 and HTTP/2 flows. As shown in Fig. 7, the ALPN field has an impact on classification performance, with lower performance when it is obfuscated. However, the performance degradation is only around 1%-2% in accuracy. For example, on the 04-2021 dataset, the model achieves 83.2% and 81.3% accuracy on HTTP/1 with clear ALPN and obfuscated ALPN, respectively. For HTTP/2, the accuracy is 83.25% versus 82.8%. Note that we leverage datasets with similar sizes as before and present average accuracy across multiple experiments. Hence, we show that a clear ALPN field is not the primary reason behind the model's performance gap between HTTP/1 and HTTP/2 flows with known ALPN, and

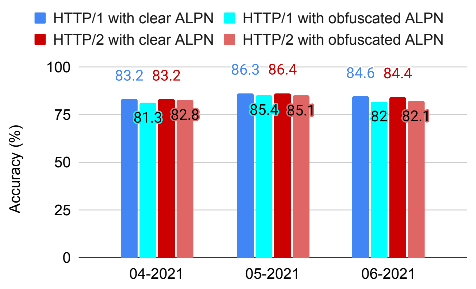

Fig. 7. Impact of ALPN obfuscation on the performance of the TLS header part of the UW Tripartite model

TABLE VII
ACCURACY OF THE TLS HEADER PART OF THE TRIPARTITE MODEL WHEN ALL DATASETS ARE MERGED BASED ON THE ALPN FILTER

ALPN	HTTP/1 or HTTP/2	Uknown
Accuracy (%)	95.2	83.0

the other flows with missing ALPN.

There are more protocols over TLS than HTTP/1 and HTTP/2 (*e.g.*, Apple push-notification), and new and updated web protocols are likely to emerge over time. However, HTTP/1 and HTTP/2 are well established standard web protocols, and it is plausible to assume that the model's performance over HTTP/1 and HTTP/2 protocols will remain rather consistent across different datasets in comparison to the unknown protocols. The results in Fig. 6 support this claim with similar model performance for HTTP/1 and HTTP/2 web protocols. Also, these results can be attributed to the existence of more information in flows that contain web traffic (*e.g.*, HTTP/1 and HTTP/2) compared to other protocols (*e.g.*, Apple push-notification). Therefore, web-related flows are easier to classify for the model.

Another hypothesis is that due to the negligible changes of the established protocols over time, training the model on all historical HTTP/1 and HTTP/2 improves the model's accuracy, while training it on all Unknown flows confuses the model despite the large number of samples in the dataset. To test this hypothesis, we merge all HTTP/1 and HTTP/2 flows of all datasets in one dataset, and all Unknown flows of all datasets in another dataset. Table VII illustrates the accuracy of the TLS header part of the UW Tripartite model on the HTTP/1 and HTTP/2 flows of all the datasets versus the merged unknown portion of all datasets. We see that the model shows an accuracy of 95.2% on the first dataset, compared to an accuracy of 83.04% on the second one. We also notice that the accuracy on the Unknown portion is low, despite the large number of flows. Therefore, it seems that training the model on a merged dataset of HTTP/1 and HTTP/2 flows helps with its performance, whereas training the model on more unknown flows seems to confuse the model, possibly because of the more varied patterns and protocols in that portion of the dataset.

V. ARCHITECTURE ADAPTATION

In this section, we examine the performance of the UW Tripartite model on the 2021 datasets. Observing a drop in model accuracy, we suggest updating the model architecture that improves accuracy on several datasets, thus making it more robust to data drift.

TABLE VIII
ACCURACY OF THE UW TRIPARTITE MODEL AND ITS DECOMPOSED PARTS ACROSS THE 2021 DATASETS

Model	Accuracy (%)		
	04-2021	05-2021	06-2021
FHA	40.0	83.4	87.1
F	11.0	81.0	85.9
H	84.3	79.0	85.8

A. Ensuring model convergence

We start by training and testing the UW Tripartite model and its decomposed parts on datasets from 2021. We skip the auxiliary part of the model as its performance is negligible compared to the other two parts. The results of these experiments are shown in Table VIII.

Although suffering a drop from the baseline 2019 dataset, the accuracy of the Tripartite model is reasonable at 83.4% and 87.1% on 05-2021 and 06-2021 datasets, respectively. However, the model has a rather peculiar accuracy of only 40% on the 04-2021 dataset, which is primarily attributed to the flow time-series part of the model, showing a mere accuracy of 11% (*i.e.*, worse than a random classifier). On the other hand, the TLS header part of the model performs reasonably on the same dataset.

Before we delve into the reasons for the under performance of flow time-series part of the UW Tripartite model, we note that the lower performance of the model on 2021 datasets compared to 2019 dataset can be attributed to dataset size. 07-2019 dataset had 119K labeled flows, whereas 04-2021, 05-2021 and 06-2021 datasets have 42K, 17K and 51K flows, respectively. Therefore, given a much larger amount of training data, we expect the model to achieve a higher accuracy on 07-2019 dataset regardless of the architecture. However, the dismal accuracy on 04-2021 dataset cannot be simply explained by dataset size, and has to do with the model itself.

To troubleshoot the flow time-series part of the model on 04-2021 dataset, we examined the confusion matrix and accuracy of the model in the training phase, epoch by epoch. We found that the model does not converge, and the same class is predicted for all samples in each epoch. We tried two alterations to the model to alleviate this problem: (i) Learning rate reduction—The learning rate for the optimizer was reduced from the default value of 0.001 [2] to 0.0001 (*i.e.*, 10x reduction); (ii) Masking layer addition—A Masking Layer was added at the beginning of the flow time-series part of the model. The Masking Layer acts as a de-noising layer to filter out time-steps that don't have any information. Therefore, these time-steps can be skipped in the LSTM layer.

The above alterations boosted the accuracy of the flow time-series part of the model on the 04-2021 dataset from 11% to 88.3%. A smaller learning rate makes it more likely for the model to eventually converge to global optima, although it increases training time. A masking layer reduces data noise, while adding to the complexity of the model. Despite the

TABLE IX
MODEL ADAPTATION BEST PRACTICES FOR FLOW TIME-SERIES PART

Dataset	Adaptation	Training flows	Accuracy (%)
04-2021	Dropout + Learning Rate	33,900	89.4
	Dropout + Learning Rate + Masking Layer	33,900	90.1
05-2021	Dropout	14,024	87.3
	BLSTM + Dropout	14,024	88.2

downsides, evidently, in the case of the 04-2021 dataset, these alterations are necessary for the model to achieve reasonable performance.

B. Adjusting to dataset size

Given that dataset size can contribute to the model's drop is accuracy on the 2021 datasets, we suggest a number of best practices in designing a model architecture for smaller datasets, based on a number of experiments carried out on the two smallest datasets, *i.e.*, 04-2021 and 05-2021.

1) Dropout rate reduction: The stacked LSTM in the UW Tripartite model is followed by a dropout layer. The dropout layer randomly sets the units of LSTM output to zero based on the dropout rate, which is often used to avoid model overfitting. We found that in a smaller dataset, a high dropout rate does not help, as it sets units of valuable information to zero, thus leaving the final layers of the model with little information to work with. By reducing the dropout rate from 0.5 (*i.e.*, default in [2]) to 0.3, we saw a boost in model accuracy on both 04-2021 and 05-2021 datasets, the two smallest datasets, as shown in Table IX.

2) Flow time-series part simplification: The stacked LSTM layer proposed in [2] is a complex model with too many parameters for a small dataset. By reducing the number of LSTM layers by one, thus turning the stacked LSTM to a bidirectional LSTM (BLSTM), we were able to obtain better results on datasets smaller than 20K flows, as shown for 05-2021 dataset in Table IX. We further found that on datasets smaller than 10K flows, even reducing the stacked LSTM layer to a 1D Convolution layer helps with the model's performance, contrary to what was shown in [2] for large datasets.

3) Best practices: Table X summarizes our recommended best practices based on a given dataset's size. We suggest that when using the UW Tripartite model, the flow time-series part of the model should be adapted to the training dataset's size. When there are fewer than 50K training flows, reducing the dropout layer value (*e.g.*, 0.3) is sufficient. If the number of samples are fewer than 20K, a simpler architecture such as BLSTM is preferred over stacked LTSM. In the UW Tripartite model architecture shown in Fig. 1, changing the stacked LSTM to a BLSTM would simply remove the last LSTM layer in the stack, as each LSTM works in reverse direction to the previous one. Finally, if the dataset has fewer than 10K flows, using simple 1D Convolutions (*i.e.*, mentioned in [2] Appendix) is adequate and preferable over the LSTM layer.

C. QUIC results

We also evaluate the performance of the UW Tripartite model on real-world QUIC data, before and after employing the adaptation guidelines proposed in the previous subsection.

TABLE X
MODEL ARCHITECTURE ADAPTATION RULES FOR THE FLOW TIME-SERIES PART OF THE UW TRIPARTITE MODEL

Number of flows	Adaptation
<= 50K	Dropout reduction
<= 20K	BLSTM
<= 10K	1D Convolutions [2]

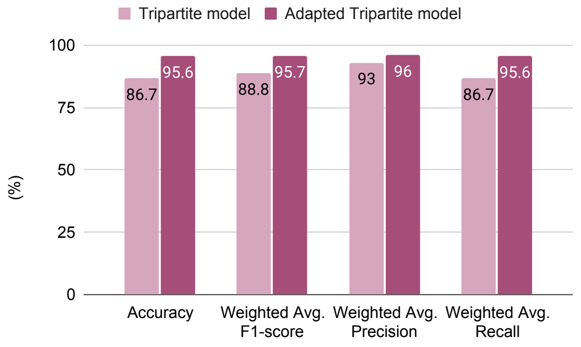

Fig. 8. Performance for flow time-series part of the UW Tripartite model with and without adaptations on the QUIC-05-2021 dataset

We show that these guidelines indeed improve the accuracy of the UW Tripartite model on a dataset consisting of QUIC flows, thus proving that our adaptation practices generalize to encrypted protocols other than TLS.

The flow time-series part of the UW Tripartite model was shown to acquire over 99% accuracy on a synthetic QUIC dataset [2]. However, on our real-world QUIC data, *i.e.*, QUIC-05-2021, the model obtained 86.7% accuracy. Therefore, we chose the following architectural adaptations for the model: (i) decreasing the initial learning rate, (ii) adding a masking layer, and (iii) reducing the dropout rate to 0.4. Fig. 8 shows the performance of the flow time-series part of the model before and after these adaptations.

The model achieves an accuracy of 86.7% before adaptation, whereas the adapted model achieves 95.6%. A similar trend is visible in other performance metrics, such as weighted average F1-score, precision, and recall, where the adapted model outperforms the original UW Tripartite model's flow time-series part by 3% to 9%. The precision for both models is quite high, however, the main advantage of the adapted model is correctly predicting a larger portion of flows for each class, which results in a 9% increase in recall. Fig. 9 shows the confusion matrices of the flow time-series part of the UW Tripartite model and its adapted version. The recall increase is visible in the confusion matrices, where the adapted model achieves a higher accuracy per class. The most significant increase is for the Resources class with 10% increase in the accuracy.

VI. CONCLUSION

In this work, we investigated the effect of data drift on two state-of-the-art deep encrypted traffic classification models. We examined the robustness of these models to data drift, providing insights about the type of drift that occurs in network traffic data.

We showed that a model that operates on the traffic shape is more resilient to data drift than one that operates on TLS

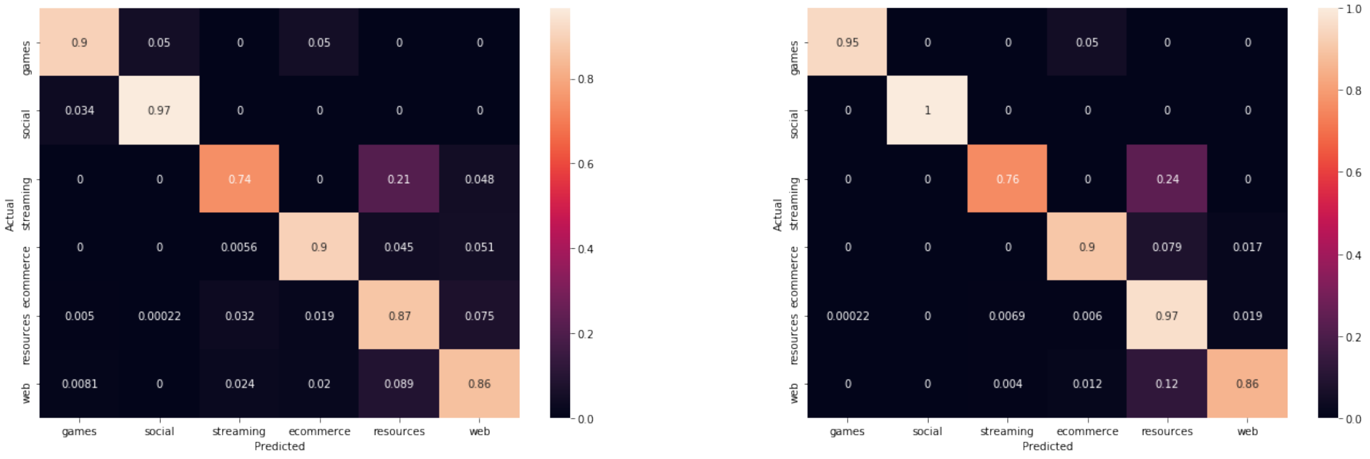

Fig. 9. Confusion matrix of the flow time-series part of the UW Tripartie model (left) vs. its adapted version (right) on the QUIC-05-2021 dataset

headers. Also, we examined the impact of model architecture and feature engineering on model robustness by comparing the two models over the same datasets. Additionally, we examined the impact of the application-layer protocols on model robustness, demonstrating that the model performance improves by selecting more stable protocols (*e.g.*, HTTP/1, HTTP/2) for the model to train on, regardless of dataset collection time.

To warrant the need for architectural adaptations, we showcased the performance and convergence issues that arise when a state-of-the-art architecture is trained on different datasets with no adaptations. We examined the performance of different parts of the model, as well as the effect of changing some of its structural parameters, to provide best practices for designing an architecture that performs well on unseen and possibly newer datasets. We showed the generalizability of our guidelines to different encryption protocols by testing the adapted architecture on a dataset of QUIC traffic.

The adaptation approaches proposed in this paper are manual. An automatic choice of parameters that leads to a robust classifier is a direction for future work. Another direction is to improve the generalizability of the classifier by using transfer learning or incremental learning methods that leverage previously learned knowledge, both to reduce training time and increase performance on new datasets.

References

[1] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2020.

[2] I. Akbari, M. A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "A look behind the curtain: Traffic classification in an increasingly encrypted web," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 1, 2021.

[3] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[4] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[5] Y. Kumano, S. Ata, N. Nakamura, Y. Nakahira, and I. Oka, "Towards real-time processing for application identification of encrypted traffic," in *2014 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2014, pp. 136–140.

[6] R. Alshammari and A. N. Zincir-Heywood, "Can encrypted traffic be identified without port numbers, ip addresses and payload inspection?" *Computer networks*, vol. 55, no. 6, pp. 1326–1350, 2011.

[7] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *International Conference on Passive and Active Network Measurement*. Springer, 2007, pp. 165–175.

[8] C. Bacquet, A. N. Zincir-Heywood, and M. I. Heywood, "Genetic optimization and hierarchical clustering applied to encrypted traffic identification," in *2011 IEEE symposium on computational intelligence in cyber security (CICS)*. IEEE, 2011, pp. 194–201.

[9] R. Bar-Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime classification for encrypted traffic," in *International Symposium on Experimental Algorithms*. Springer, 2010, pp. 373–385.

[10] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.

[11] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[12] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," *arXiv preprint arXiv:1708.06376*, 2017.

[13] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features." in *ICISSp*, 2017, pp. 253–262.

[14] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[15] M. Abbasi, A. Shahraki, and A. Taherkordi, "Deep learning for network traffic monitoring and analysis (ntma): a survey," *Computer Communications*, vol. 170, pp. 19–41, 2021.

[16] M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai, "Robust and rapid adaption for concept drift in software system anomaly detection," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 13–24.

[17] S. Saurav, P. Malhotra, V. TV, N. Gugulothu, L. Vig, P. Agarwal, and G. Shroff, "Online anomaly detection with concept drift adaptation using recurrent neural networks," in *Proceedings of the acm india joint international conference on data science and management of data*, 2018, pp. 78–87.

[18] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[19] W3Techs, "Historical yearly trends in the usage statistics of site elements for websites," Accessed Feb. 2022. [Online]. Available: https://w3techs.com/technologies/history_overview/site_element/all/y