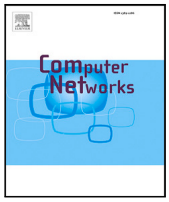
Computer Networks 225 (2023) 109648

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



# Deep learning for encrypted traffic classification in the face of data drift: An empirical study

Navid Malekghaini [a,*], Elham Akbari [a], Mohammad A. Salahuddin [a], Noura Limam [a], Raouf Boutaba [a], Bertrand Mathieu [b], Stephanie Moteau [b], Stephane Tuffin [b]

[a] David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
[b] Orange Labs, Lannion, France

ARTICLE INFO

*Keywords:*
Data drift
Encrypted traffic classification
Deep learning
Web traffic
HTTP/2
QUIC

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 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 for encrypted traffic classification 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 state-of-the-art encrypted traffic classifiers change on multiple real-world datasets collected over the course of two years from a major ISP's network. We investigate the changes in traffic data patterns highlighting the extent to which these changes, also known as data drift, impact the performance of the two models in service-level as well as application-level classification. We propose best practices for architecture adaptations to improve the accuracy of the model in the face of data drift. We show that our best practices are generalizable to other encryption protocols and different levels of labeling granularity.

## 1. Introduction

Deep learning (DL) models have shown superior performance in encrypted traffic classification [1–4]. 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 dataset. In practice, the model performance on a given dataset is tightly coupled with the intrinsic properties of the dataset. The effect of the target dataset on model accuracy has been previously highlighted by comparing the performances of different traffic classification models on varying datasets [2,5].

Data drift is a phenomenon in which the distribution of input data over classes changes with time. For example, a service may switch to another transport protocol leading to a different flow time series (*i.e.*, 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 leads to model decay, significantly impacting model performance.

In this paper, we study the effect of data drift on the performance of two state-of-the-art encrypted network traffic classifiers [1,2]. Using several real-world datasets collected from a major ISP's mobile network and consisting of traffic over both Transport Layer Security (TLS) [6] and Quick UDP Internet Connections (QUIC) [7] encryption protocols, 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 traffic input that the models struggle on. We also analyze the architecture of the models, offering guidelines for designing architectures that we empirically show are more robust to data drift. Guided by the observations 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, we also study the effect of dataset size on model performance. Our main contributions can be summarized as:

- We study the data drift phenomenon using five real-world TLS datasets collected over a course of more than two years from a major ISP's mobile network. To the best of our knowledge, we

* Corresponding author.
*E-mail addresses:* nmalekgh@uwaterloo.ca (N. Malekghaini), eakbaria@uwaterloo.ca (E. Akbari), mohammad.salahuddin@uwaterloo.ca (M.A. Salahuddin), noura.limam@uwaterloo.ca (N. Limam), rboutaba@uwaterloo.ca (R. Boutaba), bertrand2.mathieu@orange.com (B. Mathieu), stephanie.moteau@orange.com (S. Moteau), stephane.tuffin@orange.com (S. Tuffin).

https://doi.org/10.1016/j.comnet.2023.109648
Received 18 October 2022; Received in revised form 13 February 2023; Accepted 18 February 2023
Available online 23 February 2023

are the first to address the problem of data drift in real-world encrypted traffic classification.
- We provide insights into the type of data drift that happens in network traffic at different levels of labeling granularity, *i.e.*, service-level and application-level classes. These insights are useful to practitioners working with traffic classification models in production.
- We perform an ablation study to analyze the impact of data drift on two state-of-the-art features for encrypted traffic classification: (i) TLS header bytes, and (ii) flow time-series information. We reason data drift on these features, and quantify the drift per service class and corresponding applications.
- We offer guidelines for designing models that are robust to a change of dataset, labeling granularity, and encryption protocol. Our guidelines have the distinction of being empirically tested on real-world data with different encryption protocols and for both service- and application-level classification.

This work is an extension of [8]. The primary additions include the investigation of data drift on application-level classification, its impact on model performance and comparison to service-level classification. The efficacy of the proposed guidelines for robust model architectures are also showcased for application-level classification. The rest of paper is organized as follows. Section 2 presents the closely related works, while Section 3 presents the datasets and models used in the paper. In Section 4, 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 5, we present our insights and guidelines on designing a robust model architecture, along with the supporting results. Section 6 concludes the paper and outlines directions for future work.

## 2. 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 their statistics) that were difficult to tweak without affecting quality of service. Before the advent of DL, the performance of several traditional supervised ML models, such as Naïve Bayes, AdaBoost, Support Vector Machine (SVM), Decision Tree, and Random Forest, was evaluated using these features for encrypted traffic classification (*e.g.*, [9–11]). Furthermore, semi-supervised approaches based on Gaussian Mixture Models, k-Means, k-Nearest Neighbor clustering, and Multi-Objective Genetic Algorithms were studied for real-time encrypted traffic classification (*e.g.*, [12–14]). A survey of traditional ML approaches is available in [15,16].

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 DL models including Multi-Layer Perceptrons, Stacked Autoencoders, Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) (*e.g.*, [1,3,17,18]). The models were primarily evaluated using public mixed-protocol datasets such as ISCXVPN2016 [11] and ISCXIDS2012 [19]. The work in [5] 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 is available in [20].

Several works in Website fingerprinting (WF) attacks against The Onion Router (Tor) observe the need for training ML models on fresh traffic traces to ensure attack effectiveness. Herrmann et al. [21] show that their Naïve Bayes approach is robust to data drift when the training and test data were collected within two days of one another. Their approach works on packet size and direction sequences of flows similar to the flow time-series feature used in this paper. Rimmer et al. [22] evaluate the resilience of several state-of-the-art DL models to data drift on traffic periodically collected over two months. The authors show that different DL models age differently, with their accuracies dropping from around 95% to between 55% and 75% in the course of two months.

A critical study of WF attacks [23] evaluates the effect of data staleness on WF by measuring an SVM-based classifier's accuracy on the data over the course of 90 days. The data is gathered by crawling Alexa Top 100 websites at different instants in time. The authors show that the classifier's accuracy drops from around 80% to around zero in less than 90 days when the number of websites (*i.e.*, the number of classes) is 100, with the accuracy dropping below 50% in less than 10 days. To provide a solution to the data staleness problem, the authors in [24,25] propose models that use less data to train, so that crawling the websites and collecting the necessary traces to re-train the model is feasible in the small window of time in which the model decays.

Andresini et al. [26] address robustness to data drift for intrusion detection in network traffic, a context in which data drift is especially important because of the continuously evolving nature of attacks. Their proposed approach consists of three phases: (i) initial training on historical data, (ii) incremental learning on unlabeled data facilitated by a learned oracle, and (iii) an explanation phase for how the model adapts to new attack categories. The authors use variable length time windows that span several minutes rather than time splits to evaluate the model. Their approach is evaluated on a recently published version of the CICIDS2017 dataset [27], a dataset of benign and malware traffic traces spanning over 5 days. The time window they consider is much shorter than the time windows considered in this paper. Furthermore, their domain is also different to ours, *i.e.*, intrusion detection versus service or application detection.

Ma et al. [28] propose a framework to detect and adjust to data drift in an anomaly detection system. The authors define data 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 data drift detection algorithm to free operators from manually tuning per-KPI parameters. Their data 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 data drift algorithms to their domain is rather straightforward.

Saurav et al. [29] 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 data drift, *e.g.*, sudden, gradual and incremental.

Taylor et al. [30] 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: (i) time of collection, (ii) app device, and (iii) 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 conclude 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 [30] is based on traditional ML models, it relates to our work 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 [30], our work is based on real-world datasets.

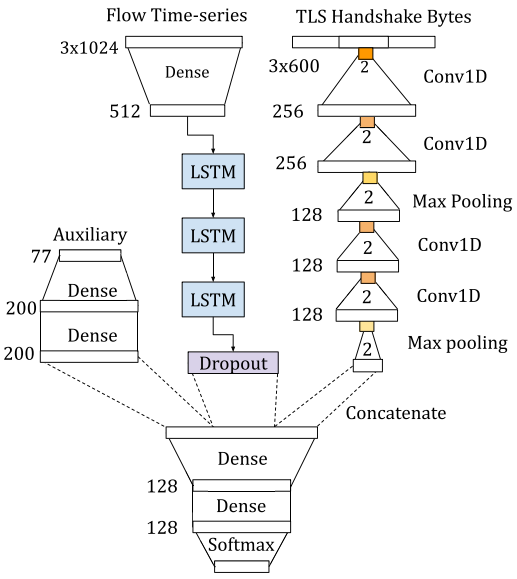



**Fig. 1.** UW model architecture [2].

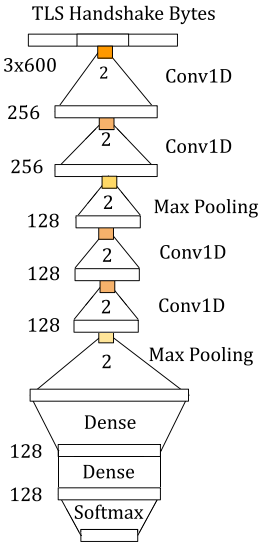



**Fig. 2.** UW-H, *i.e.*, decomposed TLS header part of UW model.

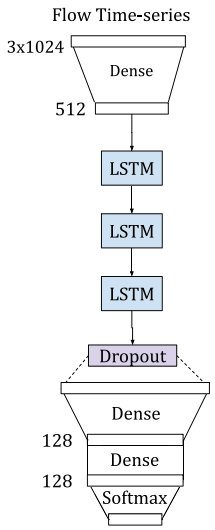



**Fig. 3.** UW-F, *i.e.*, decomp. flow time-series part of UW model.

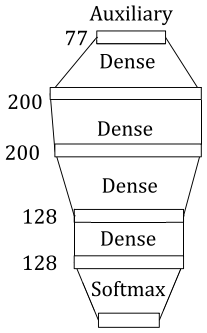



**Fig. 4.** UW-A, *i.e.*, decomposed statistical part of UW model.

## 3. Methodology

### 3.1. Deep learning models

#### 3.1.1. UW tripartite model

The University of Waterloo Tripartite model (UW) is a DL model proposed in our previous work [2]. The UW model achieves an accuracy of over 90% on purely encrypted TLS network traffic. It is a three-part model, as depicted in Fig. 1, where each part is designed to operate on a different type of input data. Note that the orange and yellow boxes in the figure depict convolution and max-pooling layer kernels, respectively. Furthermore, each layer's output vector is depicted by a white box accompanied by its size.

Firstly, 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. Secondly, the model 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. We refer to these statistical features as auxiliary features. Our experiments suggest that the auxiliary features 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 final classification.

To the best of our knowledge, the UW model obtains the highest accuracy to date on a fully encrypted dataset, for service-level classification. In this paper, we perform an ablation study on the different parts of the UW model. The decomposed parts of UW, *i.e.*, for TLS header bytes (UW-H), flow time-series information (UW-F), and auxiliary features (UW-A), are depicted in Figs. 2, 3, and 4, respectively.

#### 3.1.2. UCDavis CNN model

The authors in [1] propose a CNN model for early classification of network flows. Their CNN model operates on the first six packets of a flow, for each of which, the first 256 raw bytes from L3 and above are extracted and concatenated together to form the input feature vector. The model consists of convolutional, max-pooling and dense layers as shown in Fig. 5. We leverage the UCDavis CNN model in this paper, as it was shown in [2] that after the UW model, the UCDavis CNN obtains the best accuracy on their fully encrypted dataset among a number of evaluated models.

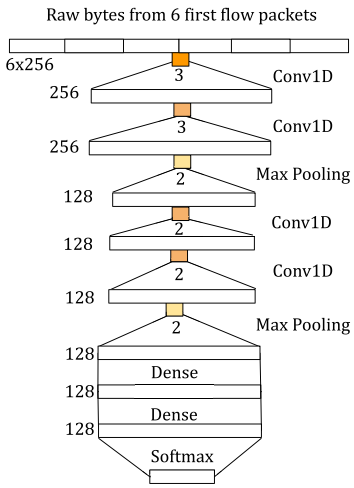



**Fig. 5.** UCDavis CNN architecture [1].

**Table 1**
Service-level datasets properties.

| Protocol | Dataset | 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 |

**Table 2**
Application-level datasets properties.

| Protocol | Dataset | Total flows (K) | Labeled flows (K) | Labeled flows (%) |
|---|---|---|---|---|
| TLS | 07-2019 | 762.7 | 83.1 | 10.9 |
| | 09-2020 | 411.7 | 59.8 | 14.52 |
| | 04-2021 | 284.8 | 26.3 | 9.2 |
| | 05-2021 | 124.0 | 11.1 | 9.0 |
| | 06-2021 | 261.2 | 34.6 | 13.2 |
| QUIC | QUIC-05-2021 | 37.8 | 9.3 | 24.6 |

### 3.2. Datasets description

We use a total of six real-world datasets in this paper which consist of TLS and QUIC traffic traces collected from a major ISP's mobile network. The source and destination IP addresses are obfuscated and the packets are truncated after 400 bytes, except for the TLS handshake packets. A flow is assumed to be a quintuple of source IP, destination IP, source port, destination port and protocol.

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 raw TLS header bytes from the flows, as well as flow time-series information consisting of 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 [11]. The auxiliary features include statistical information about flows, *e.g.*, mean, median, minimum, and maximum of packet sizes in each direction.

The labeling module is used to label the flows according to the Server Name Indication (SNI) field. The flows are labeled at two levels of granularity: (i) service level, and (ii) application level. Service-level labels consist of 8 classes each representing a service category, including *chat*, *download*, *games*, *mail*, *search*, *social*, *streaming*, and *web*. For each service level, there is a corresponding set of applications. For example, the *mail* class consists of *mailGmail*, *mailHotmail*, and *mailOutlook* applications. There are a total of 19 applications, which act as a finer level of labeling per service class. Note that not all the applications in a service class have enough flows to be categorized as an application class. Therefore, the number of labeled flows in the application level is smaller than service level. The service-level classes and corresponding applications are presented in Table 3. The *games* service class does not have corresponding application classes, as it consists of many applications with a very small number of flows. Nevertheless, these applications' flows together form the *games* class at the service level.

**Table 3**
Service-level and corresponding application-level classes for TLS datasets.

| Service-level class | Application-level classes | | | |
|---|---|---|---|---|
| chat | Facebook | Snapchat | Whatsapp | – |
| download | Apple | GooglePlay | – | – |
| mail | Gmail | Hotmail | Outlook | – |
| search | Google | – | – | – |
| social | Facebook | Instagram | Twitter | – |
| streaming | Facebook | Netflix | Snapchat | Youtube |
| web | Amazon | AppleLocalization | Microsoft | – |
| games | – | – | – | – |

The employed datasets can be categorized into two types based on encryption protocol, *i.e.*, *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 MM-YYYY format, *i.e.*, 07-2019, 09-2020, 04-2021, 05-2021, and 06-2021, respectively.

*(ii) QUIC dataset*: The QUIC dataset, *i.e.*, QUIC-05-2021, is extracted from a packet trace of QUIC traffic captured at the same time as the TLS 05-2021 dataset. The TLS handshake bytes are tightly coupled to the TLS protocol and thus irrelevant to QUIC. Therefore, the QUIC dataset only consists of flow time-series information. Auxiliary data was not added to this dataset as the effect of flow statistics on model performance was negligible in our experiments. The QUIC dataset is used to show that our architecture adaptation best practices, which are centered around the UW-F model, generalizes to non-TLS encrypted data (*cf.*, Section 5).

Tables 1 and 2 show the total number of flows, labeled flows, and the percentage of labeled flows, for service-level and application-level datasets, respectively. The number of labeled flows depict the size of each dataset. The percentage of labeled flows in each dataset highlights the performance of the employed labeling module for the TLS flows. Evidently, the percentage of the labeled flows across the datasets are more or less inline with each other, asserting the suitability of the labeling module. Additionally, the labeled distribution of TLS flows for service and application classes are depicted in Figs. 6 and 7, respectively. There is insignificant difference (*i.e.*, 0.01 average standard deviation) in class distribution across the datasets. Hence, accuracy (*cf.*, Section 3.3) is a suitable performance metric for comparing the models across the datasets. To deal with class imbalance we adopt a weighting strategy, *i.e.*, we up-sample classes with smaller number of flows.

There are several interesting takeaways when we compare the distribution of application classes. Notably, downloadApple has the highest number of flows. Furthermore, streamingNetflix has the lowest number of flows among all applications. This implies that although Netflix is an extremely popular application, not many users watch Netflix on their mobile devices when compared to Snapchat, Facebook, and Youtube. Recall that the datasets were captured on the ISP's mobile network.

For labeling the QUIC dataset, we update the classes in the TLS dataset. Since QUIC is still not widely adopted by services across the Web, not all classes from the TLS dataset have enough flows 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, justifying the absence of flows labeled as *download* in the QUIC dataset. Hence, we keep the *games*,

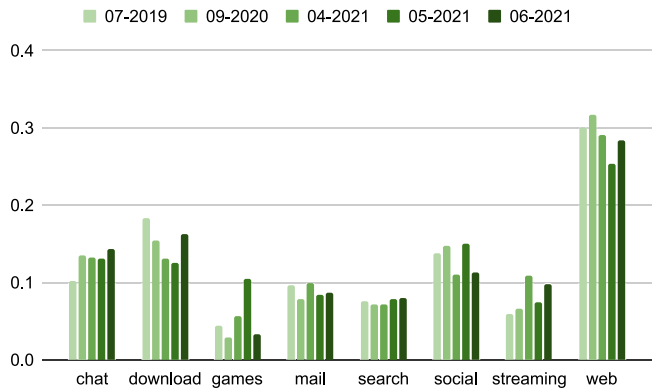



**Fig. 6.** Service-level class distribution of the TLS datasets.

*social*, *streaming*, and *web* classes, while adding some new classes, *i.e.*, *e-commerce* and *resources*. The *resources* class corresponds to the flows that are essentially shared among different websites that mostly deliver tools, such as JavaScript APIs or design content for websites. The QUIC labeling module can label up to 68% of the flows, a large improvement over the less than 20% labeling performance on the TLS datasets. We attribute this to fewer services using QUIC and most of them corresponding to the *resources* class. Therefore, the SNIs are not as varied in this dataset as they are in the TLS datasets.

### 3.3. Software stack and performance metrics

The software stack for data pre-processing, model training, and evaluation includes Tensorflow [31] with Keras API [32], CUDA, PySpark [33], 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 paper, 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.

Another metric that we use is top-k accuracy. This metric measures how often the model is able to predict the right class in the first k guesses. For instance, top-1 accuracy corresponds to the accuracy metric defined above. The logarithmic weighted mean of top-k accuracies is the weighted average of top-k accuracies, where the weights logarithmically decrease as k increases.

## 4. Investigation

In this section, we study the performance of the UW model when trained on a baseline TLS dataset and tested on a different, target TLS dataset. We investigate model decay in time by leveraging the decomposed models and experimenting with different TLS datasets. A summary of key findings in this section is available in Table 12.

**Table 4**
Model performance on the 07-2019 TLS dataset in service-level classification.

| Dataset | Accuracy (%) | | | |
|---|---|---|---|---|
| | UW | UW-H | UW-F | UW-A |
| 07-2019 | 94.5 | 94.8 | 86.3 | 43.8 |

**Table 5**
Model performance on the 07-2019 TLS datasets in application-level classification.

| Dataset | Accuracy (%) | | | |
|---|---|---|---|---|
| | UW | UW-H | UW-F | UW-A |
| 07-2019 | 95.7 | 96.0 | 84.2 | 29.1 |

### 4.1. Baseline performance

We start by highlighting the performance of the UW model on the 07-2019 dataset, which is our oldest and largest TLS dataset. For insight into the performance of each part of the UW model separately, we also conduct experiments on the decomposed models (*i.e.*, UW-H, UW-F and UW-A). Table 4 shows the performance of these models in service-level classification.

We notice that when we train the decomposed models on the 07-2019 dataset, the UW-H model shows the highest accuracy, which is 0.3% higher than the accuracy of the UW model on the same dataset. We attribute this to more data leakage in TLS headers at the time of the corresponding dataset collection (*cf.*, Section 4.3). The UW-A model shows the lowest accuracy of 43.8%. With such a low accuracy, it is evident that the flow statistics are not helping but rather confusing the UW model, resulting in an even inferior performance to the UW-H model.

The results for application-level classification are depicted in Table 5. These results concur with the previous findings, with similar trends for UW and decomposed models. Again, the UW-H model achieves the highest accuracy of 96%, which is better than the UW model, while the auxiliary input achieves the worst performance, *i.e.*, 29.1% in accuracy.

### 4.2. Robustness to performance decay

We study the performance of the UW model in service-level classification on different target (or test) TLS datasets after training it on the baseline 07-2019 dataset. The target datasets, *i.e.*, 09-2020, 04-2021, 05-2021, 06-2021, were collected at different points in time upto two years from the 07-2019 dataset. As our experiments have shown that the performance of UW-A is inferior with little to no impact on UW model performance, we focus our study on the UW-H and UW-F models.

The results of the first set of experiments is shown in Fig. 8. Evidently, the prediction ability of the model decays over time, which is quantified in Table 6. We see that the performance decay of the UW 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 two years apart.

Model performance decay over time is an expected phenomenon. Nevertheless, we see that it does not have an equal impact on the UW-H and UW-F models. In fact, the performance of UW-H decays 7% more on average than the performance of UW-F (*i.e.*, 40.75% compared to 33%). This suggests that using the traffic shape features, which is captured by the UW-F input, makes the classifier comparatively 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 model.

The previous experiments also highlight that performance decay correlates with the time difference between the training and target datasets. Therefore, we run experiments to further investigate this observation. In particular, we train UW-H using different datasets, *i.e.*, 07-2019, 09-2019, 04-2021 and 05-2021, and measure how much the

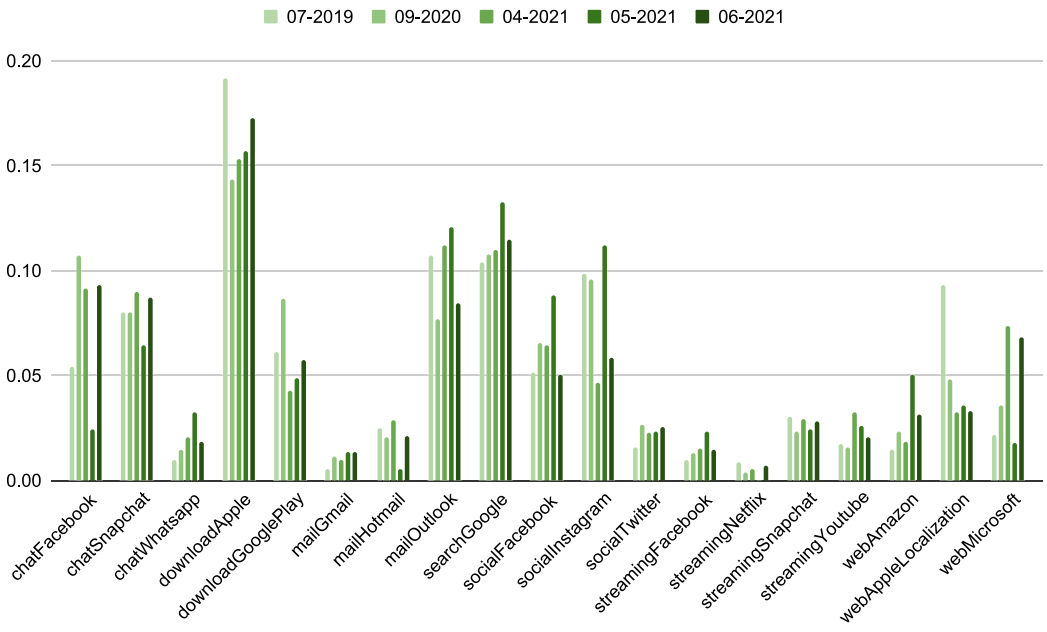



**Fig. 7.** Application-level class distribution of the TLS datasets.

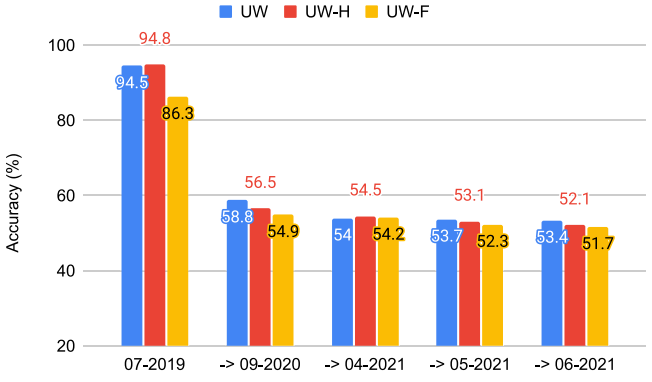



**Fig. 8.** Model performance when trained on baseline 07-2019 dataset and tested (notation →) on target datasets.

**Table 6**
Drop in model's predictive accuracy when trained on the baseline 07-2019 dataset and tested on subsequent datasets.

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

performance of the trained model decays by 06-2021. We conduct the same set of experiments with the UCDavis CNN model and compare the performance of both models. Given that the size of training set has an impact on the accuracy of a DL model, we down-sample the training datasets to the size of the smallest dataset (*i.e.*, 05-2021 dataset) and average the results on all samples.

The performance decay in decreasing order of time span is shown in Fig. 9. It is evident that the closer the datasets are in time of capture, the lower the performance decay of the UW-H model. For example, the accuracy of UW-H in service-level classification decays by 43.9%, 26.1%, 10.3%, and 7.2% roughly after 2 years, 1 year, 2 months, and 1 month, respectively. 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 for the UCDavis CNN model up to 04-2021, although the performance decay is even more noticeable than on UW-H. For instance, when the training and target datasets are 2 years apart, the accuracy of the UCDavis CNN model decays by 49.6%, compared to 43.9% for UW-H in service-level classification. Two aspects of the UW-H model could be contributing to its comparatively higher robustness to data drift: (i) more regularization layers (*i.e.*, higher dropout rates and L2 kernel regularization for the dense 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. We note that the performance decay of the UCDavis CNN model in service-level classification is lower in the span of 2 months (*i.e.*, between 04-2021 and 06-2021) compared to the span of 1 month (*i.e.*, between 05-2021 and 06-2021), hence breaking the previous trend. We speculate that the model simply overfits the training dataset rather than naturally decay as data drifts over time. This also suggests that the UCDavis CNN model might be less generalizable than the UW-H model.

Fig. 9 also shows that the accuracy of UW-H in application-level classification decays by 40.5%, 26%, 6.5%, and 3.7% over the span of 2 years, 1 year, 2 months, and 1 month respectively, similar to service-level classification. The performance decay of the UCDavis CNN model in application-level classification also follows the same trend as in service-level classification. Specifically, the drop in accuracy decreases from 59.9% over the span of 2 years, to 27.8% over the span of 1 year, to 10.2% over the span of 2 months, and increases again to reach 13.2% when the datasets are 1 month apart. Interestingly, the decay is much worse with the UCDavis CNN model than UW-H in application-level classification. This suggests that the UCDavis CNN model is even more susceptible to data drift and overfits to the training datasets in application-level classification.

In the following, we investigate which service classes are most affected by data drift. Furthermore, we investigate what a given service is confused with, as time passes. Finally, we investigate whether this confusion holds across different architectures or not. We strategically focus our study on two particular scenarios. The first is when the training and testing datasets are 2 years apart, *i.e.*, the model is trained on the 07-2019 dataset and used to classify the 06-2021 dataset. That

**Table 7**
Per-service class accuracy of UW-H and UCDavis CNN on (a) the 04-2021 dataset, and (b) the 07-2019 dataset.

| (a) | | | (b) | | |
|---|---|---|---|---|---|
| Class | Accuracy (%) | | Class | Accuracy (%) | |
| | UW-H | UCDavis CNN | | UW-H | UCDavis CNN |
| chat | 77 | 84 | chat | 96 | 92 |
| download | 86 | 82 | download | 95 | 89 |
| games | 95 | 82 | games | 97 | 88 |
| mail | 83 | 89 | mail | 97 | 95 |
| search | 87 | 83 | search | 99 | 95 |
| social | 82 | 82 | social | 96 | 93 |
| streaming | 88 | 82 | streaming | 93 | 82 |
| web | 86 | 79 | web | 92 | 91 |

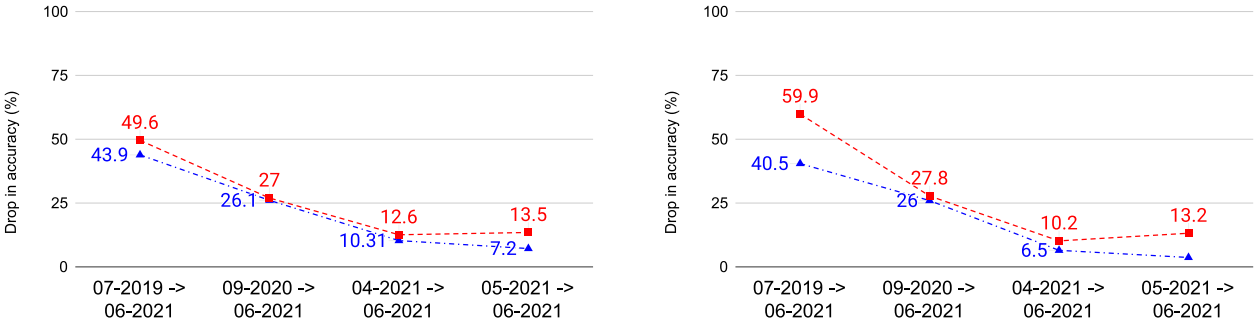



**Fig. 9.** Performance decay of UW-H and UCDavis CNN in service-level classification *(left)* and application-level classification *(right)*.

(a)

| Class | Accuracy (%) | |
|---|---|---|
| | UW-H | UCDavis CNN |
| chat | 77 | 84 |
| download | 86 | 82 |
| games | 95 | 82 |
| mail | 83 | 89 |
| search | 87 | 83 |
| social | 82 | 82 |
| streaming | 88 | 82 |
| web | 86 | 79 |

(b)

| Class | Accuracy (%) | |
|---|---|---|
| | UW-H | UCDavis CNN |
| chat | 96 | 92 |
| download | 95 | 89 |
| games | 97 | 88 |
| mail | 97 | 95 |
| search | 99 | 95 |
| social | 96 | 93 |
| streaming | 93 | 82 |
| web | 92 | 91 |

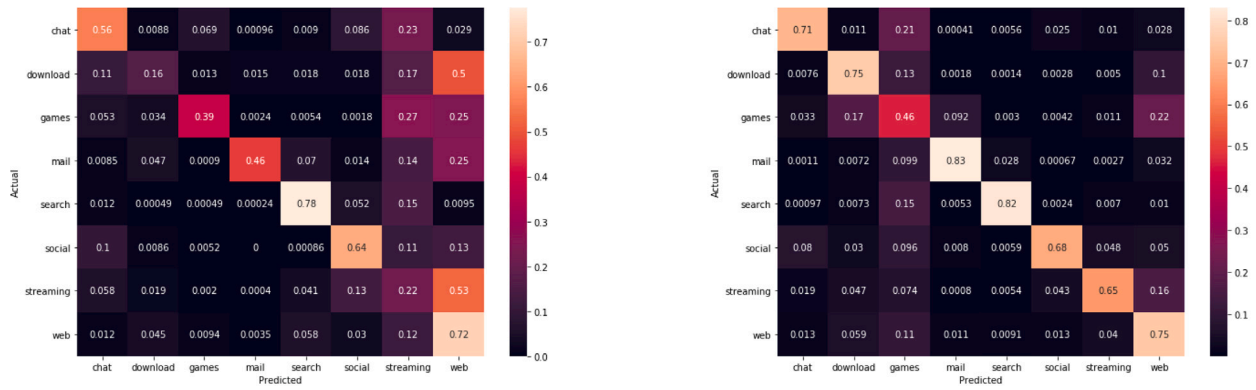



**Fig. 10.** Confusion matrices of UW-H in service-level classification when training and target datasets are two years (*left*) and two months (*right*) apart.

is when the datasets are the furthest apart and the effect of data drift is most noticeable on the overall performance of UW-H as well as UCDavis CNN, both in service-level and application-level classification. The second is when the datasets are only 2 months apart, *i.e.*, the model is trained on the 04-2021 dataset and used to classify the 06-2021 dataset. This is when data drift affects the performance of UCDavis CNN the least. Tables 7(a) and (b) present the baseline per-class accuracies, *i.e.*, when the target dataset is the same as the training dataset.

Fig. 10 depicts the confusion matrices of UW-H in service-level classification, in each of the above scenarios. When the training and test datasets are 2 years apart, *streaming* and *download* are the two services UW-H misclassifies the most, achieving 22% and 16% accuracy on these classes, respectively, and hence a drop of roughly 70% and 80% in classification accuracy. In particular, the model misclassifies 53% of the *streaming* flows and 50% of the *download* flows as *web* flows. We note that *web* is the class most of the misclassified flows are confused with, *e.g.*, 53% of the *streaming* flows, 50% of the *download* flows, 25% of the *games* flows, and 25% of the *mail* flows. *streaming* is the second most confused with, *e.g.*, 27% of the *games* flows, 23% of the *chat* flows, and 17% of the *download* flows. We can associate the model's tendency

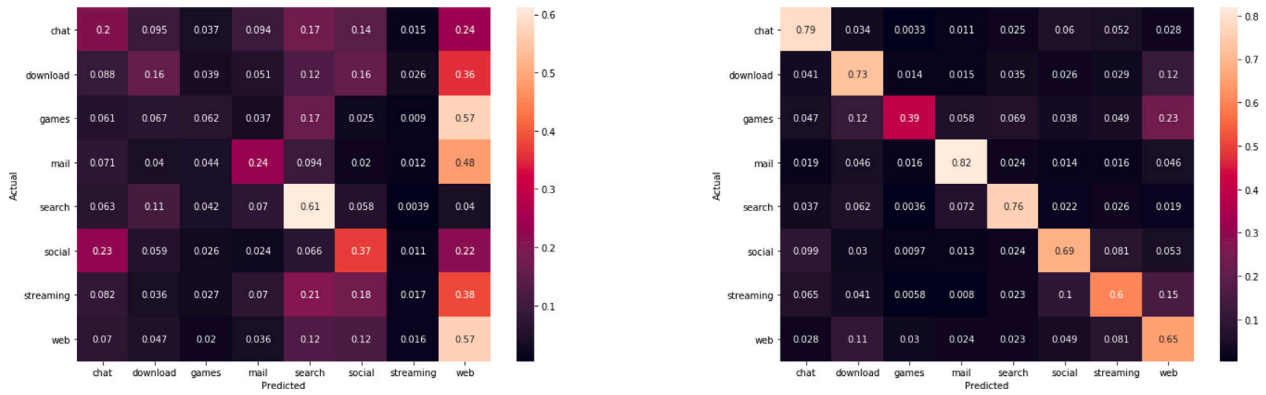



**Fig. 11.** Confusion matrices of the UCDavis CNN model in service-level classification when training and target datasets are two years (*left*) and two months (*right*) apart.

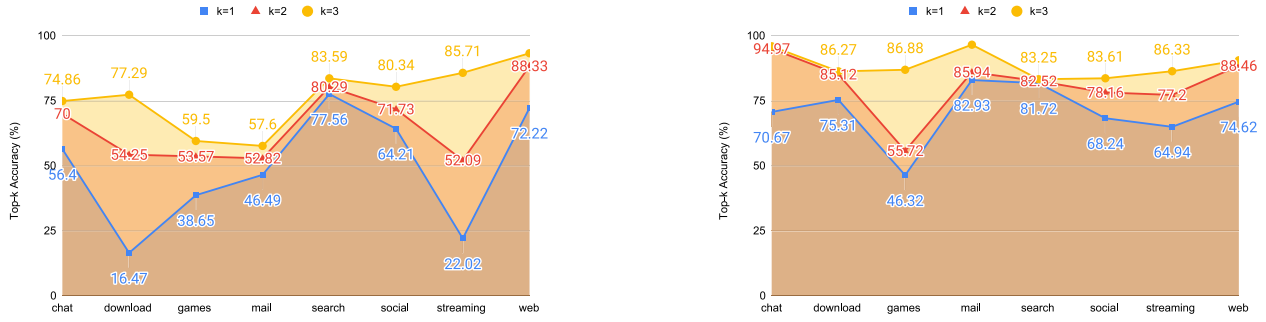



**Fig. 12.** Top-k accuracy for the UW-H model in service-level classification when training and target datasets are two years (*left*) and two months (*right*) apart.

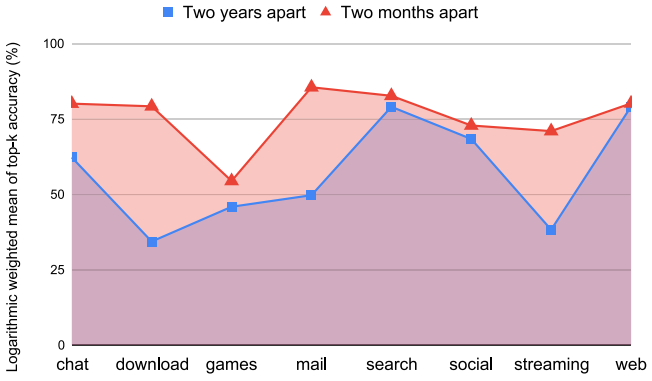



**Fig. 13.** Logarithmic weighted mean of top-k (k = 1, 2, 3) accuracy for the UW-H model with service-level classes.

to misclassify flows as *web* to the higher percentage of *web* flows in the training set which creates a bias for this class. Therefore, *web* is the *default* label that the classifier selects for a flow when its confidence in the true label is low.

When the training and target datasets are 2 months apart, however, we see a drastic increase in the model's ability to correctly classify all flows in general, in particular the *streaming* and *download* flows, *i.e.*, 65% and 75% accuracy respectively. Some but fewer flows remain misclassified as *web* flows, *e.g.*, 22% of the *games* flows, 16% of the *streaming* flows, and 10% of the *download* flows, compared to 25%, 53%, and 50% previously. However, the model seems also to have a bias for the *games* class, as it now confuses more flows with *games* flows, *e.g.*, 21% from the *chat* class, 15% from *search*, and 13% from *download*.

Fig. 11 presents the confusion matrices of the UCDavis CNN model. Evidently, when the training and target datasets are two years apart, the UCDavis CNN model has a much higher tendency to misclassify flows than UW-H. While *streaming* and *games* are the two classes with highest misclassification rates, *i.e.*, the accuracy of the classifier does not exceed 1.7% and 6.2% respectively on these classes and thus experiences a drop of over 90% in per-class accuracy, the UCDavis CNN also misclassifies over 75% of the *chat*, *download*, and *mail* flows, confusing these with *web* traffic most of the time. Similar to UW-H but at a larger extent, *web* is the class UCDavis CNN most confuses other classes with. UCDavis CNN also seems to be confused about more classes than UW-H. For instance, in addition to the *web*, UCDavis CNN equally misclassifies flows as *search* or *social*, *e.g.*, 17% of *chat* flows are misclassified as *search* and 14% as *social*, 12% of *download* flows are misclassified as *search* and 16% as *social*, 21% of *streaming* flows are misclassified as *search* and 18% as *social*, and 12% of *web* flows are misclassified as *search* and 12% as *social*. We attribute this higher misclassification and confusion rate of the UCDavis CNN model to the noisy features (*i.e.*, encrypted data) as input to the model, as discussed earlier. We note that, the misclassification and confusion rates drop significantly when the datasets are 2 months apart, and UCDavis CNN exhibits similar behavior to UW-H.

In the following, we attempt to study the extent to which classes are affected by drift in traffic data, leveraging the top-k accuracy measure. Some traffic classes may be impacted by data drift more than others such that it would take the classifier several more guesses to, eventually, correctly classify them.

Fig. 12, presents the top-k accuracy of UW-H when the target and training datasets are 2 years and 2 months apart, for k = 1, 2 and 3. Evidently, considering the model's top-2 or top-3 guesses significantly increases the model's accuracy on particular classes, thus increasing the model's overall accuracy. For example, when the target and training datasets are 2 years apart, the accuracy of the model on the *download* class increases from 16.5% to 54.2% and 77.3% with k = 2 and 3, respectively. On the contrary, the accuracy of the model on *mail* does not increase much when k is increased to 2 or 3, *i.e.*, after the second guess UW-H still misclassifies 47% of *mail* flows and 42% after 3 guesses. This suggests that *mail* traffic data has shifted so much since

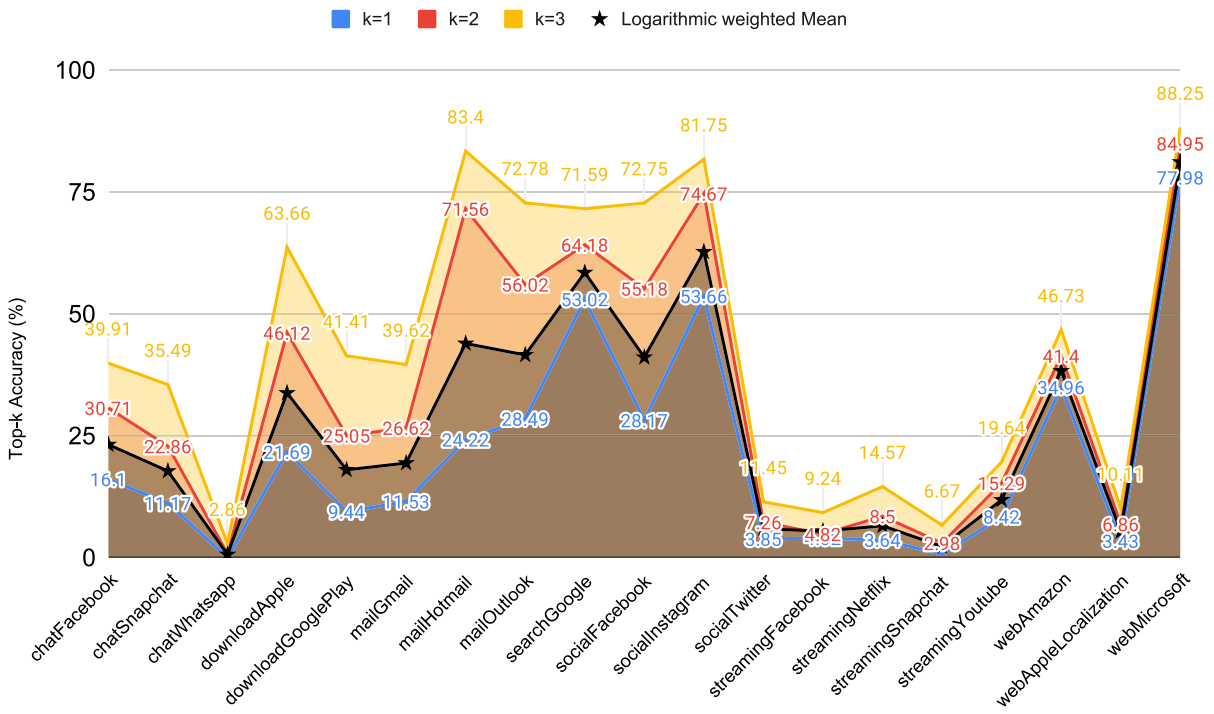



**Fig. 14.** Per-application class top-k accuracy of the UW-H model when training and target datasets are two years apart.

07-2019 (recall that the baseline accuracy of UW-H in classifying 07-2018 *mail* traffic is 97%) that it would take more than 3 guesses for the model to correctly classify 06-2021 *mail* flows. Indeed, we can see that the *download* and *mail* traffic drifted relatively less in the span of 2 months, as the model achieves relatively higher classification accuracy on these two classes, which also increases with k = 2 or 3. These findings are also inline with conclusions we drew earlier from the confusion matrices.

In Fig. 13, we summarize the top-k accuracy plots by averaging the top-k accuracies (for k = 1, 2, and 3) using a logarithmic weighted mean function. Logarithmically decreasing weights are applied to the top-k accuracies with increasing k, giving higher weights to top-1 accuracies. The goal is to study and compare, in a simplified way, the extent of drift in traffic data across service classes after 2 years versus 2 months.

We notice that *download* and *streaming* are the classes with the lowest average top-k accuracy when the training and target datasets are 2 years apart. Not only are they the most impacted by data drift but also this is where we notice the most obvious correlation between data drift and time span between the training and target datasets. The *games* class is also highly affected by the data drift due to the 2-year-time span between the training dataset and the 06-2021 dataset. However, interestingly, it is equally highly impacted by the 2-month-time span, experiencing roughly 50% drop in classification accuracy in both scenarios. On the contrary, *search*, *social*, and *web* are equally much less affected by data drift regardless of the time span between the training and target datasets, the average top-k accuracies being equally relatively high in both scenarios. Interestingly, the *games* class is the most diverse among all application classes. Several different SNIs are matched to the *games* class, and the class does not seem to be dominated by some major games. Thus, for the drop in accuracy to be this noticeable, it seems like online games and underlying protocols are in constant shift.

While we uncovered which traffic classes are most impacted by the drift in traffic data at the service level, it is worth investigating which applications within service classes are most susceptible to data drift. We conduct the 2-year-time span experiment at the application level, and measure the impact of data drift across application classes using the logarithmic weighted mean top-k metric, as reported in Fig. 14.

Fig. 14 depicts interesting findings. For instance, in the *chat* service class, the highest drop in accuracy is experienced by the Whatsapp application. In fact, the classifier fails to correctly classify all 06-2021 Whatsapp flows. Plus top-2 and top-3 classifications fail to boost the accuracy of the classifier on this particular class. In the *download* service class, we can see that *GooglePlay* is more affected by data drift with a lower logarithmic weighted mean top-k accuracy compared to the *Apple* applications. In the *mail* service class *Gmail* is drastically affected by data drift, much more than the other mailing applications. More interestingly, in the *social* class, the biggest impact is experienced by the *Twitter* application which makes *Facebook* and *Instagram* traffic seem more stable. Furthermore, almost all applications in the *streaming* service class experience roughly the same drop in accuracy and are affected almost equally by the data drift. Finally, for the *web* service class, the most stable traffic seem to belong to the *Microsoft* applications and the most affected traffic by the data drift is the one for the *AppleLocalization* application. The applications that are more affected by data drift (*e.g.*, Whatsapp, Twitter, *etc*..) seem to be the most popular ones within their respective service classes. For instance, *Whatsapp* had more daily active users in France compared to *Facebook* according to July 2021 statistics [34].

### 4.3. 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 difference in time of capture between datasets, indicates that the data distributions 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 L5 protocol distribution in the datasets, primarily looking for any time-related changes that we could identify. Note that since the data is encrypted, only some application-layer protocols are identifiable. Furthermore, as we filter on the application-layer protocols, we only perform the investigation for service-level classification.

Table 8 shows the adoption of HTTP/2 and SPDY protocols [35]. From 2018 to 2021, the adoption of HTTP/2 increased while the usage of SPDY, which is the predecessor to HTTP/2, drastically decreased.

**Table 8**
Adoption of HTTP/2 and SPDY protocols over time.

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

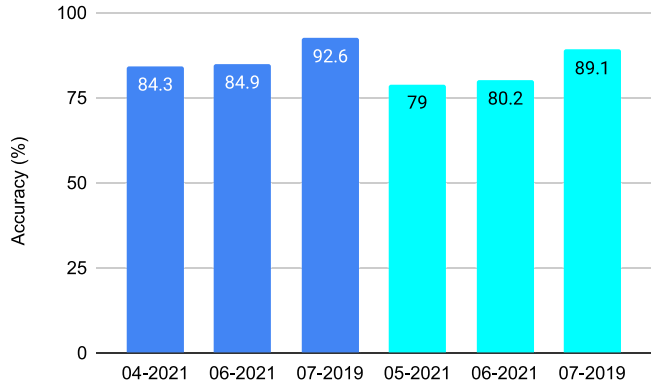



**Fig. 15.** UW-H performance in service-level classification on datasets with similar sizes (dark blue: 04-2021 size, light blue: 05-2021 size).

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. Indeed, SPDY leads to more information leakage than HTTP/2 and is easier to classify. Moreover, we expect to see a change in the accuracy of UW-H model even when it is trained and tested on the newest datasets. We hypothesize better results on the 07-2019 datasets where there could be considerably more SPDY flows than the 2021 datasets.

We know that for DL models the dataset size has a direct impact on the overall classification accuracy. Therefore, in order to have a fair comparison, we reduced the number of flows (*i.e.*, both in training and testing portions) 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 shown in Fig. 15. As can be seen, the accuracy of the model on the reduced 07-2019 datasets is still 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 the Application-Layer Protocol Negotiation (ALPN) header field of the TLS protocol. Table 9 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 larger datasets; (ii) 2021 datasets are captured closer in time, which makes their data patterns more similar as evident in Fig. 9.

From Table 9, it is apparent 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 Missing ALPN,

**Table 9**
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 |

**Table 10**
UW-H performance in service-level classification 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 |

*i.e.*, it may use HTTP/1, HTTP/2, or neither HTTP/1 nor HTTP/2. Table 10 illustrates the performance of UW-H 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, UW-H 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. Recall from Table 8 that 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 8 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 then reduce the number of HTTP/1, HTTP/2, and Missing ALPN flows in each dataset (*i.e.*, both in training and testing portions) to the smallest across all the datasets (*i.e.*, the number of HTTP/2 flows in the 05-2021 dataset). The results are shown in Fig. 16. It is evident that UW-H 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. All of these results are inline with the increase in the adoption of HTTP/2 and decrease in SPDY usage over time in Table 8. This further supports 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 ALPN) are from the Missing ALPN 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.

We then investigate whether the model is biased on the ALPN field. 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 UW-H model with the obfuscated ALPN field on HTTP/1 and HTTP/2 flows. Note that we leverage datasets with similar sizes as before and present average accuracy across multiple experiments. As shown in Fig. 17,

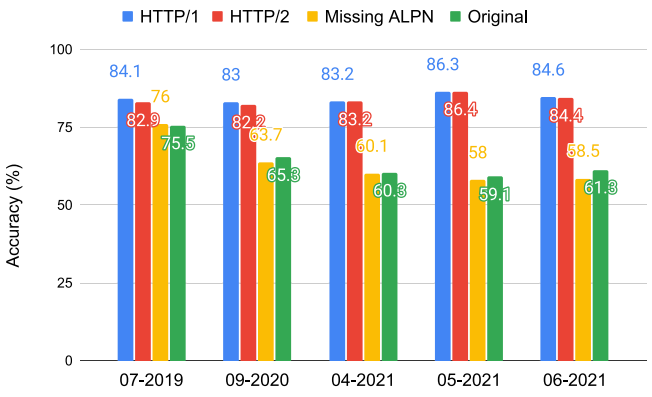



**Fig. 16.** UW-H performance in service-level classification with ALPN filter on similar size datasets.

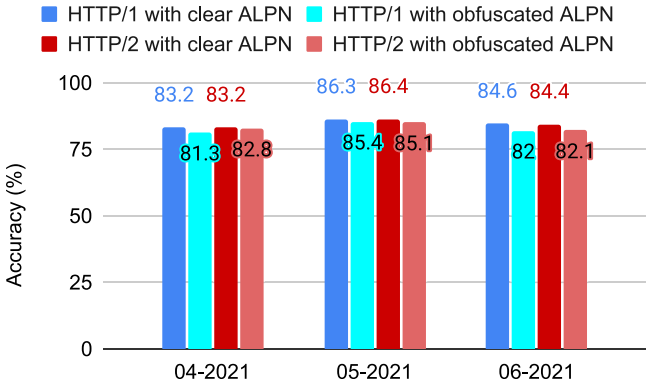



**Fig. 17.** UW-H performance in service-level classification with ALPN obfuscation.

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%. Hence, 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 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 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. 16 support this claim with similar model performance for HTTP/1 and HTTP/2 Web protocols. 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 UW-H 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 11 illustrates the accuracy of the UW-H 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 dataset. 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 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.

**Table 11**
UW-H performance in service-level classification on datasets merged based on the ALPN filter.

| ALPN | HTTP/1 or HTTP/2 | Missing |
|---|---|---|
| Accuracy (%) | 95.2 | 83.0 |

## 5. Architecture adaptation

In this section, we examine the performance of the UW 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.

### 5.1. Ensuring model convergence

We start by training and testing the UW model and the decomposed models on datasets from 2021. Again, we discard the UW-A model due to its negligible performance, as discussed in Section 4.1. The results of these experiments for service-level classification are shown in Table 13.

Although suffering a drop from the baseline 2019 dataset, the accuracy of the UW 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 UW-F, showing a mere accuracy of 11% (*i.e.*, worse than a random classifier). On the other hand, the UW-H performs reasonably on the same dataset.

Before we delve into the reasons for the under performance of UW-F, we note that the lower performance of the model on 2021 datasets compared to 07-2019 dataset can be attributed to dataset size. Recall that the 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.

The results for the same experiment in application-level classification are summarized in Table 14. The overall classification results are better than for service-level classification, which is inline with the previous experiments (*i.e.*, on the 2019 dataset). Furthermore, the same trends as service-level classification more or less hold with application-level classification. UW-H model performs relative to the dataset sizes, *i.e.*, it achieves the highest and lowest accuracies on the largest and smallest datasets, respectively. Moreover, the trends for UW-F are similar as for service-level classification but with more promising results. With application-level classification, there is no performance peculiarity for UW-F on the 04-2021 dataset. Nevertheless, UW-F performs the worst (*i.e.*, 81.2% accuracy) on the 04-2021 dataset, albiet much better than the accuracy in service-level classification (*i.e.*, 11%).

To troubleshoot the UW-F model performance 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

**Table 12**
Summary of key findings in the investigations.

| Finding | Figures & Tables |
|---|---|
| The UW model performance is more aligned with the UW-H part of the model, while the UW-A does not contribute much to the performance of UW. | Tables 4, 5 |
| The UW model performance drops when it is trained on the original dataset (*i.e.*, the dataset it was designed for) and evaluated on the newer datasets. Performance drop is associated more with the UW-H model compared to UW-F model. | Fig. 8, Table 6 |
| The performance drop has a direct relationship to the collection time difference between training and target datasets, highlighting the impact of data drift. | Fig. 9 |
| Model architecture and feature engineering plays an important role in the amount of performance drop due to data drift. The UW model is more robustto data drift than UCDavis CNN. | Fig. 9 |
| Different service classes have varying susceptibility to data drift. The model architecture does not change the overall order of susceptibility of service classes, but rather impacts the measure of confusion between service classes. | Figs. 10, 11, Table 7 |
| Download, streaming, mail, and games service classes are more impacted by data drift over the course of two years. However, the games class is more susceptible to data drift, as in the course of two months the performance drop is still significant. | Figs. 12, 13 |
| Applications per service class experience different levels of performance drop due to data drift compared to one another, *e.g.*, Whatsapp experiences more drop in performance compared to other chat applications. | Fig. 14 |
| The TLS headers are easier to classify on the original dataset compared to the newer datasets. The majority of this ease comes from the Missing ALPN portion of the original dataset. The Missing ALPN portion makes around 70% of each dataset. | Fig. 15, Tables 9, 10 |
| The ALPN value itself does not affect the information in the TLS headers. | Fig. 17 |
| The reasoning behind easier classification of Missing ALPN portion of original dataset is evidently the decline in the usage of SPDY and increase in the usage of HTTP/2. | Fig. 16, Tables 8, 11 |

**Table 13**
Model performance across the 2021 datasets in service-level classification.

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

**Table 14**
Model performance across the 2021 datasets in application-level classification.

| Model | Accuracy (%) | | |
|---|---|---|---|
| | 04-2021 | 05-2021 | 06-2021 |
| UW | 85.2 | 79.6 | 88.9 |
| UW-F | 81.2 | 84.2 | 86.4 |
| UW-H | 85.9 | 74.1 | 86.5 |

**Table 15**
UW-F adaptation best practices for service-level classification.

| 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 |

of the UW-F. The masking layer acts as a de-noising layer to filter out time-steps that do not have any information. Therefore, these time-steps can be skipped in the LSTM layer.

The above alterations boosted the accuracy of the UW-F model on the 04-2021 dataset from 11% to 88.3% in service-level classification. 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 downsides, evidently, in the case of the 04-2021 dataset, these alterations are necessary for the model to achieve reasonable performance in service-level classification.

### 5.2. 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.

#### 5.2.1. Dropout rate reduction

The stacked LSTM in the UW-F 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 15 for service-level classification. With the same adaptation, a performance boost is also noticeable in application-level classification for the 04-2021 and 05-2021 datasets, as depicted in Table 16.

#### 5.2.2. UW-F simplification

The stacked LSTM layer proposed in [2] is a complex UW-F 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 with less than 20K flows, as shown for 05-2021 dataset in Table 15. We further found that on datasets smaller than 10K flows, even reducing the stacked LSTM layer to a 1D Convolution (CONV1D) layer helps UW-F performance in achieving comparable or better results with a lower number of parameters (*i.e.*, a lighter and faster to train model), contrary to what was shown in [2] for large datasets.

The results for application-level classification with similar adaptations are shown in Table 16. For application-level classification the reduction of stacked LSTM layers to BLSTM results in a slightly lower model accuracy on the 05-2021 dataset (*i.e.*, 85.3%). However, since this dataset has smaller than 10K flows, we leverage CONV1D layers with best practices including masking layer addition and learning rate reduction. Evidently, with these adaptions, the UW-F model achieves the best performance in application-level classification, achieving an accuracy of 85.9%.

#### 5.2.3. Best practices

Table 17 summarizes our recommended best practices based on a given dataset's size. We suggest that when leveraging the UW model, UW-F should be adapted to the training dataset's size. When there are fewer than 50K training flows, reducing the dropout layer value (*e.g.*,

**Table 16**
UW-F adaptation best practices for application-level classification.

| Dataset | Adaptation | Training flows | Accuracy (%) |
|---|---|---|---|
| 04-2021 | Dropout + Learning rate | 21,040 | 88.6 |
| | Dropout + Learning rate + Masking layer | 21,040 | 90.2 |
| | BLSTM + Dropout + Learning rate + Masking layer | 21,040 | 90.1 |
| 05-2021 | Dropout | 8861 | 85.8 |
| | BLSTM + Dropout | 8861 | 85.3 |
| | CONV1D + Learning rate + Masking layer | 8861 | 85.9 |

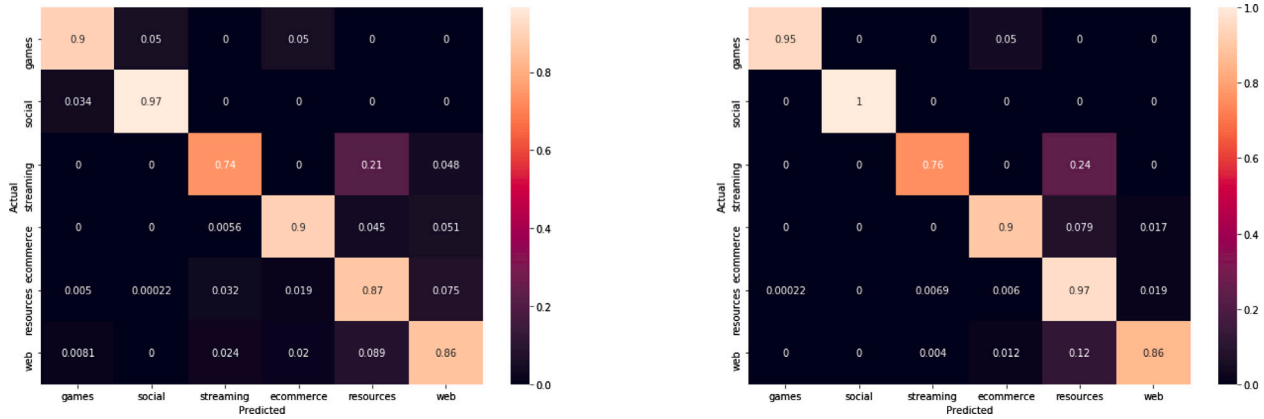



**Fig. 18.** Confusion matrices for UW-F (*left*) vs. its adapted version (*right*) on the QUIC-05-2021 dataset in service-level classification.

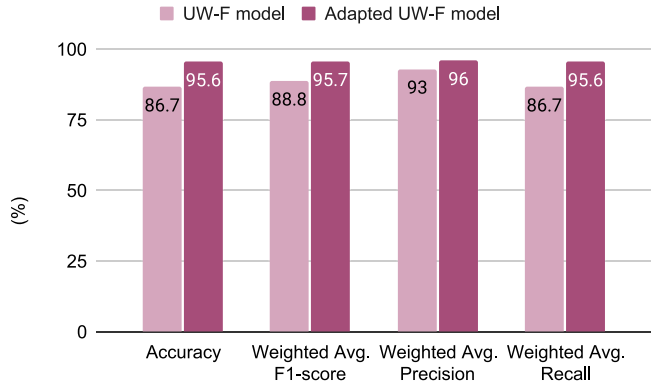



**Fig. 19.** UW-F performance with and without adaptations on the QUIC-05-2021 dataset in service-level classification.

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 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. As an example, since 04-2021 dataset has 21K training flows in application-level classification, we also tried the BLSTM architecture instead of stacked LSTM layers. This resulted in a similar accuracy of 90.1% which is only 0.1% lower than the accuracy of the more complex stacked LSTM architecture, as shown in Table 16. Finally, if the dataset has fewer than 10K flows, using simple 1D Convolutions (*i.e.*, shown in [2] appendix) is adequate and preferable over the LSTM layer.

### 5.3. QUIC results

We also evaluate the performance of the UW model on real-world QUIC data, before and after employing the adaptation guidelines proposed in the previous subsection. We show that these guidelines indeed improve model accuracy on a dataset consisting of QUIC flows, thus showing that our adaptation best practices generalize to encrypted protocols other than TLS.

**Table 17**
UW-F architecture adaptation rules.

| Number of flows | Adaptation |
|---|---|
| ≤50K | Dropout reduction |
| ≤20K | BLSTM |
| ≤10K | 1D Convolutions [2] |

UW-F was shown to achieve over 99% accuracy on a synthetic QUIC dataset [2]. However, on our real-world QUIC data, *i.e.*, QUIC-05-2021, the model achieved 86.7% and 83% accuracy in service-level and application-level classification, respectively. 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. The performance of UW-F before and after adaptions for service-level and application-level classification are shown in Figs. 19 and 21, respectively.

The model achieves an accuracy of 86.7% before adaptation, whereas the adapted model achieves 95.6% for service-level classification. Furthermore, the application-level classification accuracy is boosted from 83% to 91%. 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-F model by 3% to 9% in service-level classification and by 3.8% to 8% in application-level classification. The precision for both models is quite high, however, the main advantage of the adapted model is correctly predicting a larger portion of the flows for each class, which results in a 9% and 8% increase in recall for service-level and application-level classification, respectively. Figs. 18 and 20 show the confusion matrices of UW-F and its adapted version for service-level and application-level classification, respectively. The recall increase is visible in the confusion matrices, where the adapted model achieves a higher accuracy per class in service-level classification. Furthermore, for application-level classification the adapted UW-F model receives significantly higher accuracy across 75% of the application classes, especially for classes with the lowest accuracy without adaptation.

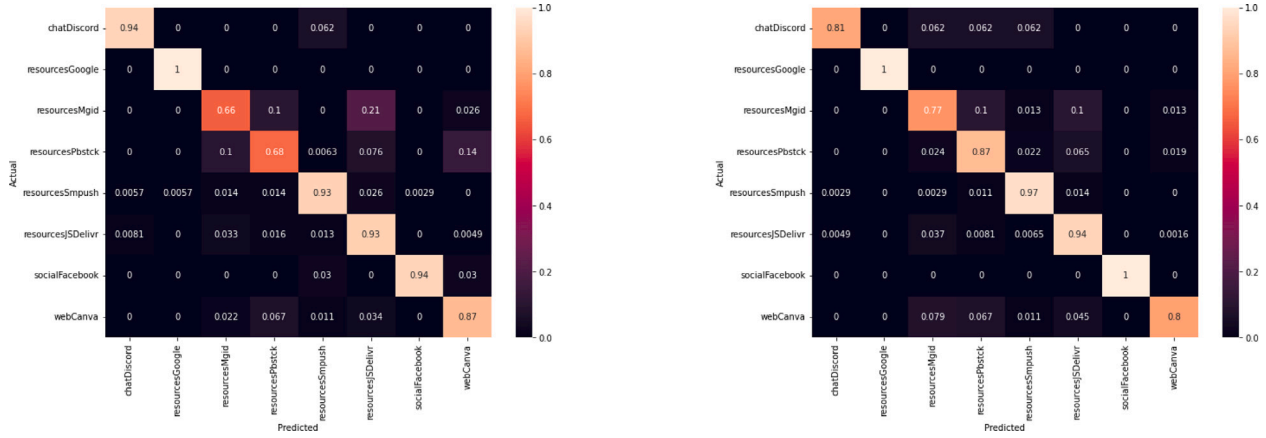



**Fig. 20.** Confusion matrices of the UW-F (*left*) vs. its adapted version (*right*) on the QUIC-05-2021 dataset with application level-classification.

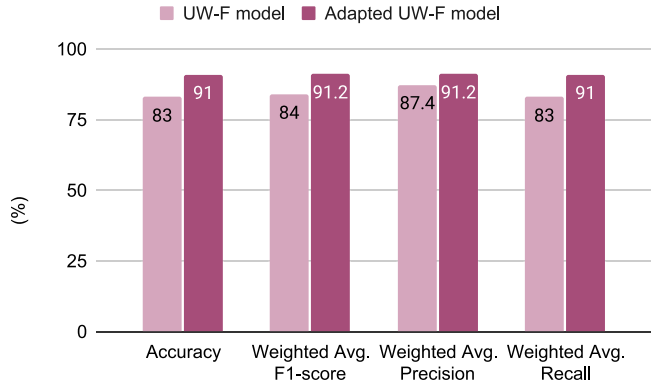



**Fig. 21.** UW-F performance with and without adaptations on the QUIC-05-2021 dataset in application-level classification.

Therefore, the adaptations allow the model to achieve a higher classification accuracy across classes. The most significant increase is for the *resources* class with 10% increase in accuracy for service-level classification. Similarly, the adaptation results in a 19% increase in accuracy for the *resourcePbstck* class in application-level classification.

## 6. 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. We showed that a model which operates on the traffic shape is more resilient to data drift than one that operates on TLS headers. Also, we examined the impact of model architecture and feature engineering on model robustness by comparing the two models over the same datasets.

We investigated how model architectures are affected by data drift using confusion matrices for both UW-H and UCDavis CNN models. Furthermore, we presented the top-k accuracy and its logarithmic weighted mean to measure the amount of confusion due to data drift amid service classes when training and test datasets are close (*i.e.*, 2 months) and far (*i.e.*, 2 years) apart. We also analyzed the contribution of each application towards performance drop over a long period among different service classes. 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 model is trained on different datasets with no adaptations. We performed an ablation study and examined the performance of decomposed models, as well as the effect of changing structural parameters, to propose best practices for designing an architecture that performs well on unseen and possibly newer datasets. We showed results for service-level and application-level classification to highlight generalizability of proposed adaptions at different levels of labeling granularity. We also showed the generalizability of our guidelines to different encryption protocols by evaluating the adapted architecture on a dataset of QUIC traffic for service-level and application-level classification, which resulted in up to 9% higher classification accuracy than the default model without adaptations.

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.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The authors do not have permission to share data.

## References

[1] S. Rezaei, B. Kroencke, X. Liu, Large-scale mobile app identification using deep learning, IEEE Access 8 (2020) 348–362.

[2] I. Akbari, M.A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, S. Tuffin, A look behind the curtain: Traffic classification in an increasingly encrypted web, Proc. ACM Meas. Anal. Comput. Syst. 5 (1) (2021).

[3] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, M. Saberian, Deep packet: A novel approach for encrypted traffic classification using deep learning, Soft Comput. 24 (3) (2020) 1999–2012.

[4] I. Akbari, M.A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, S. Tuffin, Traffic classification in an increasingly encrypted web, Commun. ACM 65 (10) (2022) 75–83.

[5] G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapé, Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges, IEEE Trans. Netw. Serv. Manag. 16 (2) (2019) 445–458.

[6] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246 (Proposed Standard), Internet Engineering Task Force, 2008, updated by RFCs 5746, 5878, 6176. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt.

[7] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, Z. Shi, The QUIC transport protocol: Design and internet-scale deployment, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 183–196, [Online]. Available: https://doi.org/10.1145/3098822.3098842.

[8] N. Malekghaini, E. Akbari, M.A. Salahuddin, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, S. Tuffin, Data drift in DL: Lessons learned from encrypted traffic classification, in: 2022 IFIP Networking Conference (IFIP Networking), 2022, pp. 1–9, http://dx.doi.org/10.23919/IFIPNetworking55013.2022.9829791.

[9] Y. Kumano, S. Ata, N. Nakamura, Y. Nakahira, 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.

[10] R. Alshammari, A.N. Zincir-Heywood, Can encrypted traffic be identified without port numbers, IP addresses and payload inspection? Comput. Netw. 55 (6) (2011) 1326–1350.

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

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

[13] C. Bacquet, A.N. Zincir-Heywood, 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.

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

[15] P. Velan, M. Čermák, P. Čeleda, M. Drašar, A survey of methods for encrypted traffic classification and analysis, Int. J. Netw. Manag. 25 (5) (2015) 355–374.

[16] R. Boutaba, M.A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, O.M. Caicedo, A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, J. Internet Serv. Appl. 9 (1) (2018) 1–99.

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

[18] A. Khajehpour, F. Zandi, N. Malekghaini, M. Hemmatyar, N. Omidvar, M.J. Siavoshani, Deep inside tor: Exploring website fingerprinting attacks on tor traffic in realistic settings, in: 2022 12th International Conference on Computer and Knowledge Engineering, ICCKE, 2022, pp. 148–156, http://dx.doi.org/10.1109/ICCKE57176.2022.9960104.

[19] A. Shiravi, H. Shiravi, M. Tavallaee, A.A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (3) (2012) 357–374.

[20] M. Abbasi, A. Shahraki, A. Taherkordi, Deep learning for network traffic monitoring and analysis (NTMA): a survey, Comput. Commun. 170 (2021) 19–41.

[21] D. Herrmann, R. Wendolsky, H. Federrath, Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier, in: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, 2009, pp. 31–42.

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

[23] M. Juarez, S. Afroz, G. Acar, C. Diaz, R. Greenstadt, A critical evaluation of website fingerprinting attacks, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 263–274.

[24] T. Wang, I. Goldberg, On realistically attacking tor with website fingerprinting., Proc. Priv. Enhanc. Technol. 2016 (4) (2016) 21–36.

[25] P. Sirinam, N. Mathews, M.S. Rahman, M. Wright, Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1131–1148.

[26] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, L. Cavallaro, Insomnia: Towards concept-drift robustness in network intrusion detection, in: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, 2021, pp. 111–122.

[27] G. Engelen, V. Rimmer, W. Joosen, Troubleshooting an intrusion detection dataset: the CICIDS2017 case study, in: 2021 IEEE Security and Privacy Workshops, SPW, IEEE, 2021, pp. 7–12.

[28] M. Ma, S. Zhang, D. Pei, X. Huang, 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.

[29] S. Saurav, P. Malhotra, V. TV, N. Gugulothu, L. Vig, P. Agarwal, 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.

[30] V.F. Taylor, R. Spolaor, M. Conti, I. Martinovic, Robust smartphone app identification via encrypted network traffic analysis, IEEE Trans. Inf. Forensics Secur. 13 (1) (2017) 63–78.

[31] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/.

[32] F. Chollet, et al., Keras, 2015, https://keras.io.

[33] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache spark: A unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65, [Online]. Available: https://doi.org/10.1145/2934664.

[34] M. Mehner, Whatsapp, WeChat and meta messenger apps - global usage of messaging apps, penetration and statistics, 2022, Accessed Oct. 14, 2022. [Online]. Available: https://www.messengerpeople.com/global-messenger-usage-statistics/#France.

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

**Navid Malekghaini** received his B.Sc. from Sharif University in computer engineering. He is currently a graduate student at the University of Waterloo. His research mostly revolves around the intersection of machine learning, network security, network management, and cloud computing with a focus on automation and scalability. He is also a reviewer for peer-reviewed journals and conferences.

**Elham Akbari** received her B.Sc. and M.Sc. degree from Sharif University of Technology. She is currently a Ph.D. student at the University of Waterloo. Her work concerns network traffic classification and the application of few-shot and meta-learning algorithms in encrypted network data processing.

**Mohammad A. Salahuddin** received the M.Sc. and Ph.D. degrees in computer science from Western Michigan University in 2003 and 2014, respectively. He is currently a Research Assistant Professor of Computer Science with the University of Waterloo. His research interests include the Internet of Things, content delivery networks, network softwarization, network security, and cognitive network management. His co-authored publications have received numerous awards, including ACM SIGMETRICS best student paper, IEEE/IFIP NOMS best paper, and IEEE CNOM best paper. He serves on the TPC for international conferences and is a reviewer for various peer-reviewed journals, magazines and conferences.

**Noura Limam** is a research assistant professor of computer science at the University of Waterloo, Canada. She received the MSc and PhD degrees in computer science from the University Pierre & Marie Curie (now Sorbonne University), France, in 2002 and 2007, respectively. She is an active researcher and contributor in the area of network and service management. Her current interests revolve around network automation and cognitive network management. She is the chair of the IEEE ComSoc Network Operations and Management Technical Committee, an Associate Editor of the IEEE Communications Magazine, and a Guest Editor of the IEEE Communications Magazine Network Softwarization and Management Series. She served and continues to serve on the Technical Program Committee as well as the Steering Committee of several conferences.

**Raouf Boutaba** received his M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Chair Professor and the director of the School of Computer Science at the University of Waterloo. He is the founding Editor-in-Chief of IEEE Transactions on Network and Service Management (2007-2010) and served as the Editor-in-Chief of the IEEE Journal on Selected Areas in Communications (2018-2021). He is a Fellow of the IEEE, the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.

**Bertrand Mathieu** received the M.Sc. degree from the University of Marseille, the Ph.D. degree from the Sorbonne University, Paris, and the Habilitation à Diriger des Recherches degree from the University of Rennes. He is a Senior Researcher with Orange Innovation. He contributed to 14 national/European projects and published more than 80 papers in international conferences, journals, or books. He is working on programmable networks, QoS and QoE and new network solutions. He is a member of several conferences' TPC.

**Stéphanie Moteau** is an R&D engineer at Orange Labs since 18 years recognized as an Orange Expert on Future Networks. She works on the IP metrology domain for various purposes: security, quality of service and quality of experience, Content Delivery Network, traffic classification, etc.

She participated in many collaborative projects (French and European ones) as contributor and project leader for Orange. Two years ago, she became Data Scientist. This gives her new methods to perform her research activity.

**Stéphane Tuffin** is an R&D engineer at Orange Labs since 20 years recognized as an Orange Expert on Future Networks. He has been deeply involved in the transition from circuit-switched voice to IP and spent several years in preparing and supporting IMS deployments in Orange countries.

In 2013, he took the lead of a research project in the field of Web real-time communications which gave birth to a "no backend" offer.

Since 2017, he is leading a research program on end-to-end network quality which covers network planning, traffic classification, monitoring-troubleshooting the performance of Internet applications and adapting networks to the need of interactive services. His own research interests are the monitoring of encrypted traffic and low-latency networking. These last years, he is refocusing research efforts on the integration of environmental limits in network management.