Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Large-scale Mobile App Identification Using Deep Learning

SHAHBAZ REZAEI[1], (Member, IEEE), BRYCE KROENCKE[2], AND XIN LIU.[3], (Fellow, IEEE)
[1]Computer Science Department, University of California, Davis, CA, USA (e-mail: srezaei@ucdavis.edu)
[2]Computer Science Department, University of California, Davis, CA, USA (e-mail: bakroencke@ucdavis.edu)
[3]Computer Science Department, University of California, Davis, CA, USA (e-mail: xinliu@ucdavis.edu)

Corresponding author: Shahbaz Rezaei (e-mail: srezaei@ucdavis.edu).

ABSTRACT Many network services and tools (e.g. network monitors, malware-detection systems, routing and billing policy enforcement modules in ISPs) depend on identifying the type of traffic that passes through the network. With the widespread use of mobile devices, the vast diversity of mobile apps, and the massive adoption of encryption protocols (such as TLS), large-scale traffic classification becomes inevitable and more difficult. In this paper, we propose a deep learning model for mobile app identification. The proposed model only needs the payload of the first few packets for classification, and, hence, it is suitable even for applications that rely on early prediction, such as routing and QoS provisioning. The deep model achieves between 84% to 98% accuracy for the identification of 80 popular apps. We also perform occlusion analysis for the first time to bring insight into what data is leaked from SSL/TLS protocol that allows accurate app identification. Moreover, our traffic analysis shows that many apps generate not only app-specific traffic, but also numerous ambiguous flows. Ambiguous flows are flows generated by common functionality modules, such as advertisement and traffic analytics. Because such flows are common among many different apps, identifying the source app that generates ambiguous flows is challenging. To address this challenge, we propose a CNN+LSTM model that takes adjacent flows to learn the order and pattern of multiple flows, to better identify the app that generates them. We show that such flow association considerably improves the accuracy, particularly for ambiguous flows. Furthermore, we show that our approach is robust to mixed traffic scenarios where some unrelated flows may appear in adjacent flows. To the best of our knowledge, this is the first work that identifies the source app for ambiguous flows.

INDEX TERMS convolutional neural network, deep learning, flow association, mobile app identification, occlusion analysis, recurrent neural network, smartphone app fingerprinting, traffic classification

I. INTRODUCTION

NETWORK traffic classification assigns labels to network traffic flows. Depending on its purpose, the labels can be various apps (e.g. Facebook, YouTube, and Skype), different traffic types (e.g. streaming, downloading, and browsing), user actions (e.g. liking or sharing), etc. Network traffic classification has been widely used for applying routing or billing policy, enforcing QoS requirements, identifying security threats or anomalies in firewalls or malware detection systems, etc. Because of its wide applications, traffic classification has been studied for more than two decades.

The earliest attempts to classify network traffic relied mostly on port numbers in TCP/UDP packets. Such approaches, however, are no longer sufficient because modern apps and protocols are often not tightly bound to a specific unique port number. The next set of approaches has focused on manually identifying patterns and keywords in unencrypted parts of the payload, called deep packet inspection (DPI). Nevertheless, such approaches are time-consuming and labor-intensive since they require continual manual extraction of signatures [1]. Furthermore, their applicability has degraded as more traffic is encrypted.

To address the limitations of such methods, classical machine learning approaches have emerged. Classical machine learning approaches, such as support vector machine (SVM), K-nearest neighbor (KNN), etc., have achieved the state-of-the-art accuracy for many years. However, these approaches mostly relied on statistical features obtained from the entire flow which are not suitable for early classification, where the result of the classification is needed after observing the beginning of a flow. Furthermore, the rapid growing usage of mobile devices has dramatically changed the traffic patterns

of Internet. Mobile app identification poses a new challenge, namely the existence of a large number of applications. Due to the inability to capture more complex patterns for a large number of classes, the accuracy of traditional approaches has degraded.

In recent years, deep learning has been widely adopted in various application scenarios from image recognition to text translation. A few recent studies show that deep learning models can achieve high accuracy for network traffic classification [2]–[4] and they do not require manual feature selection. However, these studies are limited to a small number of services or traffic types and their accuracy has yet to be investigated on large-scale mobile app identification.

In this paper, we focus on two main challenges in mobile app identification. First, we aim to solve the app identification problem for a large number of classes, generally known as extreme classification in machine learning. Although a few studies have already achieved acceptable accuracy in extreme classification, we perform an occlusion analysis to better understand the reason why deep models work. Second, we aim to identify source apps for *ambiguous* traffic flows. Our analysis of the traffic shows that many apps generate not only app-specific flows, but also numerous ambiguous flows. Ambiguous flows are generated by embedded modules that are common among many different apps, such as advertisement, or common Web-API traffic flows [5]. Identifying ambiguous traffic (i.e. labeling them as *ambiguous*) is shown to be possible [5]. However, identifying the source app that generates ambiguous traffic flows has not been studied before. The problem is challenging because ambiguous traffic can be generated by multiple apps and patterns in such traffic might be similar even when it is generated by different apps.

To tackle these challenges, we propose two deep learning models. We use a convolutional neural network (CNN) model for the identification of large scale mobile apps. Our dataset is comprised of 80 apps from a wide range of categories, including streaming, messaging, news, navigation, etc. We show that the proposed deep learning model achieves high accuracy across many network and encryption protocols for app identification by taking payload information of the first few packets.

To solve the second challenge of identifying the source app of an ambiguous flow, we develop a joint CNN and LSTM architecture building upon the developed CNN model. The model takes a few consecutive flows as input to identify the app that generates the first flow. The insight is that although one ambiguous traffic flow may not present any unique pattern, a set of consecutive flows that are generated by a single app presents unique patterns. The CNN part of the model captures the patterns in the header and payload of packets in each flow, while the LSTM part captures sequential patterns in adjacent flows. We show that our model is robust even if some flows belonging to other apps are taken into the model along with the app that we are interested in identifying.

In summary, the contribution of this paper is as follows:

- We show that a CNN model is capable of identifying mobile apps for 80-class classification with high accuracy. We perform fine-grained occlusion analysis on our proposed CNN model for the first time for traffic classification task. Our results bring insight into why a deep learning model can classify SSL and transport layer security (TLS) traffic flows. We show that certain handshake fields of SSL/TLS protocol leak enough information for app identification.
- Building upon the first CNN model, we propose a joint CNN+LSTM model to identifying the source app of ambiguous traffic flows as well as regular app-related traffic flows. Unlike previous studies where ambiguous traffic flows were removed or only detected as ambiguous, our joint model takes adjacent flows to identify the source app of the first flow and achieve high accuracy. To the best of our knowledge, this is the first paper that takes the relation of several flows into account for identifying the source apps of ambiguous flows.

The remainder of this paper is organized as follows. Section II covers the most related work in literature about general traffic classification and smartphone app identification approaches. In Section III, we explain the detailed system architecture, including some definitions, input features, and deep model architectures. Section IV contains our dataset description and evaluation of our models on real data. In Section V, we conclude the paper.

II. RELATED WORK

Although general traffic classification seems to be similar to mobile app identification, there are some differences in the types of traffic and patterns generated by smartphones that make them distinct from traditional traffic classification or website fingerprinting [5], [6]. As a result, the studies in literature are divided into two groups: Some studies focused on general traffic classification and the others focused on mobile app identification.

A. GENERAL TRAFFIC CLASSIFICATION

Classical machine learning approaches have been widely used in the past for network traffic classification [7]. Both supervised methods, including C4.5 [8], [9], K-nearest neighbor [10], [11], SVM [12]–[14], naive Bayes [15], [16] and unsupervised methods, such as Gaussian mixture model [17] and K-means [18], [19] have been used in the past. These methods often rely on statistical features of the entire flow, such as average packet length, standard deviation of inter-arrival time of packet, etc., for classification which is not suitable for *early prediction*. Early prediction refers to scenarios where prediction is required as soon as traffic is observed by using the beginning of the flow. Routing, QoS provisioning, and malware-detection are just a few applications of traffic classification where early prediction is necessary. Moreover, the accuracy of such classical approaches have declined because of their simplicity, manual feature selection, and incapability of learning complex patterns [20].

Recently, deep learning models have shown tremendous success in a wide range of applications from image recognition to text translation and speech recognition. The network community has also started applying deep learning models to tackle the network traffic classification task [21]. Statistical features of flows were converted into 2-dimensional arrays and used as input of a CNN model in [22]. They successfully trained a model for a small number of classes for traffic type classification. However, it cannot be used for early prediction as the input is built based on the entire flow. In [23], the authors used a combination of classical machine learning and deep learning for classification of five Google services that use QUIC protocol. They proposed a two step classification procedure: first, they used statistical features with random forest to classify three classes (that is, voice, chat, others), and then they used payload data with a CNN model for classifying other classes. This method is also not suitable for early prediction. A combination of CNN and stacked auto-encoders (SAEs) were used in [3] to classify traffic types and applications of ISCX dataset [24]. Their approach achieved over 90% accuracy for the classification of 17 apps. However, these studies focused on a handful of classes and treated deep learning models as black-box without investigating the features their models rely on.

In [25], the authors investigated several combinations of deep models, including CNNs and long short-term memory (LSTMs), for classification of 15 services, including Office365 and Google. They also studied the combination of header and time-series features as input and their best model achieved about 96% accuracy. In [26], a CNN model was used to classify five protocols and five applications, separately. They converted time-series features into a 2-dimensional image by using reproducing kernel Hilbert space (RKHS). They achieved 99% accuracy for 5-class classification. Interestingly, in their paper, SVM and decision tree also achieved over 97% accuracy.

In [20], a general guideline for applying deep learning models for the traffic classification was provided along with a general framework for capturing data, selecting a suitable deep model, and extracting proper input features. Although most traffic classification studies fall under the general framework, there is no concrete study to show that the framework works for large-scale classification problems with numerous classes.

In [2], the authors used a transfer learning approach to obviate the need for large labeled datasets. First, they trained a CNN model that takes the sampled time-series features of a flow and predicts several statistical properties of the flow, such as average packet length and average inter-arrival time. Then the model weights were transferred to a new model and the new model was trained on a small labeled dataset. The paper also showed that classification based on sampled time-series is feasible. In [4], the authors used a CNN model in multi-task learning framework to obviate the need for a large training dataset. They used a combination of a large unlabeled dataset and a small labeled dataset to train a model that predicts 3 tasks: application, bandwidth, and duration of flows. Both approaches were only studied on small-scale problems, with as few as 5 classes. None of these studies investigated large-scale classification. Moreover, transfer learning is shown to have inherent security vulnerabilities [27].

Instead of classifying various traffic types with different underlying protocols, a few studies focused only on traffic of SSL/TLS protocols. In [28], the authors defined a new set of statistical features and used C4.5 and random forest to identify 9 services. In [29], [30], the authors used header information in SSL/TLS packets to create a fingerprint corresponding to a first-order [29] or second-order [30] homogeneous Markov chain. They achieved high accuracy on a dataset of 12 web services. In [31], the authors extended the previous work by adding a weighted ensemble classifier to improve the accuracy. The model was also extended in [32] to include length block sequence that considers the context of packets in a time-series manner to further improve the accuracy of fingerprinting. However, these approaches only work for SSL/TLS traffic and cannot be used for general mobile app identification where a wide range of protocols, including, SSL/TLS, unencrypted HTTP, RTP, SRTP, MTProto, etc., co-exist. Furthermore, these studies only considered a few classes and it is not shown to be scalable for a large number of classes.

B. SMARTPHONE TRAFFIC IDENTIFICATION

A wide range of analysis has been done on smartphone traffic [33], including identifying users by the pattern generated from their set of installed apps [34], producing fingerprints of apps from their unencrypted HTTP traffic [35], inferring sociological information (e.g. religion, health condition, sexual preference) [36], discovering the geographical position of a smartphone [37], detecting information leakage [38], identifying user actions [39], [40], etc.

In [41], the authors used the size and timing of 802.11 frames from WiFi traffic to identify 13 iOS apps. They used random forest and achieved over 90% accuracy. However, it is only suitable when data is captured from WiFi medium. In [5], [42], AppScanner was introduced which is an approach to identify mobile apps using statistical features extracted from the vector of the sizes of packets. They captured the traffic of 110 most popular apps in Google Store and achieved high accuracy. However, they simply removed ambiguous traffic without identifying them automatically. They extend AppScanner in [5] by adding a module to identify ambiguous traffic and removing them automatically. However, they were not able to identify the source app of the ambiguous traffic.

In [43], the authors used the website fingerprinting method, proposed in [44], [45], to identify a large number of apps using three ML methods, including Gaussian naive Bayes and Multinomial naive Bayes. They only used the packet size of the first 64 packets and achieved maximum accuracy of 87%. Their dataset only contains Android traffic and it only works with captured data from WiFi. Despite

their high accuracy on the collected dataset, they concluded that each device and Android/iOS version produces different packet size fingerprints. Hence, to have a practical model that uses their method, one should collect data from all combinations of devices and Android/iOS versions, which is impractical. Additionally, they completely ignored user activities and they only captured sessions with no meaningful interaction. In [46], AntMonitor, a framework for collecting and analyzing network traffic, was introduced. They studied 70 Android apps. They used SVM and took 84 statistical features from traffic flows as input. They reported F1 measure of 70.1%. However, both these methods are not suitable for early prediction. Furthermore, their dataset only contained very limited traffic types obtained only from a few Android devices.

Recently, deep learning has also been applied for mobile app identification. In [47], a two-stage learning was proposed that performs unsupervised feature extraction in the first stage. The first stage does not need labeled data and can use public unlabeled data to improve accuracy. The second stage is a supervised category mapping using labeled data. They used variational auto-encoders and reached high accuracy for 12 apps. In [48], and their extended work [6], the authors investigated several deep learning architectures, including 1-D CNN, 2-D CNN, LSTM, SAE, and multi-layer perceptron (MLP), to identify up to 49 mobile apps. They used the first N bytes of the payload or both payload and header, depending on the setting, and even reached around 85% accuracy for top-1 classification. They used the dataset containing 49 mobile apps generated by real user activities in [49]. In [50], Aceto et al. extended their previous work by introducing a novel framework that takes two different input viewpoints into account, namely data payload and protocol/time-series features. The multi-modal deep learning model can improve the performance by capturing patterns in both viewpoints. In comparison with [6], [50], we show the applicability of deep learning models on even larger number of classes. Moreover, we perform occlusion analysis and deal with ambiguous traffic flows, which have not been studied in previous work.

III. SYSTEM ARCHITECTURE

In this section, we first introduce and define traffic objects. Then, we explain how extracted data from traffic objects is used as input features of our deep models. Then, we introduce our CNN and CNN+LSTM models which aim to solve mobile app identification and tackle ambiguous flows.

A. TRAFFIC OBJECT

Traffic object determines how raw captured packets are divided into multiple units with appropriate labels. The most common traffic object is a *flow*. A flow is a set of packets that shares the same 5-tuple (i.e. source IP, source port, destination IP, destination port, and transport-layer protocol). In such a definition, direction is taken into account. For example, a TCP connection is divided into two flows: one containing packets in a forward direction (from client to server) and the other in a backward direction (from server to client). However, in most studies, the direction of a flow is ignored and all packets in both directions are considered as a part of a single flow, sometimes called *biflow*. In this paper, we consider a flow with both directions because they achieve higher accuracy and also they essentially share the same label. For TCP connections, the beginning and the end of a connection are easily determined by TCP handshake packets, such as SYN and FIN. For other traffic types, such as UDP, a threshold (often between 15 to 60 seconds) is often used to indicate how long a flow can stay inactive before it is considered terminated.

By observing traffic flows of a single app, we notice that apps often generate flows that are common among others and they are not specific to the apps that generate them. This phenomenon is reported in [5] as *ambiguous* traffic. Ambiguous traffic includes advertisement traffic, third-party library traffic, etc. How to label such ambiguous traffic depends on the purpose of the classification. For instance, for billing purposes, it is preferred to identify the source application that generates the ambiguous traffic. In this paper, we conduct experiments with ambiguous traffic as well as without it to illustrate the effect of such traffic. More importantly, we also introduce a CNN+LSTM model that is specifically designed to improve identification of the source application of the ambiguous traffic.

B. INPUT FEATURES

Four types of input features are often used for traffic classification task [20]:

Statistical features: These features represent statistical properties of the entire flow, such as average packet length or maximum inter-arrival time, and often require to observe the entire or considerable portion of a flow. Note that in some cases such as [12], it was shown that statistical features can be computed based on the first K packets and achieved the accuracy as high as using the entire flow, e.g. the first 50 packets for maximum packet size or the first 150 packets for average packet size. Although they showed that for some specific statistical features of a one-directional flow, the first 10 packets suffice, it is still larger than the first six packets used in this paper.

Time-series features: These features contain packet-related properties of a set of consecutive packets in a flow, including the packet length, inter-arrival time, and direction. Although most papers use the time-series features of the first few packets, consecutive packets from the middle of the flow and sampled packets have also been shown to achieve good accuracy [2].

Header: It refers to layer 3 and 4 header fields, such as port number, IP/TCP flags, protocol, etc. Nowadays, header fields alone often result in poor performance. For instance, the destination port number that was once heavily used for app classification leads to a poor performance, if used alone [20]. Hence, it is often used alongside payload or time-series features.

Payload: Payload data contains raw bytes of a packet that reside above layer 4, containing application data, SSL handshake, etc. In literature, the payload of the first few packets are often used for classification.

Statistical features are not suitable for online classification, or early classification, where the result of the classification should be available as soon as possible. In many applications, such as routing, QoS provisioning, or filtering, early classification is inevitable. The usefulness of time-series features heavily depends on the dataset and application within the dataset. We find that time-series features of the first few packets are not suitable for our classification goal. Note that the choice of input feature heavily depends on the dataset and one should try all possible ones to obtain the best model as suggested in [20]. In our early experiments, the accuracy of CNN, LSTM, and random forest models with time-series of the first 30 packets was considerably lower than our approach, which is shown in Section IV. Hence, time-series and statistical features are not good identifier for our dataset. We find that payload data with deep learning models leads to a significantly better model for our dataset, although they are computationally more resource-hungry than traditional machine learning algorithms [20].

Therefore, in this paper, we use header and payload information of the first 6 packets of a flow for classification. For each packet, we only use the first 256 bytes of header and payload. We find that using more bytes or packets barely changes our model's accuracy on our dataset. If there are not enough data bytes in the packet, we pad the input data with zeros. We normalize each byte to constrain it within $[0, 1]$. The first 256 bytes of the first six packets are concatenated and the input is represented by a 1 dimensional vector of length 1536.

As far as input design is concerned, we have two options: 1) concatenate all 6 packets into a single channel, or 2) put each packet into a separate channel. In this paper, we chose the first approach for the following reason: For example, in the case of SSL/TLS, by analyzing the data, we find out that depending on how TCP handshake and SSL/TLS handshake goes, the SSL/TLS handshake packets can appear in different packets. For instance, SSL Hello often appeared in the third, forth, fifth, and sixth packets. Note that we do not remove duplicate Acks or re-transmitted packets during pre-processing because such pre-processing increases the computational complexity of an online classifier. Hence, if we use 6 different channels for each packet, the filters for each channel have to learn the same pattern all over again for each channel, which may need more data and computational resources. On the other hand, the only drawback of using a single channel and concatenating all packets is that some random patterns may appear at the position where two packets are concatenated. We empirically find the first option is slightly better, but the difference is not very significant.

C. MODEL ARCHITECTURE FOR APP IDENTIFICATION

The goal of app identification is to assign a correct label to each flow only using the input features of that flow. In other words, the input of the deep learning model is the header and payload data of the first few packets of a flow, as explained in Section III-B, and the output is the corresponding class label. In our first model, we use convolutional neural networks (CNNs) as a deep learning model for app identification.

CNNs, inspired by visual mechanisms of living organs, work well with high dimensional inputs. Although multi-layer perceptrons (MLPs) are shown to be powerful, they fail to be efficiently trained on high dimensional input data which requires a large amount of hidden parameters to be trained. CNNs reduce the number of learning parameters significantly by using a set of convolution filters (kernels) in convolutional layers. These convolutional filters have only a few learning parameters and are moved on the entire input to cover all of it. In other words, instead of learning parameters of each section of the input individually, the model learns a set of filters applied to the entire input. This also helps the CNN models to be shift-invariant because the same filter can detect the same patterns regardless of its position on the input.

In addition to convolutional layers, CNNs also have pooling and fully connected (FC) layers. A typical CNN structure is shown in Figure 1. Pooling layers act as a non-linear down-sampling. The most common form of pooling is max pooling. In max pooling, the maximum values of each cluster of neurons are selected and used as an input of the next layer. Pooling layers reduce the size of their input and, consequently, reduce the number of training parameters and computation. FC layers are often used for the last few layers and they are known to perform the high level reasoning.

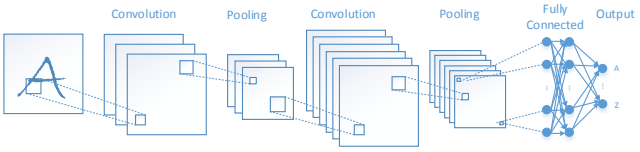


FIGURE 1: A typical CNN structure

In this paper, we use a 1 dimensional CNN, as shown in Figure 2. We concatenate the 256 bytes of the first 6 packets into a single vector to feed to the CNN model. The reason we use a CNN model is that they are shift-invariant. This is crucial because the patterns that we find out to be useful for classification can appear almost anywhere in a packet. For example, TCP options can appear in a range of bytes, depending on the size of IP header and other TCP options. Moreover, app-specific data patterns can also appear anywhere in the payload. We also show that SSL/TLS handshake extensions are very useful in classification of SSL/TLS traffic and they can appear in a wide range of bytes within a packet. Since the location of such features are not predetermined in a packet, shift-invariant models are more suitable.

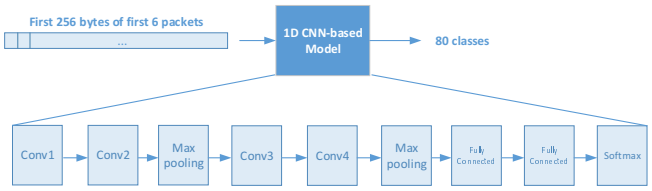


FIGURE 2: The proposed CNN model

D. AMBIGUOUS FLOW IDENTIFICATION

When a single app is running, such as YouTube, not all traffic flows are directly related to the content of the video streaming. For example, some flows contain Google analytics or Ads data. Such traffic flows that are not specific to the app and can also be generated by other apps are called ambiguous flows. A typical example of such a session is shown in Figure 3. There are many other flows during the session. But, we only keep 5 flows to have a clear figure. These flows are generated as a result of the YouTube session. Hence, we are interested in identifying the source app that causes these flows to be generated. In other words, we want to label Google Ads and all other ambiguous flows as YouTube. However, the Google Ads traffic may also appear in other Google apps, such as Google Music, where we want to label (the source app of) Google Ads as Google Music when it generates the Google Ads traffic. Clearly, to identify the source app that generates the Google Ads traffic, only looking at the Google Ads traffic would lead to low accuracy.

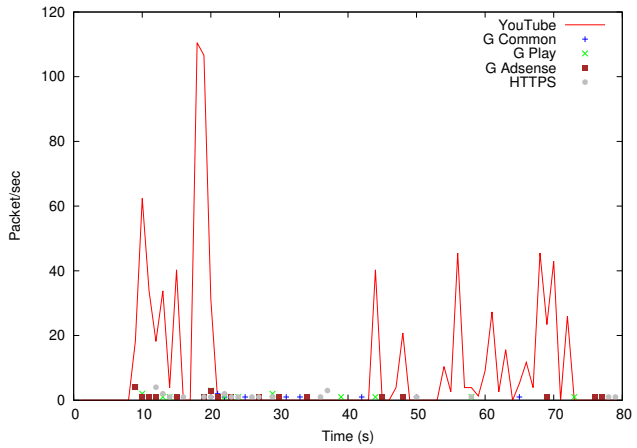


FIGURE 3: A sample of YouTube session with ambiguous flows

To solve the problem, we propose a multiple flows classification scheme where several subsequent flows are considered to identify the source app of the target flow. We call this problem *flow association* because, unlike typical traffic classification, we exploit the relation of several adjacent flows all together. Here, the target flow refers to the flow that we are interested in identifying the source app and the source app refers to the app that generates the ambiguous target flow. Our intuition is that although a single ambiguous flow may not provide enough information to identify the source app, looking at the adjacent flows and observing their order, patterns, and relative starting time will help identify the source app. Hence, in our model, to identify the source app of a target flow, we also use the next k flows as well. Note that in many cases all adjacent k flows that we feed to the model are ambiguous flows, but our insight is that each app generates a unique set of ambiguous flows with unique patterns and order. Furthermore, if the target flow is not an ambiguous flow, feeding the next adjacent flows should not hurt the accuracy.

Interestingly, Google services generate the most number of ambiguous traffic flows in our dataset. Hence, for flow association task, we focus on 7 Google classes: Google Map, Google Music, Hangouts, Gmail, Google Earth, YouTube, and Google Play. In addition to these traffic flows, Google apps also generate the following flows: Google Common, Google Analytics, Google Search, Google Adsense, TCP Connect, HTTP, and HTTPS. So, in total, we have 7 source apps and 7 ambiguous traffic labels. We are interested in identifying the source app of each flow. In other words, the final classification task is a 7-class classification (i.e. Google Map, Google Music, etc).

To achieve this goal, we also use a type of recurrent neural networks (RNNs), called LSTM, which is suitable for sequential data. Such models have been successfully applied to speech recognition, translation task, time series prediction, and language modeling. The output of a LSTM model depends not only on the last input, but also on the previous inputs. In the flow association task, we can consider each flow as a single input and the set of consecutive flows as a sequence of input data which is suitable for LSTM.

To solve the problem of identifying source apps, we introduce a two-step training process. We first train a CNN model to predict the 14 true class labels using single flow. True class labels refer to the 7 Google apps and 7 ambiguous traffic labels altogether. We use the same CNN architecture and data input format as the previous section, as shown in Figure 2. At the next step, we use an LSTM model to predict the source application of a flow. To do that, we feed not only the input features of the target flow, but also the input features of the next k flows. The structure of the model is shown in Figure 4. We first pass each flow input feature through the CNN model that predicts the true label. The weights of the CNN model are transferred from the first training step, shown in Figure 2. Then the predicted output is concatenated with the relative time of the flow (i.e. relative to the first/target flow). Then, it is fed as the first step of the LSTM model. We repeat the same process for the next k flows as well. Then, the LSTM model is expected to predict the source app of the first flow.

We hypothesize that the flows related to a single session of an app start close to each other, as shown in Figure 3. Hence, we define a threshold of 2 seconds to include adjacent flows. In other words, we only feed the CNN+LSTM model with the next two flows within the 2 seconds of the first flow. Otherwise, we pad the remaining flows input with zeros. The

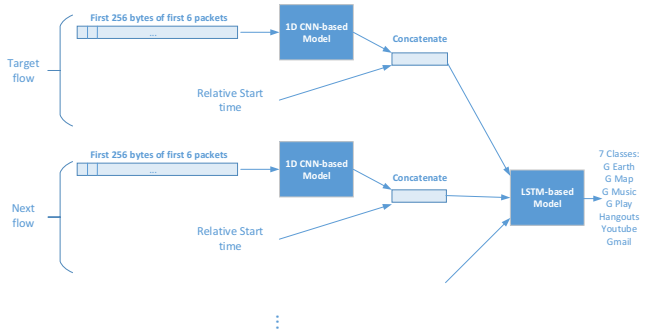


FIGURE 4: The joint CNN+LSTM model

2 seconds threshold also ensures that the traffic flows of two different apps are not mixed because it is highly unlikely that a user opens two different Google apps within two seconds. Hence, the flows that are fed to the CNN+LSTM model corresponds to only one app. However, in reality, the traffic flows of two different source apps may appear close to each other if one is run on the background. In Section IV-D, we evaluate the robustness of our approach to such mixed traffic scenarios.

IV. EVALUATION

In this section, we describe our large mobile traffic dataset. Then, we evaluate our CNN-based model for app identification task with 80 classes. We use accuracy, precision, recall, and F1 to report the performance in different scenarios. We also illustrate the confusion matrix to show the most common misclassification patterns. Moreover, we perform occlusion analysis on the SSL/TLS encrypted traffic to shed a light on why apps using such an encrypted traffic protocol can also be identified. Finally, we evaluate our flow association approach using CNN+LSTM model and illustrate that such a model can improve the accuracy of identifying the source app of ambiguous flows.

A. DATASET DESCRIPTION

The data was collected under controlled mobile operators networks in 2018[1]. Two methodologies were used to obtain data: 1) Human users interacting with each app, where the data was captured by Wireshark. 2) Mobile device emulators where scripts emulated how users access the apps without human intervention. The human-generated portion of data and script-generated portion are roughly the same in terms of volume. The labels were assigned in a controlled environment where all unrelated apps were closed or removed.

The dataset contains 80 mobile apps, including streaming, social media, messaging, Email, and navigation apps. It contains app traffic for both Android and iOS devices, except for apps that only have one version, such as Apple Map and Bit Torrent. Figure 5 shows the frequency of each application traffic in our dataset. As it is shown, the dataset is highly imbalanced. To mitigate the imbalance problem in the dataset, we use random under-sampling on the two most frequent apps (that is, QQLive and LETV) and random over-sampling on all apps with less than 1% frequency. The reason we do not use more advanced over-sampling methods, such as Synthetic Minority Over-sampling Technique (SMOTE), is that our input feature is high dimensional. Since we feed the model with the raw packet payload without any feature selection, the data points of the same class maybe far away in the input space. So, methods such as SMOTE may create unrealistic synthetic points.

As shown in Figure 5, the majority of the traffic flows are TCP and only around 10% are UDP. Only streaming or video/voice applications contain a significant amount of UDP traffic, such as QQLive, PPLive, Facebook messenger, WhatsApp, Hangouts, etc. UDP traffic flows of Google services, such as YouTube and Hangouts, use Google QUIC protocol. Other UDP traffic mostly have RTP or SRTP (e.g. Skype and WhatsApp) or some custom protocols (e.g. MTProto for Telegram).

TCP traffic flows contain plain HTTP, HTTPS and a small number of other protocols. HTTPS constitute 45.05% of the TCP traffic flows and 40.36% of the entire dataset. Figure 6 illustrates the amount of HTTPS traffic among TCP flows for each app. The traffic flows are shown based on encryption protocols and their version. Note that SSL and TLS are the primary encryption protocols used for HTTPS. While a SSL/TLS connection can be established on any port number, including port 80 [51], we do not observe a significant amount of such flows. As it is shown, some applications (on the right side of the figure) heavily use HTTP instead of HTTPS. Moreover, some applications, such as Sky TV, still use SSL 3.0 which is deprecated. Furthermore, many applications, including Waze, Google Earth, Maxdome, still use TLS 1.0 traffic instead of the much stronger versions, such as TLS 1.2 or TLS 1.3. As it is reported in [51], this is mainly a result of misconfiguration by developers because the dataset was captured using a wide range of OS versions that mostly support TLS 1.2.

B. APP IDENTIFICATION USING CNN MODEL

1) Classification without Ambiguous Traffic

In this section, we train the CNN model, explained in Section III-C, for the 80-class classification task. The details of the CNN model are summarized in Table 1. Batch normalization is also used at the end of each pooling layer. We train the model with Adam optimization algorithm with maximum of 30 epochs and 0.001 as the learning rate. We use early stopping, similar to [25], where the training is terminated if the loss function does not improve for 5 epochs. Moreover, the whole training data is used during one epoch. Our results are based on 10-fold cross-validation. In the first experiment, we remove all ambiguous traffic from the dataset. It has been shown that it is possible to remove ambiguous traffic automatically in [5].

[1]Due to the NDA with the data provider, we cannot reveal the name, network, and other details about the dataset.

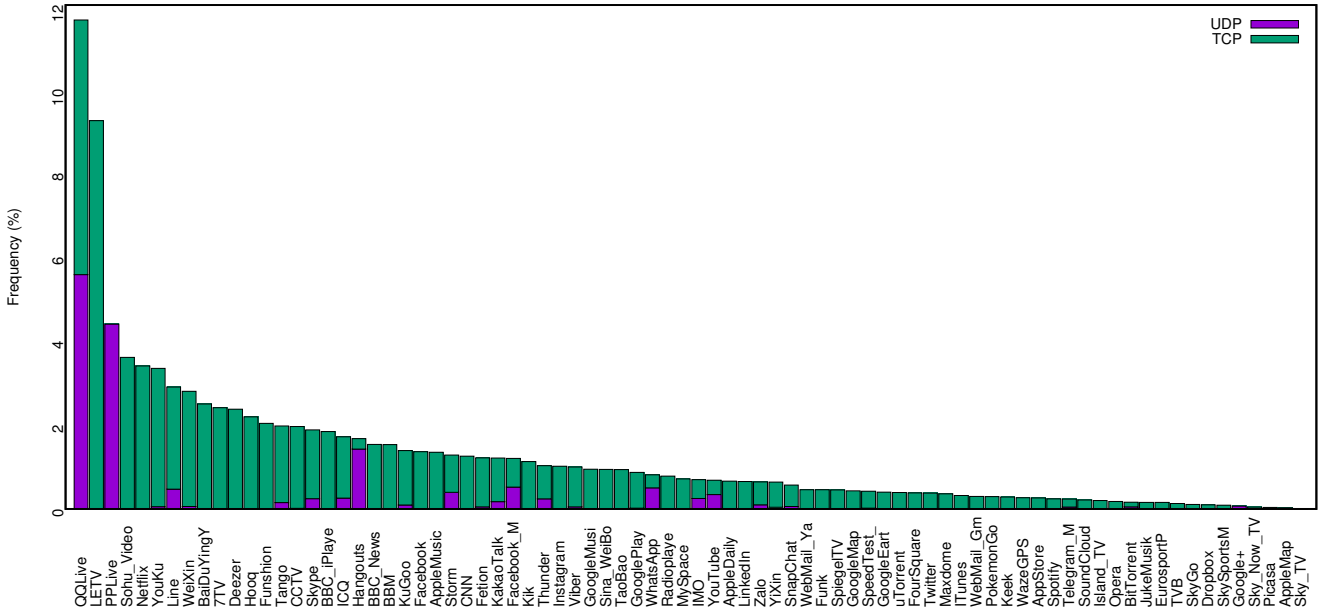


FIGURE 5: Frequency distribution of 80 classes.

To compare our approach with state-of-the-art methods, we also conduct experiments with deep learning models on time-series features, similar to [25], and random forest on statistical features, based on [48]. In [25], several models, including LSTM (called RNN in [25]), CNN and various combinations of them, are trained on time-series features. We use their CNN, RNN, and their best performed model, namely CNN+RNN-2, in our experiment. We use direction, payload size, inter-arrival time, and window size (only in TCP packets). Furthermore, we also train a random forest (RF) model similar to the experiment in [48]. The RF models takes 40 carefully handcrafted features obtained from the entire flow [42]. Other details of our experiments are similar to [25] and [48].

The results of the CNN model are shown in Table 2. The highest accuracy is 99.12% when the entire 256 bytes in the first 6 packets are used. However, due to the limited number of IP addresses in the dataset, the model may overfit to non-related features. Hence, we removed the layer 2 and 3 data and further evaluated the model's accuracy. Moreover, we also removed layer 4 data as well to investigate the effect of port numbers and TCP options on accuracy. In these cases, we still used the first 256 bytes of the first six packets, but we padded the those layers to remove them. As shown in Table 2, by only using the payload data without any header data, we can obtain 94.22% accuracy. This clearly illustrates that mobile apps leave enough fingerprints for identification. Moreover, the accuracy of identifying apps are 96.87%, 80.64%, for HTTP and HTTPS traffic, respectively. This shows that even encrypted traffic has enough information for reasonable accuracy. We analyze how our model can classify encrypted traffic in Section IV-C.

2) Classification with Ambiguous Traffic

In the second experiment, we keep the ambiguous traffic flows and the goal is to label them based on their source apps that generate them. This is a significantly harder task since certain types of ambiguous flows, such as advertisement traffic, are often generated by multiple apps and they might not have enough distinguishable features to properly identify the source app. Table 3 presents the performance metrics of the second experiment. By comparing the results to the first experiment, we find that the presence of ambiguous traffic degrades the accuracy considerably. However, the accuracy of the model without header data is 84.01% which is still higher than previous studies for large-scale app identification, such as [5] with the highest accuracy of 73.5% for a 65-class classification and similar settings. The biggest difference between our study and [5] is that we use payload data for classification whereas they use time-series features. This shows that, even in the presence of encryption protocols, payload data still contains information that can improve the classification accuracy.

Accuracy across different classes varies. Figure 7 shows the confusion matrix of the second experiment. Among all classes, Sky TV, Telegram, Picasa, and Google+ are the hardest to classify. These apps are amongst the classes with lowest number of samples. Moreover, according to Figure 6, Sky TV and Picasa are amongst the classes with highest ratio of SSL and TLS traffic which make them hard to identify.

Sometimes the traffic patterns are different for different versions of an app, particularly between Android and iOS

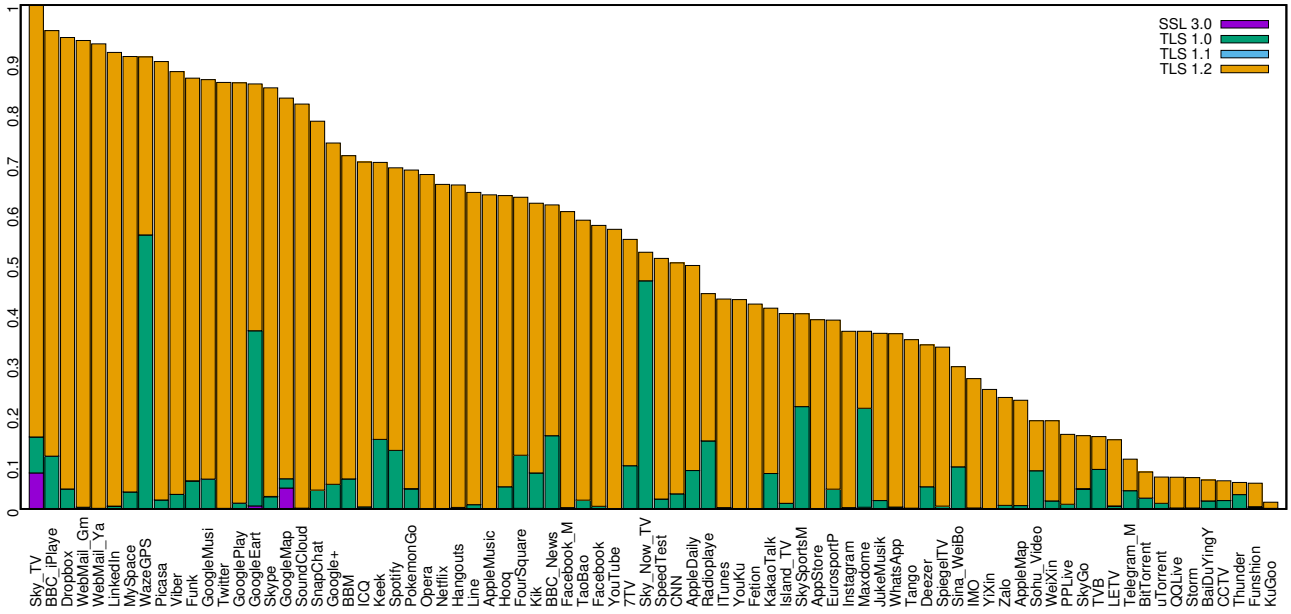


FIGURE 6: Portion of TCP flows encrypted with SSL/TLS

TABLE 1: Structure of the CNN model

-	Conv1	Conv2	Pool3	Conv4	Conv5	Pool6	FC7	FC8
Number of filters/neurons	256	256	-	128	128	-	128	128
Kernel size	3	3	2	2	2	2	-	-
stride size	1	1	2	1	1	2	-	-

TABLE 2: 80-class classification without ambiguous traffic

Input detail	Precision	Recall	F1	Accuracy
All layers	99.53%	98.72%	99.03%	99.12%
L 2/3 removed	98.36%	95.58%	96.78%	96.95%
L 2/3/4 removed	96.99%	93.22%	94.99%	94.22%
RNN-1 [25]	81.96%	74.36%	77.97%	78.55%
CNN-1 [25]	71.84%	70.04%	70.92%	71.27%
CNN+RNN-2 [25]	82.43%	78.11%	80.21%	80.22%
RF [48]	64.21%	59.05%	61.52%	63.58%

TABLE 3: 80-class classification with ambiguous traffic

Input detail	Precision	Recall	F1	Accuracy
All layers	96.39%	94.67%	95.52%	95.90%
L 2/3 removed	90.55%	90.12%	90.32%	90.35%
L 2/3/4 removed	84.81%	83.21%	83.99%	84.01%
RNN-1 [25]	70.32%	67.12%	68.68%	69.61%
CNN-1 [25]	63.27%	59.94%	61.56%	62.22%
CNN+RNN-2 [25]	74.82%	70.74%	72.72%	72.97%
RF [48]	61.83%	59.29%	60.53%	61.24%

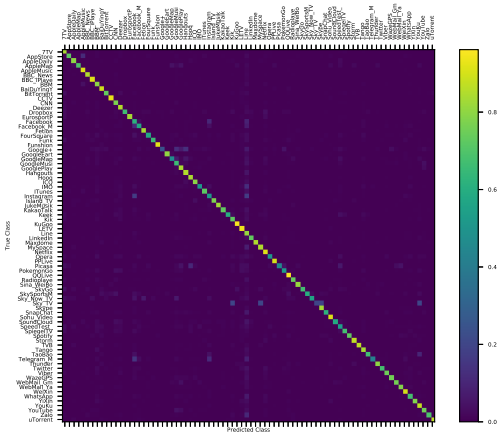


FIGURE 7: Confusion matrix of the 80-class classifier

versions. To see whether classification accuracy is different for iOS and Android, we test the performance separately. Table 4 presents the performance metrics of the two versions with a model trained only on the payload data. Although our dataset contains more samples of Android apps, the accuracy, recall, and precision of the two versions are not significantly different.

We also test the performance on different types of traffic. As shown in Table 5, the highest performance is obtained for unencrypted HTTP which is around 45% of the entire dataset. UDP traffic covers about 10% of the entire dataset and the accuracy is lower than the unencrypted HTTP. HTTPS traffic, covering around 40% of the entire dataset, has the lowest

TABLE 4: Performance of iOS vs Android (L 2/3/4 removed). Dist stands for distribution of the traffic within the entire dataset.

Device	Dist	Precision	Recall	F1	Accuracy
iOS	43.61%	84.11%	83.35%	83.74%	83.41%
Android	56.39%	85.31%	83.98%	84.63%	84.54%

TABLE 5: Performance of different traffic types (L 2/3/4 removed)

Traffic	Dist	Precision	Recall	F1	Accuracy
TCP	89.58%	84.46%	83.52%	83.99%	83.72%
UDP	10.41%	88.25%	86.71%	87.47%	86.76%
HTTP	45.58%	91.72%	91.43%	91.57%	91.60%
HTTPS	40.36%	76.66%	74.53%	75.58%	75.43%

accuracy. Although, the encrypted traffic should supposedly not reveal information about the content of the traffic, the accuracy is still significantly high (75.43%) for 80-class classification. In Section IV-C, we analyze the features that allow the classification of the SSL/TLS traffic by conducting occlusion analysis.

Figure 8 illustrates the confusion matrix of only the HTTPS traffic. The majority of the other apps are classified correctly by our model even for HTTPS traffic, as shown in Figure 8. The recall of the AppleMap, and BitTorrent classes were almost zero because both of these classes have a small number of samples and among them only a small portion of samples are HTTPS. KuGoo's recall for HTTPS is also extremely low since it has the lowest number of HTTPS samples. Note that the overall recall of these classes are high because the majority of their flows are non-HTTPS and our model can correctly identify them. The recall of the AppStore, BaiDuYingY, Google+, IMO, Opera, Picasa, SkyGo, SkyTV, SpeedTest, Telegram, Thunder, Zalo, and uTorrent are also less than 60%. Although, some of these apps have many HTTPS samples, their low recall shows that they have similar features to other applications that make the classification difficult. For example, IMO traffic is often confused with Instagram and Facebook.

C. OCCLUSION ANALYSIS

In this section, we perform occlusion analysis to reveal the features that allow the model to classify HTTPS encrypted traffic. We do not analyze the unencrypted HTTP traffic because deep packet inspection (DPI) methods have been shown to work on unencrypted traffic and it is trivial that payload information in traffic such as HTTP provides significant information about the apps. Moreover, for occlusion analysis of UDP traffic, we need to know the exact protocol (e.g. RTP, SRTP, DTLS, etc.,) used in each flow to be able to associate each byte of the payload to the protocol's fields. However, it is not possible for the majority of UDP traffic flows that are captured in the middle of the route to do such analysis. Hence, we only perform occlusion analysis for HTTPS traffic

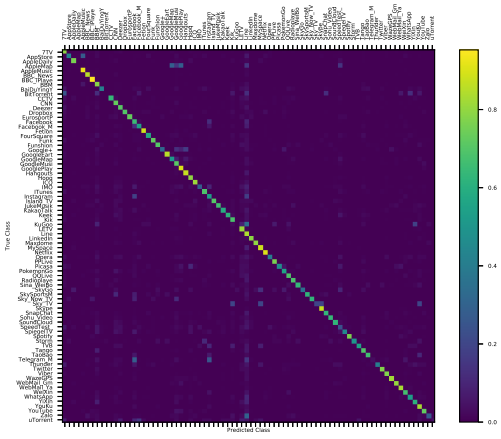


FIGURE 8: Confusion matrix of the 80-class classifier on HTTPS traffic

in this paper.

Occlusion analysis has been used extensively in image recognition task to show the sensitivity of CNN-based models to different components of an input image [52]. Typically, it is done by blocking a part of an image at inference time to see how it affects the classification accuracy. For example, one can block the trunk of an elephant on animal classification task, to see how much the model is sensitive to the trunk for classification.

In this section, we replace different parts of SSL/TLS handshake fields with a default value at inference time to see the effect of that fields on the classification accuracy. Note that not all traffic flows contain all SSL handshake fields. Moreover, each handshake field can appear at a wide range of bytes within a packet, depending on the size of previous layers and order and existence of SSL/TLS extensions. Hence, we wrote a parser that parses the entire packet and finds the target SSL/TLS fields and replaces it with a default value (zeros).

Table 6 presents the result of the occlusion analysis on the HTTPS traffic using SSL/TLS. Handshake fields with the most significant impact on the accuracy are highlighted. Extension number zero, or Server Name Indication (SNI), is the most important field in the SSL/TLS handshake. The importance of SNI is not surprising since it contains the destination hostname a client wants to connect and sometimes it is even used to obtain the ground truth [28]. Cipher info is also useful for classification because it covers a wide range of cipher suite's combinations that may reveal target apps. Interestingly, SSL/TLS length also helps app identification. This is because different combinations of the extensions lead to different SSL/TLS length values and may reveal target apps.

Interestingly, unassigned extensions (number 54 to 65279) also have an impact on the accuracy. It is reasonable because these extensions are used in a custom way by apps and it

TABLE 6: 80-class classification occlusion analysis

Occluded part	Accuracy
-	75.43%
Version	75.34%
Record+Handshake Length	**59.31%**
Handshake type	75.44%
Rand (Client Hello)	71.40%
Rand (Server Hello)	75.31%
SID info (Client Hello)	71.84%
SID info (Server Hello)	75.30%
Compression info (Client Hello)	75.43%
Compression info (Server Hello)	75.43%
Cipher info (Client Hello)	**51.67%**
Cipher info (Server Hello)	75.36%
Certificate info (type 11)	74.39%
Server key exchange info (type 12)	75.43%
Client key exchange info (type 16)	75.43%
All extensions	**27.52%**
Extension #0 (SNI)	**38.19%**
Extension #5 (Status request)	75.40%
Extension #10 (Supported Group/Elliptic Curves)	72.20%
Extension #11 (Elliptic curve point format)	74.93%
Extension #13 (Signature algorithm)	75.01%
Extension #16 (App layer protocol negotiation)	72.26%
Extension #18 (Signed certificate timestamp)	75.12%
Extension #23 (Extended master secret)	74.87%
Extension #35 (Session ticket)	73.97%
Extension #54-65279 (Unassigned)	**66.32%**
Extension #65281 (Renegotiation info)	72.36%
Cipher info (Client), All extensions	12.42%
Cipher info (Client), All extensions, Rand (Client)	8.20%

is highly unlikely that two apps use the same unassigned extension. So, for apps that use unassigned extensions, it is an important feature for classification. We observe unassigned extensions across 7 different mobile applications, including Gmail, Tango, and BBM.

It is interesting to see that removing the random number in the client hello also slightly degrades the accuracy. Note that the random field in the client hello packet consists of GMT time and a random number. In our dataset, some applications are captured at only a few time instances. As a result, we believe that the GMT part of the random number is probably exploited by the model. In a completely unbiased dataset, the client random number should not affect the performance of the model.

Our analysis is mostly based on client hello and server hello packets. Note that we only use the first 6 packets of each flow for classification. As a result, we mostly feed the model with the client and server hello packets. Only a small portion of flows has server certificate, server key exchange, or certificate request in their first 6 packets, but we found out that it did not have a significant role in the model's accuracy. Note that our model achieved acceptable accuracy for 80-class classification. Hence, we did not take more packets into input features to avoid making the model highly complex for a negligible performance gain. Moreover, using fewer packets for classification is more suitable for early (online) classification. That is the reason we only report data fields of client and server hello packets in Table 6.

TABLE 7: 7-Google-class classification with ambiguous traffic (Thr stands for threshold)

Model	Thr	Precision	Recall	F1	Accuracy
CNN	-	92.01%	91.97%	91.99%	91.98%
CNN+LSTM	0.5	95.37%	95.01%	95.19%	95.08%
CNN+LSTM	1	95.88%	95.91%	95.89%	95.91%
CNN+LSTM	2	96.35%	96.23%	96.25%	96.23%
CNN+LSTM	3	96.30%	96.23%	96.26%	96.29%
CNN+LSTM	4	96.32%	96.19%	96.25%	96.29%
RNN-1 [25]	-	77.17%	75.47%	76.31%	76.41%
CNN-1 [25]	-	71.38%	68.06%	69.67%	69.97%
CNN+RNN-2 [25]	-	82.56%	81.08%	81.81%	82.13%
RF [48]	-	70.87%	66.24%	68.47%	68.89%

D. SOURCE APP IDENTIFICATION FOR AMBIGUOUS TRAFFIC

1) Multiple Flows Evaluation

In this section, we show that our flow association approach with multiple flows can better solve the identification of source apps of ambiguous flows. In this section, we only focus on 7 Google services that generate the most ambiguous traffic flows, that is, Google Map, Google Music, Hangouts, Gmail, Google Earth, YouTube, and Google Play. Although we show that 80-class classification is achievable, one can easily divide traffic flows into several groups based on the IP addresses. For instance, Google traffic flows can be identified by the pool of Google IP addresses. However, the IP address is not useful for fine-grained classification of apps. Hence, we still need a classifier that looks into payload or other features. In all the evaluations in this section, we remove all layer 2 to layer 4 data headers.

Similar to Section IV-B, we first train the CNN model, described in Section III-C, for Google service classification. The training procedure for the CNN part is explained in Section III-C. We use a single LSTM layer with 50 units and dropout ratio of 0.5. For the CNN+LSTM model, we re-train the entire model with Adam optimization algorithm with maximum of 50 epochs. Other training parameters are similar to Section III-C. The performance results are shown in Table 7. As it is shown, the single-flow classification only achieves 91.98%. Note that RNN-1, CNN-1, and CNN+RNN-2 models take time-series features as input, whereas our CNN and CNN+LSTM models take payload data.

To improve the accuracy of ambiguous flows, we conduct an experiment, explained in Section III-D. We first train the CNN model to classify all 14 classes, including both Google apps labels and ambiguous class labels, shown in Figure 2. Then, we transfer the weight of the CNN model to the CNN+LSTM model, shown in Figure 4. Then, we train the CNN+LSTM model with multiple flows as described in Section III-D. For each flow, we find that the next two flows (within two seconds of the target flow) are enough

for multiple-flow classification with high accuracy. So, we only feed the CNN+LSTM model with only three flows. The performance of the CNN+LSTM model is shown in Table 7. The multiple-flow classification improves the accuracy more than 4% which is significant since the majority of Google traffic is encrypted. There is a trade-off between accuracy and how early we can make a prediction. As the time threshold increases, the accuracy improves. However, it is also desirable to classify apps as early as possible. Given the accuracy does not significantly improve for thresholds that are larger than 2 seconds, we choose the 2-second threshold. Note that there is a higher chance that flows are generated from different apps if we consider two far away flows in the time domain. Hence, we choose a relatively small threshold that does not affect accuracy too much.

2) Robustness Analysis

In reality, however, traffic flows of other Google apps may appear close to the target flow. In such cases, the second or third flows that are fed to the CNN+LSTM model may not have the same source apps as the first/target flow. To evaluate the robustness of our model to such a mixed traffic flows, we conduct the following experiment with noisy data: First, we duplicate each sample (containing 3 adjacent flows) in our dataset. Then, for each duplicate sample, we randomly replace a flow in our multiple-flow input with a random flow from another source app. The modified samples mimic traffic when captured in the real world where apps are not run in isolation. The results are shown in Table 8. The number after *R* represents the flow number that was replaced: R3 means only the third flow is replaced, R2 means only the second flow is replaced, and R23 means both second and third flows are replaced with random flows of another app. In Table 8, *T* represents the scenarios where we also train the model with replaced flows to mimic the flows in a real network.

As is shown, the model is pretty robust when only one flow (out of three flows) is replaced with another app flows (R2 and R3 scenarios). However, when two flows are replaced (R23 scenario), the accuracy of the CNN+LSTM model with multiple flows becomes lower than the CNN model with single flow (Table 7). This is reasonable because it adds noisy data at inference time without training the model to handle such noisy data. However, if we train the model to deal with such a mixed traffic scenario, the model outperforms single flow classification in all scenarios of mixed flows (R2+T, R3+T, and R23+T). That is probably because the model learns to identify mixed traffic and it only uses the second and third flows when it is related to the first flow. Therefore, our flow association formulation with CNN+LSTM model always achieves highest accuracy for app identification with ambiguous traffic.

Note that during the evaluation, we find that if we feed the CNN+LSTM model with the sequence of flows in reverse order, we achieve better accuracy. In other words, we first take the third flow, then the second next flow, and then the first/target flow as input to the model. As it is shown in Table 8, the accuracy of the model is slightly higher when it is fed with flows in backward direction, particularly when we train the model with noisy data. The reason is related to how forget gates work in LSTM models. These gates help the model to forget the previous hidden representation if they receive particular kinds of input. In many cases, particularly in the mixed traffic scenario, the most relevant flow to identify the source app is the first/target flow, although it might be ambiguous. So, by feeding the first/target flow last, we allow the LSTM model to forget the second and third flows if they are unrelated. All the results in this section is for the input in reverse order. However, to avoid confusion, we still use the first flow to refer to the target flow in this section.

TABLE 8: Robustness of the CNN+LSTM model to mixed traffic flows. The number after *R* represents the flow(s) number that was replaced. *T* represents the scenarios where the model trained with the replaced flows to mimic the flows in a real network.

Scenario	Direction	Precision	Recall	F1	Accuracy
R3	Backward	95.34%	94.85%	95.08%	95.09%
R2	Backward	95.11%	94.86%	94.98%	94.99%
R23	Backward	91.20%	91.06%	91.10%	91.10%
R3+T	Backward	96.15%	95.98%	96.06%	96.03%
R2+T	Backward	95.25%	95.76%	95.49%	95.74%
R23+T	Backward	95.59%	95.41%	95.49%	95.50%
R3	Forward	95.05%	94.99%	94.01%	95.00%
R2	Forward	95.02%	94.82%	94.86%	94.89%
R23	Forward	90.60%	90.51%	90.55%	90.55%
R3+T	Forward	96.03%	95.91%	95.96%	95.99%
R2+T	Forward	95.46%	95.30%	95.36%	95.38%
R23+T	Forward	95.11%	94.56%	94.83%	94.85%

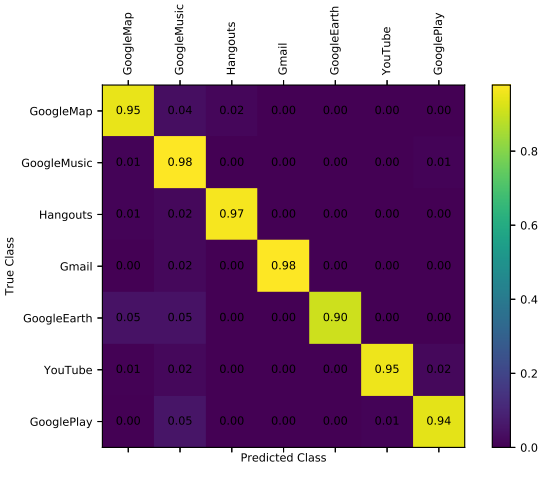


FIGURE 9: Confusion matrix of the multiple-flow CNN+LSTM model for flow association

The confusion matrix of the 7-class app identification is

TABLE 9: 7-Google-class occlusion analysis

Occluded part	Accuracy
-	90.95%
Record+Handshake Length	89.30%
Rand (Server Hello)	87.33%
Cipher info (Client Hello)	**70.64%**
All extensions	**59.14%**
Extension #0 (SNI)	87.31%
Extension #16 (App layer protocol negotiation)	88.10%
Extension #18 (Signed certificate timestamp)	**79.23%**
Extension #35 (Session ticket)	89.36%
Extension #54-65279 (Unassigned)	89.03%
Extension #65281 (Renegotiation info)	85.72%
Cipher info (Client), All extensions	32.85%

illustrated in Figure 9. Google Music, Hangouts, and Gmail are the apps with the highest recall and Google earth has the lowest recall. Similar to the CNN model for single-flow classification, accuracy of HTTPS traffic is lower than unencrypted traffic. The accuracy of HTTPS traffic is 90.95%, as it is shown in the first row of Table 9.

3) Occlusion Analysis of the CNN model for Google Services

Table 9 presents the occlusion analysis for Google services. Here, we only report the occluded parts that degrade the accuracy to lower than 90% for brevity. The occlusion analysis of the model for 7 Google apps reveals a small difference in comparison with CNN model of 80 apps. Instead of the SNI extension, the most sensitive fields are client cipher info and extension number 18. Moreover, accuracy is not highly dependent on a single extension. But, if all extensions are occluded, the accuracy drops significantly. In both models, all extensions together are the most sensitive data field for classification of SSL/TLS traffic.

V. CONCLUSION

In this paper, we introduce two deep learning models to tackle the mobile app identification task. We obtained a large dataset of 80 mobile apps from a large ISP in 2018. Analyzing the data revealed that around 90% of traffic flows were TCP, of which around 45% were encrypted with SSL or TLS. Furthermore, many apps generate not only app-specific traffic, but also ambiguous traffic that refers to common traffic, such as advertisement, that is generated by many other apps.

Using a CNN model and raw payload data of the first 6 packets of each flow without layer 2 to 4 headers, we achieve 94.22% accuracy for 80-class classification for non-ambiguous traffic flows. When ambiguous traffic flows are also taken into account, the accuracy of our model is 84.03% which is reasonable because identifying the source apps that generate the ambiguous traffic flows is a much more challenging task. We find that HTTPS traffic flows are the hardest one to classify. However, our model's accuracy is 75.43% for HTTPS showing the feasibility of mobile app identification even for encrypted traffic in large-scale.

We also perform occlusion analysis on SSL/TLS traffic flows to understand why these encrypted traffic flows are still classifiable. We find out that unencrypted handshake fields, including cipher info and many extensions, reveal enough information for identifying mobile apps.

To improve the accuracy of classification, particularly for ambiguous flows, we introduce a CNN+LSTM model that takes the first 6 packets of three consecutive flows to identify the first flow. The intuition is that when an app launches, it generates several flows in a short time and some of them are ambiguous. CNN+LSTM model can look at several adjacent flows and capture the relation and order of these flows which helps improve the accuracy of traffic identification. We show that CNN+LSTM model improves the accuracy of Google app identification from 91.98% with the CNN model to 96.23% with the CNN+LSTM model. To the best of our knowledge, this is the first time that adjacent flows are also used to improve the accuracy of classification.

In this paper, we use payload data and a deep model to classify adjacent flows and tackle ambiguous flow problem. In the future, we study how we can reduce the complexity of the model by using time-series and statistical features alongside a simpler machine learning algorithm. Moreover, as more secure protocols are constantly deployed, such as TLS 1.3 which exchanges fewer unencrypted handshake fields, using a classifier that relies on unencrypted payload data may not achieve acceptable accuracy. However, using adjacent flows may help the classifier if not all adjacent flows are encrypted with a strong protocol, such as TLS 1.3. Moreover, we will study if adjacent flows using QUIC protocol can also be identified by our approach as a future work.

REFERENCES

[1] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, "Towards self adaptive network traffic classification," Computer Communications, vol. 56, pp. 35–46, 2015.

[2] S. Rezaei and X. Liu, "How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets," arXiv preprint arXiv:1812.09761, 2018.

[3] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," Soft Computing, pp. 1–14, 2017.

[4] S. Rezaei and X. Liu, "Multitask learning for network traffic classification," arXiv preprint arXiv:1906.05248, 2019.

[5] 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.

[6] 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, 2019.

[7] 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.

[8] 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.

[9] ——, "An investigation on the identification of voip traffic: Case study on gtalk and skype," in 2010 International Conference on Network and Service Management. IEEE, 2010, pp. 310–313.

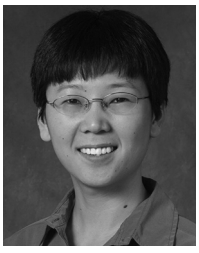
[10] R. Bar-Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime classification for encrypted traffic," in International Symposium on Experimental Algorithms. Springer, 2010, pp. 373–385.

[11] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," Journal of Machine Learning Research, vol. 7, no. Dec, pp. 2745–2769, 2006.
[12] 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.
[13] A. R. Khakpour and A. X. Liu, "An information-theoretical approach to high-speed flow nature identification," IEEE/ACM Transactions on Networking (TON), vol. 21, no. 4, pp. 1076–1089, 2013.
[14] L. Kong, G. Huang, and K. Wu, "Identification of abnormal network traffic using support vector machine," in 2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT). IEEE, 2017, pp. 288–292.
[15] Y. Okada, S. Ata, N. Nakamura, Y. Nakahira, and I. Oka, "Application identification from encrypted traffic based on characteristic changes by encryption," in 2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR). IEEE, 2011, pp. 1–6.
[16] G.-L. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An novel hybrid method for effectively classifying encrypted traffic," in 2010 IEEE Global Telecommunications Conference GLOBECOM 2010. IEEE, 2010, pp. 1–5.
[17] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in International Conference on Passive and Active Network Measurement. Springer, 2007, pp. 165–175.
[18] M. Zhang, H. Zhang, B. Zhang, and G. Lu, "Encrypted traffic classification based on an improved clustering algorithm," in International Conference on Trustworthy Computing and Services. Springer, 2012, pp. 124–131.
[19] Y. Du and R. Zhang, "Design of a method for encrypted p2p traffic identification using k-means algorithm," Telecommunication Systems, vol. 53, no. 1, pp. 163–168, 2013.
[20] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," IEEE Communications Magazine, vol. 57, no. 5, pp. 76–81, 2019.
[21] P. Wang, X. Chen, F. Ye, and Z. Sun, "A survey of techniques for mobile service encrypted traffic classification using deep learning," IEEE Access, 2019.
[22] H. Zhou, Y. Wang, X. Lei, and Y. Liu, "A method of improved cnn traffic classification," in Computational Intelligence and Security (CIS), 2017 13th International Conference on. IEEE, 2017, pp. 177–181.
[23] V. TONG, H. A. TRAN, S. SOUIHI, and A. MELLOUK, "A novel quic traffic classifier based on convolutional neural networks," in IEEE International Conference on Global Communications (GlobeCom), 2018, pp. 1–6.
[24] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in Proceedings of the 2nd international conference on information systems security and privacy (ICISSP), 2016, pp. 407–414.
[25] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," IEEE Access, vol. 5, pp. 18 042–18 050, 2017.
[26] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks," in Big Data (Big Data), 2017 IEEE International Conference on. IEEE, 2017, pp. 1271–1276.
[27] S. Rezaei and X. Liu, "A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning," arXiv preprint arXiv:1904.04334, 2019.
[28] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify https services," in NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2016, pp. 240–248.
[29] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014, pp. 781–789.
[30] M. Shen, M. Wei, L. Zhu, M. Wang, and F. Li, "Certificate-aware encrypted traffic classification using second-order markov chain," in 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS). IEEE, 2016, pp. 1–10.
[31] W. Pan, G. Cheng, and Y. Tang, "Wenc: Https encrypted traffic classification using weighted ensemble learning and markov chain," in 2017 IEEE Trustcom/BigDataSE/ICESS. IEEE, 2017, pp. 50–57.
[32] C. Liu, Z. Cao, G. Xiong, G. Gou, S.-M. Yiu, and L. He, "Mampf: Encrypted traffic classification based on multi-attribute markov probability fingerprints," in 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS). IEEE, 2018, pp. 1–10.
[33] M. Conti, Q. Q. Li, A. Maragno, and R. Spolaor, "The dark side (-channel) of mobile devices: A survey on network traffic analysis," IEEE Communications Surveys & Tutorials, vol. 20, no. 4, pp. 2658–2713, 2018.
[34] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: smartphone fingerprinting via application behaviour," in Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks. ACM, 2013, pp. 7–12.
[35] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in 2013 Proceedings IEEE INFOCOM. IEEE, 2013, pp. 809–817.
[36] M. V. Barbera, A. Epasto, A. Mei, V. C. Perta, and J. Stefa, "Signals from the crowd: uncovering social relationships through smartphone probes," in Proceedings of the 2013 conference on Internet measurement conference. ACM, 2013, pp. 265–276.
[37] N. Husted and S. Myers, "Mobile location tracking in metro areas: malnets and others," in Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010, pp. 85–96.
[38] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, p. 5, 2014.
[39] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," IEEE Transactions on Information Forensics and Security, vol. 11, no. 1, pp. 114–125, 2015.
[40] ——, "Can't you hear me knocking: Identification of user actions on android apps via traffic analysis," in Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. ACM, 2015, pp. 297–304.
[41] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in 2015 IEEE Conference on Communications and Network Security (CNS). IEEE, 2015, pp. 433–441.
[42] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2016, pp. 439–454.
[43] H. F. Alan and J. Kaur, "Can android applications be identified using only tcp/ip headers of their launch time traffic?" in Proceedings of the 9th ACM conference on security & privacy in wireless and mobile networks. ACM, 2016, pp. 61–66.
[44] D. Herrmann, R. Wendolsky, and 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. ACM, 2009, pp. 31–42.
[45] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006, pp. 255–263.
[46] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou, "Antmonitor: A system for monitoring from mobile devices," in Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data. ACM, 2015, pp. 15–20.
[47] D. Li, Y. Zhu, and W. Lin, "Traffic identification of mobile apps based on variational autoencoder network," in 2017 13th International Conference on Computational Intelligence and Security (CIS). IEEE, 2017, pp. 287–291.
[48] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning," in 2018 Network Traffic Measurement and Analysis Conference (TMA). IEEE, 2018, pp. 1–8.
[49] ——, "Multi-classification approaches for classifying mobile app traffic," Journal of Network and Computer Applications, vol. 103, pp. 131–145, 2018.
[50] ——, "Mimetic: Mobile encrypted traffic classification using multimodal deep learning," Computer Networks, p. 106944, 2019.
[51] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill, "Studying tls usage in android apps," in Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies. ACM, 2017, pp. 350–362.
[52] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in European conference on computer vision. Springer, 2014, pp. 818–833.

[53] 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.
[54] S. Wen and L. Itti, "Overcoming catastrophic forgetting problem by weight consolidation and long-term memory," arXiv preprint arXiv:1805.07441, 2018.
[55] J. Kampeas, A. Cohen, and O. Gurewitz, "Traffic classification based on zero-length packets," IEEE Transactions on Network and Service Management, vol. 15, no. 3, pp. 1049–1062, 2018.
[56] K. Shahbar and A. N. Zincir-Heywood, "Packet momentum for identification of anonymity networks," Journal of Cyber Security and Mobility, vol. 6, no. 1, pp. 27–56, 2017.

SHAHBAZ REZAEI (S'17) received his B.S. degree in Computer Engineering from University of Science and Culture, Tehran, Iran, in 2011 and M.S. degree from the Sharif University of Technology, Tehran, Iran, in 2013. His research interests include machine learning security, machine learning application, mobile ad hoc networks, Mathematical Modeling and performance Evaluation as well as security and architecture of Internet. He is currently a Ph.D. student at UC Davis working on application of deep learning on computer networks and also machine learning security.

BRYCE KROENCKE was born in Sacramento, California in 1997. He is currently pursuing a B.S. degree in computer science as a senior at the University of California, Davis. His involvement in professional research started in 2017 with an open-source project that involved machine learning and computational chemistry. He has completed a fellowship at Lawrence Berkeley National Laboratory, an internship at Oak Ridge National Laboratory, and is currently a student research assistant in the Engineering Department at the University of California, Davis. Bryce continues to focus his research on machine learning and its various applications.



XIN LIU (M'09–F'19) received the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 2002. She is currently a professor in the Computer Science Department, University of California, Davis, CA, USA. Before joining UC Davis, she was a postdoctoral research associate in the Coordinated Science Laboratory at UIUC. From March 2012 to July 2014, she was on leave from UC Davis and with Microsoft Research Asia. Her research interests are in the area of wireless communication networks, with a current focus on data-driven approach in networking. Dr. Liu received the Best Paper of year award of the Computer Networks Journal in 2003 for her work on opportunistic scheduling. She received the NSF CAREER award in 2005 for her research on Smart-Radio-Technology-Enabled Opportunistic Spectrum Utilization, and the Outstanding Engineering Junior Faculty Award from the College of Engineering, University of California, Davis, in 2005. She became a Chancellor's Fellow in 2011.