Received September 12, 2020, accepted October 4, 2020, date of publication October 6, 2020, date of current version October 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3029190

Automatic Mobile App Identification From Encrypted Traffic With Hybrid Neural Networks

XIN WANG[1], SHUHUI CHEN[1], AND JINSHU SU[1,2], (Senior Member, IEEE)
[1]College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China
[2]National Key Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China

Corresponding author: Xin Wang (wangxin09@nudt.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2018YFB0204301.

ABSTRACT The proliferation of handheld devices has led to an explosive growth of mobile traffic volumes on the Internet. Identifying mobile apps from network traffic has become a crucial task for mobile network management and security. Traditionally, the design of accurate identifiers relies on the deep packet inspection (DPI) techniques. However, such approaches have become less effective with the raising adoption of encrypted protocols in mobile applications (mostly TLS). To address the problem, various machine learning methods have been studied and used. Most of them use linear classifiers on top of hand-engineered features, which are unreliable due to the complexity of mobile traffic. In this article we propose App-Net, an end-to-end hybrid neural network for mobile app identification from encrypted TLS traffic. App-Net is designed by combining RNN and CNN in a parallel way and can automatically learn effective features from raw TLS flows. With coordinated fusion and optimized training, the hybrid and multimodal architecture is able to characterize both flow sequence patterns and app signatures to learn a joint flow-app embedding. We evaluate App-Net on a real-world dataset covering 80 apps. The results show that our method can achieve an excellent performance and outperform the state-of-the-art methods.

INDEX TERMS Mobile app identification, encrypted traffic classification, neural network, deep learning.

I. INTRODUCTION
In order to gain visibility and control over applications traversing the network, network operators need to identify an application by the traffic it generates. The problem of associating traffic flows with the applications that generated them is called traffic classification [1]. It is instrumental to a number of activities that are of extreme interest to carriers or enterprises, from service differentiation (e.g., policing, QoS, etc.) to security operations (e.g., firewalling, filtering, anomaly detection, etc.). With the massive adoption of handheld devices, the nature of traffic on the network has greatly changed. Traffic classification is frequently considered in mobile settings which includes identification of mobile apps from their communication traffic (APP-ID). Since the applications and services that an individual uses on a mobile device can provide valuable profiling information, APP-ID also brings about privacy implications [2]. The technique can be exploited by an adversary for malicious purposes like recognizing potentially sensitive or vulnerable apps. With the problem of APP-ID receiving increasingly attention, there has emerged considerable related works in recent years [2]–[18].

Traditionally, accurate traffic classification is performed by payload-based classifiers developed on Deep Packet Inspection (DPI) techniques [19]. The classifier is embedded in a rule-based matching algorithm deployed in a DPI engine, which associates each flow with a classification label by matching it against a database of characteristic signatures. Such DPI-based methods are popular and widely used in practice. They can apply to APP-ID as well, since most mobile apps are web applications and can leave invariable signatures in payloads that allow app identification. Numerous studies have attempted to find reliable identifiers of mobile apps [4]–[8], [20]. However, the effectiveness of their findings is weakened by the increasingly complicated mobile app features like the use of content delivery networks (CDN), the prevalence of sharing behaviors, and most importantly, the massive adoption of encrypted protocols [14].

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Khalil Afzal.

With the public awareness of the need for encryption heightened, there has been a push for deployment of encrypted protocols on mobile devices to secure sensitive information. Both Android and Apple platforms have adopted policies that encourage encryption (TLS especially) and prevent insecure network connections [21], [22]. Hence we can find that encryption has been used by more and more mobile services and applications: recent estimates suggest 80% of Android apps are encrypting traffic with TLS by default, a proportion expected to increase to 90% for apps targeting Android 9 and higher [21]. The increasing prevalence of encryption on the mobile network has necessitated a paradigm shift in the way we analyze mobile traffic. DPI-based methods, on which the prevailing traffic classification systems heavily rely, are of little use when payloads are encrypted.

For a long time, machine learning (ML) methods have become the mainstream technique for encrypted traffic classification in academia [23]. Such methods exploit a variety of flow-level measurements [24] to characterize the traffic of different applications in particular ML algorithms, which usually involves feature engineering and model training. Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models. While vital to the performance of the models, it is costly with the requirements of careful engineering and considerable domain expertise. Feature engineering is even more important in the context of mobile networks, as mobile traffic is usually fast-evolving, noise-prone and exhibits non-trivial spatial/temporal patterns [25]. Although there have been several successful ML-based methods for APP-ID [9]–[11], [26], they are inseparable from refined hand-engineered features which are hard to obtain, unstable and prone to obsolescence.

In recent years, deep artificial neural networks (ANNs), also known as deep learning (DL), have shown remarkable power in a wide spectrum of domains [27]. Networking researchers are also beginning to explore its potential to solve the traffic classification problem. The key advantage of DL over traditional ML is the ability to learn good features automatically from raw data without feature engineering. Given a lot of labeled data, DL models can be designed end-to-end (E2E) for better performance, which replaces the hand-engineered pipeline with a single learning algorithm so that it goes directly from the input to the desired output. Two mainstream architectures in DL, convolutional neural network (CNN) and recurrent neural network (RNN), have been exploited in the task of traffic classification [12], [13], [28]–[32]. RNN, particularly the long short-term memory (LSTM) network, works well on time series data with long-term dependencies, while CNN excels at extracting location-independent spatial structure from raw data.

In this article, we focus on mobile app identification from encrypted TLS traffic with DL approaches. Due to the rich characteristics of TLS protocol, multiple statistical or payload fingerprints that allowing APP-ID can be derived from TLS sessions. For example, the packet length from a TLS session naturally forms sequences which hold distinctive information of a mobile app, such as message types and certificate length. Besides, the payload data during the initial TLS handshake usually contains plaintext fields like cipher suites and extensions, which can also help to fingerprint an app service. Previously proposed methods mainly center on designing different models to take advantage of TLS data from one side [32]–[36]. Therefore, it is highly desired to develop a systematic way to model more aspects of TLS data from raw traffic, which we believe advanced multimodal and hybrid DL architectures would be suitable.

To this end, we propose an end-to-end hybrid neural network, referred to as App-Net. By combining recurrent and convolutional neural networks together, it can learn different aspects of the TLS data in a multimodal way and identify mobile app from raw traffic more effectively. In particular, App-Net consists of an LSTM recurrent neural network to learn effective features from raw flow sequences, a convolutional neural network to extract byte signatures from the initial TLS packet payload, and a feature fusion stage to learn coordinated and joint representations to take advantage of both features drawn from RNN and CNN. Such representations are used for APP-ID in the end. The experimental analysis on real world dataset demonstrates that App-Net outperforms separate RNN or CNN as well as a variety of state-of-the-art methods in term of classification accuracy and F1 score.

The rest of the paper is organized as follows. Section II reviews the related literature from two aspects. Section III presents the preliminaries of this work and Section IV describes the proposed App-Net framework in detail. The experimental results are reported in Section V and the paper is concluded in Section VI.

II. RELATED WORK

There has been a plethora of work in the field of traffic classification. Here, we first give a review of recent achievements focused on APP-ID. Furthermore, those efforts proposing DL-based methods for traffic classification are discussed in particular.

A. TRAFFIC CLASSIFICATION FOR APP-ID

The research on APP-ID mainly falls into two categories under different assumptions: (1) unencrypted traffic, (2) encrypted traffic. For the first case, researchers don't take encrypted traffic into consideration by default and basically focus on how to extract or generate reliable identifiers of mobile apps. With such identifiers, APP-ID can be performed using DPI-based methods. Most of the identifier discovery techniques are based upon direct analysis of app traffic, assuming that the mobile traces are already given or generated from automatic UI exploration techniques [4]–[6], [20]. Some others collect app packages for static analysis to guide the identifier discovery, enabling APP-ID in a massive scale [7], [8]. However, all above cases do not deal with encrypted traffic about which the second category of research cares most. Our work falls into this category.

Previous works dealing with encrypted mobile traffic mostly adopt ML techniques. Wang *et al.* [9] propose a system to identify mobile apps from encrypted wireless traffic. They collect data from 13 arbitrarily chosen apps by running them dynamically and training a Random Forest (RF) classifier with features extracted from the 802.11 frames. Similar to this work, Taylor *et al.* propose AppScanner to fingerprint and identify smartphone apps from their encrypted network traffic [2]. They further attempt to improve the framework by introducing an approach to separate ambiguous traffic ahead at the cost of reducing classified flows [10]. For evaluation, they build datasets from a much larger app set that covers 110 apps and use a demultiplexing technique to obtain accurate ground truth. With 54 refined statistical features, the trained RF classifier can have a best accuracy of 73.7%, which still exhibits non-negligible performance degradation compared to the average in traditional traffic classification. Alan and Kaur [17] investigate whether Android apps can be identified from their launch time network traffic using only TCP/IP headers. They find that popular Android apps can be identified with 88% accuracy by using the packet sizes of the first 64 packets generated on the same device. Aceto *et al.* [11] propose a multi-classification approach of intelligently combining outputs from state-of-the-art classifiers to improve the performance of APP-ID. The performance can be improved according to all considered metrics, up to +9.5% recall score with respect to the best base classifier. Recently, Aceto *et al.* [12] investigate a variety of DL-based traffic classification architectures and try to apply them for APP-ID. They believe DL may be the steppingstone toward high-performing traffic classification in the dynamic and challenging mobile context. They further propose and validate a general framework for DL-based APP-ID [16] and introduce a novel multimodal DL framework which is able to capitalize traffic data heterogeneity [15]. This new framework provides an implementation of combining CNN and RNN together which is like our work. However, it pays little attention to the modality fusion strategy and the two-stage training process is costly.

On the whole, the state-of-the-art solutions on APP-ID considering encrypted traffic have changed from traditional ML techniques with various handcrafted features to the applications of DL methods. As an app's real and comprehensive network traces are hard to come by, simulators with user interface (UI) fuzzing is often used for traffic collection [2], [10]–[12], [17], [26]. Even if real world dataset of mobile apps can be achieved [4], acquisition of the ground truth remains a difficult problem. Except for only a few works [37], [38], the mobile traces are usually labeled by approaches of unknown reliability.

B. DL-BASED METHODS IN TRAFFIC CLASSIFICATION

Recently, as much larger and much deeper neural networks have shown impressive capability across a range of difficult problem domains, researchers begin to apply deep learning on traffic classification. The first successful attempt is introduced in [28] by a security engineer. It focuses on Multilayer Perceptron (MLP) and Stacked Autoencoder (SAE) with raw traffic as input. With the real world dataset collected from enterprise network he shows that the deep learning approach works well on the applications of feature learning, protocol classification and anomalous protocol detection. Wang *et al.* [29] propose an end-to-end method of encrypted traffic classification with one-dimensional convolution neural networks (1D-CNN). They evaluate the model on the public ISCX VPN-nonVPN traffic dataset and show that it can achieve outstanding performance on both non-VPN and VPN traffic, about 10% higher than the state-of-the-art method of C4.5 decision tree in precision and recall. The same dataset is also used in [31] where a framework embedding SAE and CNN is proposed to classify network traffic. Differently, it keeps the IP header and the first 1480 bytes of each IP packet as input and perform classification at packet level. The framework achieves F1 score of 0.95 in application identification task and 0.97 in traffic characterization task. In addition to CNN, RNN is also introduced for traffic classification in [30]. It presents a technique based on both CNN and RNN that can be used for Internet of Things (IoT) traffic. In order to get better results, the authors try different set of features which include header and statistic information other than payloads. It is shown that port numbers play an important role in IoT traffic classification which is intuitive as many services keep their assigned ports. In Liu's work [32], an encrypted traffic classifier with RNN is proposed and further improved by a multi-layer encoder-decoder structure which can enhance the effectiveness of features. It outperforms those state-of-the-art Markov models [33]–[35] in experiments on a real world TLS dataset covering 18 applications. The DL practices in traffic classification mainly focus on private or public datasets which rarely include mobile traffic. Only recently did researchers design mobile traffic classifiers via the adoption of DL methods [12]. Using self-collected mobile datasets, Aceto *et al.* thoroughly evaluate all previously proposed DL architectures (SAE, CNN, RNN, etc.) with various inputs and find there is no "killer" architecture for APP-ID.

III. PRELIMINARIES

In this section we elaborate on some preliminary knowledge of our framework. First we discuss the the traffic classification object we used and give the problem definition of APP-ID. Then we introduce our way of labeling data with ground truth. Lastly, we briefly describe the constituent components used in development of App-Net, namely LSTM and CNN, respectively.

A. FLOW AND PROBLEM DEFINITION

A flow is a set of packets transmitted between two host addresses using a particular protocol, and where appropriate a particular pair of ports. The packets in a flow all share common characteristics which are known as the conventional network five-tuple: the source and destination addresses, the source and destination ports (for TCP and UDP traffic) and the protocol number. While a flow is unidirectional, interactive network sessions usually involve bidirectional flows which are either half- or full-duplex. A bidirectional flow, also called a biflow, consists of a pair of unidirectional flows whose source and destination addresses and ports are reversed, and whose time spans overlap. In the context of APP-ID, a biflow is the largest discrete traffic unit we can use for classification, which requires the least effort to classify the most traffic. Thus, we take it as our traffic classification object.

In mobile network monitoring, it is undesirable to wait for a long duration flow to finish. To avoid this indefinite wait for flow data, we extract flows with a timeout approach: a flow is created whenever it is inactive for an *inactive_timeout* period, or whenever it is active for longer than an *active_timeout* period. Here we use 10 seconds for *inactive_timeout* and 30 seconds for *active_timeout*.

With the flow definition we come to the problem definition in this work. We follow the majority of prior works and formulate encrypted traffic classification for APP-ID as a multiclass classification problem which only considers mobile TLS traffic. The task is to perform a supervised multiclass classification to find a function that maps a TLS biflow to a mobile app that generated it. More formally, suppose a biflow f that was created by an app a is an instance of the form (f, a). Given a set of observed biflows from the network: $S = \{(f_i, a_j) : f_i \in F, a_j \in A\}$ where F is the set of unlabeled biflows and A is the set of all possible apps that generate them, the problem is to find a function $g : F \rightarrow A$, such that each unlabeled biflow $f_s \in F$ can be mapped to an app $a_t \in A$, satisfying as much as possible that $(f_s, a_t) \in S$.

B. GROUND TRUTH ACQUISITION

Most previously proposed APP-ID solutions report performance with results based on private datasets which are labeled by approaches of unknown reliability [4]–[6], [10]–[12], [20]. In order to obtain the ground truth for real network traffic, the most common way is running apps one by one separately and manually label the trace [11]. It is, however, not trusted due to the noise traffic generated by system or background apps. Though some public tools may aid the process of labeling, they are hard to use with the need for root access [10]. Recently, two frameworks have been proposed to tackle the problem of mobile traffic collecting and labeling. Mobilegt [37] collects traffic trace and builds ground truth with a remote VPN server which is connected to all monitored devices, while MIRAGE [38] builds its capture system on top of wire-connected and rooted devices. With the drawbacks of VPN server as well as rooted phones, however, the scalability of these solutions are restricted. As their accurately labeled datasets are shared without raw traffic trace, we did not take them into consideration.

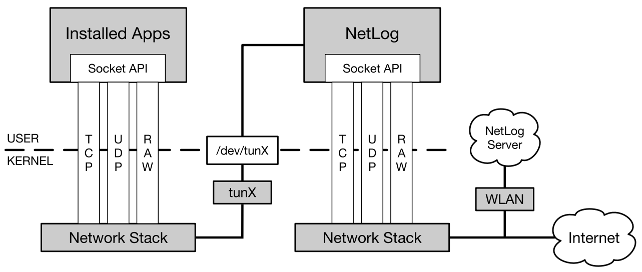


FIGURE 1. The structure of NetLog.

In comparison with the mentioned solutions, we have also overcome the obstacle of ground truth acquisition by only developing and leveraging an Android app. This tool named NetLog can help to collect smartphone's traffic with accurate app labels. It works by leveraging the Android VPN permission to capture and analyze network flows locally on the device and in user-space. Functioning as a local VPN server on the phone, it becomes a middleware between all apps and the network interface, as shown in Fig. 1. NetLog can create logs for each app's TCP/UDP communications from internal app-UID mappings and export all raw traffic that generated within a period into *pcap* files. The exported files and the corresponding logs will be uploaded to a cloud server where further analysis can be performed, including automatically associating each flow with its generating app.

C. LONG SHORT-TERM MEMORY NETWORK

A recurrent neural network (RNN) is different from the standard feedforward neural network architecture that it has feedback connections, which makes it particularly suited for modeling sequential phenomena. In practice however, learning long-range dependencies with a vanilla RNN is difficult due to vanishing/exploding gradients [39]. Long short-term memory networks (LSTMs) are explicitly designed to avoid the long-term dependency problem. This is achieved by introducing a memory state and multiple gating functions, that provide a memory-based architecture to control the write, read, and removal (forget) of the information written on the memory state [40]. LSTMs allow deep networks with multiple layers to be created, which is often crucial for obtaining competitive performance on various tasks.

Considering that the standard LSTM is a biased model, where later inputs are more dominant than earlier inputs, we use bidirectional LSTMs for robustness in this article. A bidirectional model simply put two independent LSTMs together: one reading the input from left to right and one reading it from right to left. The outputs of the two networks are usually concatenated, allowing the networks to have both backward and forward information about the sequence at every time step, thus mitigating the bias. In App-Net, the bi-LSTM model takes sequences of packet length in TLS biflows as the input and learns how to model the sequences with respect to the target app. The sequences are considered to be of fixed length and the structure of the incorporated bi-LSTM is sequence-to-vector as shown in Fig. 2.

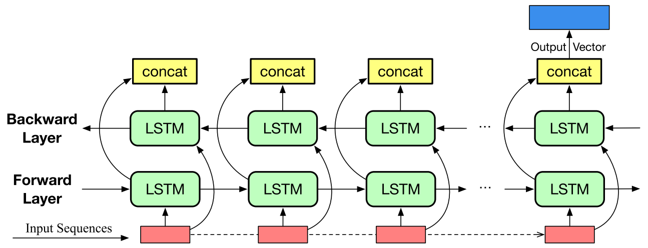


FIGURE 2. Sequence-to-vector bi-LSTM network: Input sequences are in red, output vector is in blue.

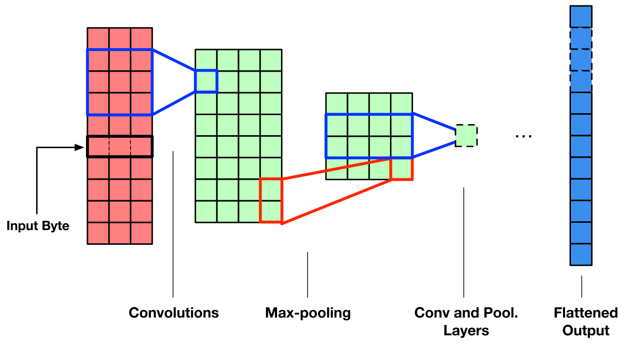


FIGURE 3. 1D CNN: Input bytes are in red, output vector is in blue. (The number of squares does not represent the actual number).

D. CONVOLUTIONAL NEURAL NETWORK

A Convolutional Neural Network (CNN) is quite similar to an ordinary neural network. It is specifically designed to process data that come in the form of multiple arrays. There are four key ideas behind CNN that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers [27]. The most important building block of a CNN is the convolutional layer. Each neuron in the convolutional layer is connected only to neurons located within a small rectangle in the preceding layer. This allows the network to concentrate on low-level features and assemble them into higher-level features in the next layer. The pooling layer is used by a CNN to subsample the input in order to reduce the computational load and the number of parameters, thereby limiting the risk of overfitting. Typical CNN architectures stack a whole series of convolution and pooling layers, and the resulting outputs need to be flattened and concluded by at least one regular fully connected layer prior to classification. Since many data modalities are in the form of multiple arrays: 1D for signals and sequences including language, 2D for images or audio spectrograms, and 3D for video or volumetric images, CNNs have been applied in a variety of tasks and achieve superhuman performance. Taking network traffic as byte sequences, it is intuitive to fit a 1D CNN model to it for classification.

The signatures of an app, which appear in uncertain positions within the packet payloads, are generally meaningful plaintext or codes made up of certain byte sequences. This makes 1D CNN, whose default behavior includes learning location-independent spatial structure of the input, an ideal model to extract effective features from the traffic byte sequences. Unlike the densely-connected neural network layer that needs 1D data as input, a 1D CNN accepts 2D matrices. Therefore, each byte in our input data can be encoded into vectors of fixed size to feed the model. The 1D CNN architecture in App-Net is illustrated in Fig. 3.

IV. THE APP-NET FRAMEWORK

The idea of App-Net is to combine LSTM and CNN to utilize their representation abilities on different aspects of the TLS data and to learn a joint feature used for APP-ID. In general, App-Net consists of two parallel paths followed by a fully connected multilayer fusion neural network as illustrated in Fig. 4.

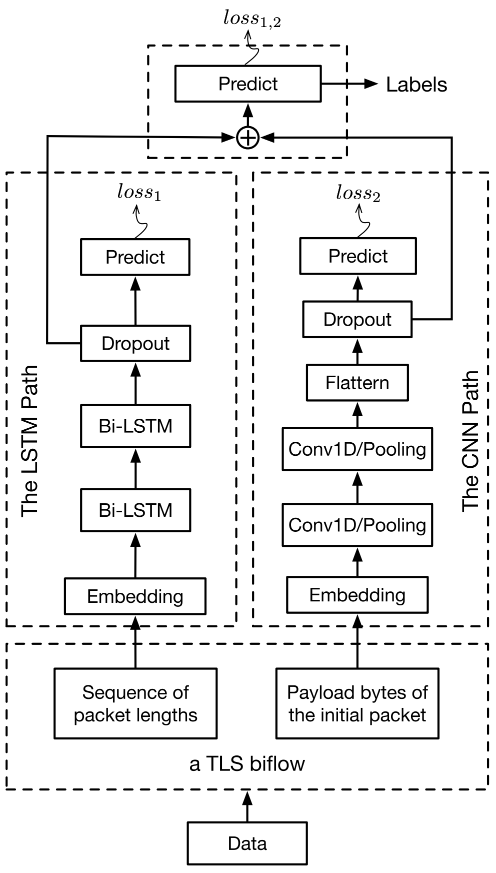


FIGURE 4. The proposed App-Net framework. "Predict" layers are fully-connected layers followed by softmax activation and $\oplus$ is the weighted element-wise addition.

To formulate the problem, we define the input of LSTM as a sequence of packet lengths $L = \{l_1, \ldots, l_T\}$, where the integer-valued l_t represents the length of a packet in bytes and T represents the total number of packets taken into account. We define the input of CNN as a sequence of bytes $B = \{b_1, \ldots, b_M\}$, where each byte b_m is represented in decimal and M represents the maximum number of bytes fit into the CNN model. Technically, App-Net is designed to learn a predictive function $g = f(R(L), C(B))$. $R(L)$ is derived by training the LSTM over the sequence L to learn useful statistical features, while $C(B)$ corresponds to high-level features extracted by CNN from sets of payload bytes in B. The features respectively learned by LSTM and CNN (i.e., $R(L)$ and $C(B)$) convey complementary information pertaining to the traffic varying. Then, the feature fusion layer merges them to classify the traffic into different apps. Finally, the app prediction is realized by the function f, which corresponds to the feature fusion and output layers.

A. THE LSTM PATH

Neural networks generally work with real numbers from the compact interval $[-1, 1]$ due to the nature of the mathematical operations they perform. Thus, we need to map each integer-valued element of sequence L to a corresponding vector of real numbers first. So that the LSTM network can ingest and process them. The transformation can be performed by adding an embedding layer at the very beginning. After embedding, each input sequence L can be viewed as a matrix $\boldsymbol{M}_L \in \mathbb{R}^{T\times d}$, where d represents the dimension of the embedding vectors.

During the training phase, for a given timestep t, the mini-batch input is $\boldsymbol{L}_t \in \mathbb{R}^{n\times d}$, where n is the number of sequence examples. In the bi-LSTM architecture, we assume that the forward and backward hidden states for this timestep are $\overrightarrow{\boldsymbol{H}}_t \in \mathbb{R}^{n\times h}$ and $\overleftarrow{\boldsymbol{H}}_t \in \mathbb{R}^{n\times h}$ respectively. Here, h indicates the number of hidden units. We compute the forward and backward hidden state updates as follows:

$$\overrightarrow{\boldsymbol{H}}_t = \phi(\boldsymbol{L}_t\overrightarrow{\boldsymbol{W}}_{lh} + \overrightarrow{\boldsymbol{H}}_{t-1}\overrightarrow{\boldsymbol{W}}_{hh} + \overrightarrow{\boldsymbol{b}}_h) \quad (1)$$

$$\overleftarrow{\boldsymbol{H}}_t = \phi(\boldsymbol{L}_t\overleftarrow{\boldsymbol{W}}_{lh} + \overleftarrow{\boldsymbol{H}}_{t+1}\overleftarrow{\boldsymbol{W}}_{hh} + \overleftarrow{\boldsymbol{b}}_h) \quad (2)$$

Here, ϕ is the hidden layer activation function. The weight parameters $\overrightarrow{\boldsymbol{W}}_{lh}$, $\overrightarrow{\boldsymbol{W}}_{hh}$, $\overleftarrow{\boldsymbol{W}}_{lh}$ and $\overleftarrow{\boldsymbol{W}}_{hh}$, and bias parameters $\overrightarrow{\boldsymbol{b}}_h$ and $\overleftarrow{\boldsymbol{b}}_h$ are all model parameters. Then, the forward and backward hidden states $\overrightarrow{\boldsymbol{H}}_t$ and $\overleftarrow{\boldsymbol{H}}_t$ are concatenated to form the hidden state $\boldsymbol{H}_t \in \mathbb{R}^{n\times 2d}$.

To further improve the learning capability of the LSTM network, we stack two layers of bi-LSTM in App-Net. The hidden state $\boldsymbol{H}_t$ of the first bidirectional layer is passed on as input to the second bidirectional layer. Finally, the output layer computes the output $\boldsymbol{O}_t$ with $\boldsymbol{H}_t$ of the second layer:

$$\boldsymbol{O}_t = \boldsymbol{H}_t\boldsymbol{W}_{hq} + \boldsymbol{b}_q \quad (3)$$

where the weight parameter $\boldsymbol{W}_{hq}$ and the bias parameter $\boldsymbol{b}_q$ are the model parameters of the output layer, and q is the number of outputs. As our bi-LSTM architecture is a sequence-to-vector RNN model, we use the final output vector $\boldsymbol{O}_T$ as the output of the LSTM path.

B. THE CNN PATH

While L, the sequence of packet length in a TLS biflow, is passed to the LSTM network as input, the payload bytes of the initial data packet in the same TLS biflow, B, will be fed into the CNN to process at the same time. To feed the sequence of bytes into 1D CNN, we also need to add an embedding layer to transform the bytes into vectors of numbers. The embedding is done by prescribing a vocabulary of size 256 for the input sequence, corresponding to all possible bytes, and then quantize each byte using one-hot encoding. In this way, the sequence of bytes is transformed into a sequence of such 256 sized vectors with fixed length M. Any byte exceeding length M is ignored, and any vacancy within length M are quantized as all-zero vectors. After embedding, each input sequence B is encoded into a matrix $\boldsymbol{M}_B \in \mathbb{R}^{M\times 256}$.

In a convolutional layer, an input array and a correlation kernel array are combined to produce an output array through a cross-correlation operation usually, which is the same as convolution but without flipping the kernel [41]. Suppose we have a discrete input function $I\colon \mathbb{N} \to \mathbb{R}$ for one of the convolutional layers in 1D CNN. Then the multiple one-dimensional cross-correlation operations $Conv1D(x)$ using a given discrete kernel function K can be defined as:

$$\begin{aligned} Conv1D(x) &= (K * I)(x) \\ &= \sum_{m=1}^{L_C} K(m) \cdot I(S_C \cdot x + m - S_C) \end{aligned} \quad (4)$$

where the count of operations x is equivalent to the count of steps when sliding the 1D convolution window (i.e., kernel or filter) through the input data, and L_C and S_C represent the length and stride of the window respectively. The convolutional layer is typically parameterized with weights by a set of such kernel functions K_i and each corresponding output $H_i(x)$ is referred to as a feature map.

Let $\boldsymbol{M}$ be a matrix and let $\boldsymbol{M}[i]$ denote the vector corresponding to the i-th row of $\boldsymbol{M}$. Specifically in our case, we can rewrite equation-(4) for the first convolutional layer which is right after the embedding layer:

$$Conv1D(x) = \sum_{m=1}^{L_C} \boldsymbol{K}[m]\boldsymbol{M_B}[S_C \cdot x + m - S_C]^{\top} \quad (5)$$

where $\boldsymbol{K} \in \mathbb{R}^{L_C\times 256}$ is the weight matrix of one kernel and $\boldsymbol{M}_B$ is the encoded input matrix.

A pooling layer is often used after a convolutional layer in order to reduce the complexity of the output and prevent overfitting of the data. The 1-D spatial maxpooling used in our case is defined as:

$$MaxPool1D(x) = \max_{m=1}^{L_P} I(L_P \cdot x + m - L_P) \quad (6)$$

where L_P represents the length of the maxpooling window.

The 1D CNN in App-Net consists of two stacked layers of 1-D convolutional, activation and pooling operations. Each layer has a specified number of kernels of a specified kernel size. Each kernel on a layer sweeps through the entire input to extract local features. The final output of the CNN path in App-Net is the flattened maxpooling outputs on the last layer.

C. THE FUSION AND TRAINING

Because the LSTM and CNN representations are from two different modalities, merging them together is a crucial task.

To capture the correlation across modalities for APP-ID, an intuitive way is to directly concatenate the different features of them, then employ a few shared representation layers to generate the high-level joint representations [15]. While such fusion is simple to implement, it increases the dimensionality and lacks the ability in capturing more complex correlation across modalities [42]. To maximize the correlation between different representations, coordinated representations [43] can be learned with some constraints to force the representations to be more complementary.

In the fusion stage of App-Net, the fusion operation for representations of LSTM and CNN, i.e., $R(\boldsymbol{L})$ and $C(\boldsymbol{B})$, is done using a learned weighted element-wise addition. It is defined as follows:

$$Fusion = w_r R(\boldsymbol{L}) + w_c C(\boldsymbol{B}) \tag{7}$$

where $w_r, w_c \in (0, 1)$ are trainable weights and $w_r + w_t = 1$. $R(\boldsymbol{L})$ and $C(\boldsymbol{B})$ are projected with zero padding to have the same size. Such fusion features are processed by a FC layer with softmax activation for the final prediction.

In the "Predict" layer of the App-Net framework, the softmax activation returns a vector which can be interpreted as estimated conditional probabilities of each class given the input. Comparing the estimates with reality by checking how probable the actual classes are, we can get the softmax loss function which is called the cross-entropy loss:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}^i) = -\sum_{j=1}^{N} (y_j \log \hat{y}_j^i) \tag{8}$$

where $\boldsymbol{y}$ are the target ground truth and $\hat{\boldsymbol{y}}^i$ are the predicted probabilities inferred by network i for each class in N. The network gives a high probability for each predicted class which results into a minimized entropy.

When training the whole hybrid network, it is important to maintain the performance or the representation quality of the unimodal network. Thus the global loss of App-Net is a combination of the unimodal LSTM and CNN network losses, which is defined as follows:

$$L = L_{1,2} + \alpha_1 L_1 + \alpha_2 L_2 \tag{9}$$

where $L_{1,2}$ is the loss computed from the fusion layer, L_1 and L_2 are the unimodal losses from the LSTM and CNN network respectively. The weight parameters α_1 and α_2 are set to 1 here by cross validation.

As we can see, the loss function is differentiable, and the architecture of App-Net allows the gradient to be backpropagated to both LSTM and CNN parts. In the training phase, all trainable parameters of App-Net including the fusion weights w_r, w_c are optimized together by applying a stochastic gradient descent.

V. EVALUATION

In this section, we describe our dataset and experiment settings and evaluate the performance of the proposed App-Net framework to demonstrate its advantage by comparing with a variety of state-of-the-art methods.

A. EXPERIMENT SETTINGS

1) DATASET

The dataset in this work has been constructed from real mobile traffic which is collected using NetLog by human users. Different from other labeled datasets whose ground truth is not always trustworthy, our dataset is accurately labeled with NetLog capable of retrieving the reliable ground truth, as described in Section III.

The original Android traces have been captured during May.2019 - Jul.2019, generated by 3 off-campus users in daily lives with different devices. The three smartphones (Xiaomi Note 3, Honor 9 and Huawei P20) are all running systems based on Android 9. After biflow segmentation and TLS filtering, we accumulate about 189k labeled TLS biflows for the top 80 apps with each app having more than 1000 TLS biflows. Table 1 reveals more details. The set of 80 apps, which covers a variety of categories, have exhibited non-negligible class imbalance due to different app service and usage. To feed into the evaluation models, we further extract some basic statistical and sequential information from each TLS biflow and also have the payload information of the initial data packet. These data together have made up our dataset in this article.

To look deep into our dataset, we plot summary statistics for the two inputs of App-Net using empirical cumulative distribution functions (ECDF). The two plots in Fig. 5 indicate two feature distributions across the dataset. From the left one we can see, almost all biflows in the dataset have an initial TLS data packet (i.e., ClientHello message) with payload length no larger than 517. The length of exact 517 bytes has constituted a great proportion because of the ClientHello padding extension [44]. On the other hand, a majority of TLS biflows have no more than 20 messages, which suggests the boundary of the input sequence length.

2) EVALUATION METRICS

Previous work mainly uses *accuracy* (defined as the number of correctly labeled examples divided by the total number of examples) as the metric for classification performance. However, when classes are unbalanced, *accuracy* is not so valid and the use of more sophisticated metrics becomes necessary. This would be the case in practice, since relatively few apps are responsible for the majority of mobile traffic. In this article, for the sake of comparison, we follow the same philosophy and use the *Top-K accuracy* in our multiclass experiments. It allows the classifier to consider the top K predicted apps which evaluates the soft-output of the model.

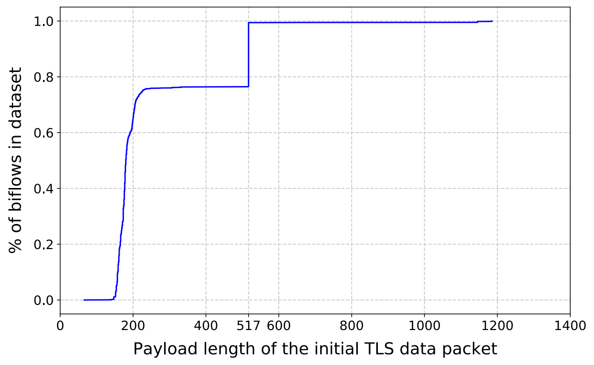
For imbalanced classification here, we turn to the metric group of precision-recall which focuses on per class. The precision and recall for a class can be calculated as follows:

$$Precision = TP/(TP + FP) \tag{10}$$

$$Recall = TP/(TP + FN) \tag{11}$$

TABLE 1. Details of our dataset.

App Category	App Name	Biflows	App Category	App Name	Biflows	App Category	App Name	Biflows
Communication	QQ	1222	Maps&Navigation	Amap	1418	Shopping	Suning	4166
	Wechat	4271		BaiduMap	2852		Taobao	9340
Entertainment	Bilibili	11697		Dida	1586		Tmall	2327
	Changba	1124		Didi	3653		Vipshop	1362
	Douyu	1691	Music&Audio	163Music	1550		Xianyu	5019
	Egame	1657		Lizhi	1234	Social	Renren	1562
	Fengxing	1390		Migu	1846		Kuaishou	1300
	Haokan	1648		QQMusic	1060		Momo	1971
	Huoshan	1493		Xiami	1843		QQZone	1362
	Huya	1119		Ximalaya	1264		Rugu	1293
	iQiyi	1105	News&Magazines	163News	1599		Soul	1673
	Kg	1137		iFeng	1516		Tantan	1305
	Mgtv	1076		Kuaibao	1919		Tianya	1622
	PPVideo	1288		QQNews	1212		Tieba	1366
	SohuVideo	1504		SinaNews	1402		Douyin	3341
	Xigua	1596		SohuNews	1453		Weibo	1840
	Youku	1822		Toutiao	2180		Weishi	1198
	QQVideo	1606		Zhihu	2467		Xhs	1296
Finance	Alipay	1094	Productivity	BaiduDisk	1954	Tools	163Mail	1332
	EastMoney	1040		Dingding	1233		Baidu	4961
	Hexin	1809		Tim	1560		QQBrowser	2615
	Jdjr	1763	Shopping	Amazon	1633		QQMail	1543
Lifestyle	Dianping	4853		Eleme	11941		UCBrowser	3311
	Douban	1501		JD	11106	Travel&Local	Ctrip	1361
	Koubei	2652		Jddj	2841		Feizhu	4477
	Meituan	1425		Juhuasuan	1267		Qunar	4189
	Taopiao	1278		Pdd	1887			



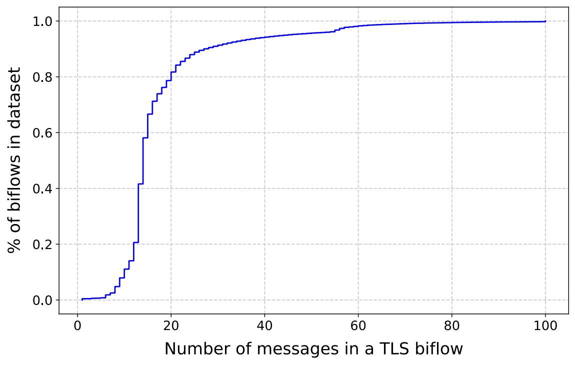


FIGURE 5. ECDF for two features of the input data.

where *TP* refers to the number of true positives (those examples correctly identified as belonging to the class), *FP* refers to the number of false positives (those examples incorrectly identified as belonging to the class), *FN* refers to the number of false negatives (those examples incorrectly identified as not belonging to the class) and *TN* refers to the number of true negatives (those examples correctly identified as not belonging to the class). The precision is intuitively the ability of the classifier not to label as positive example that is negative. The recall (also called sensitivity or true positive rate) quantifies the ability of the classifier to find all the positive samples or to avoid false negatives. For different backgrounds of APP-ID in practice, the effects of false positives and false negatives can be both important. Thus we calculate both the metrics as well as the popular combined score - F1 score. F1 score (also called F-score or F-measure) is the harmonic mean of the precision and recall, which conveys the balance between them.

For simplicity in evaluation, we calculate metrics for each class and only report the macro-averaged values over all classes. We prefer macro average in unbalanced settings because it calculate the unweighted mean and has equal emphasis on all classes [45]. When only one metric is considered for evaluation, we use F1 score by default.

3) MODEL SETTINGS AND TRAINING

The tuning of hyperparameters is fundamental in supervised classification. For deep learning architectures, however, the large amount of training data and the large number of hyperparameters have rendered an exhaustive search prohibitive in terms of computational resources. We thus aim at a good-enough classifier and make App-Net to perform well with selected tuning.

For the model settings, the bi-LSTM network has two stacked layers of LSTM cells with the hidden state size set

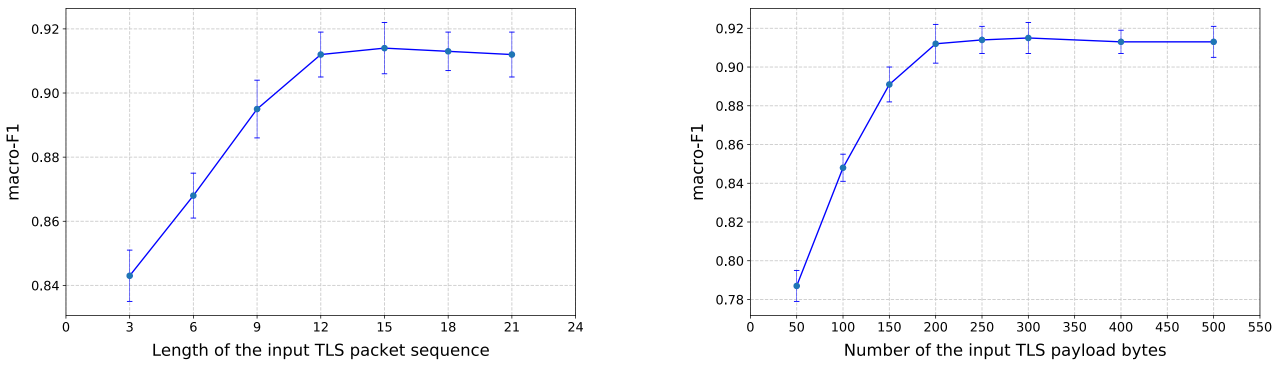


FIGURE 6. Results of App-Net with different size of input for LSTM (left) and CNN (right).

to 128. The 1D CNN has two stacked convolutional and max pooling layers with 128 filters of size 15 and a pool size of 3. Each of the dimensional parameters is selected from the set {32, 64, 128, 256, 512, 768, 1024} to achieve the best performance. The algorithm used in training App-Net is stochastic gradient descent (SGD) with a minibatch of size 128, using momentum 0.9 and initial learning rate 0.1 which is halved every 3 epochs.

During the training of App-Net, a dropout rate of 0.5 have been used in the "Dropout" layer to reduce overfitting. With dropout regularization [46], at each training iteration a random subset of all neurons in the layer are "dropped out" and output 0, which can loosen the model and allow for greater generalization. With the number of epochs set to 20, an early stopping with patience 3 is also employed to prevent overfitting. It is a simple and efficient regularization technique that works by stopping the training after the validation error reaches the minimum. Considering the imbalance of the dataset, our model is also trained with class weights. We want to have the classifier heavily weight the minority classes so that App-Net can pay more attention to traffic from an under-represented app.

4) INPUT ANALYSIS

As only informative data is required to train a good DL model, it is common to limit the size of input data to reduce the training complexity. To analyze App-Net's requirements of the input data, we run a series of tests and give the results in Fig. 6.

From the dataset summary statistics in Fig. 5 we conclude that the inputs of App-Net are within a certain range. Then the optimal length of input can be found exactly by varying the input of either LSTM or CNN. For the LSTM part, the input sequence length is required to be at least 12 to support the best result. For the CNN part on the other hand, the top 200 payload bytes are well enough to offer the best feature representation. After all, this range covers most of the valuable TLS extensions in the ClientHello message. Therefore, we have set the shapes of the two inputs of App-Net to 12 and 200 respectively, which can achieve the best performance with the least effort.

TABLE 2. Experiment results of comparison methods on the dataset of 80 apps.

Model	Top-1	Top-5	Precision	Recall	F1 score
RF	62.5 ± 0.6	89.2 ± 0.3	60.8 ± 0.7	47.9 ± 0.8	50.6 ± 1.2
CNN-784	86.7 ± 0.7	97.2 ± 0.3	86.4 ± 0.6	83.5 ± 0.6	84.2 ± 0.7
FS-Net	85.1 ± 0.9	96.6 ± 0.4	82.2 ± 0.6	80.1 ± 0.7	81.7 ± 0.9
MIMETIC	91.0 ± 0.5	98.4 ± 0.3	89.9 ± 0.5	88.2 ± 0.6	88.9 ± 0.7
App-CNN	87.1 ± 0.4	97.7 ± 0.2	86.5 ± 0.6	84.4 ± 0.5	85.2 ± 0.8
App-LSTM	84.3 ± 0.4	96.3 ± 0.3	81.4 ± 0.5	78.2 ± 0.6	78.9 ± 0.7
App-Net	93.2 ± 0.5	98.9 ± 0.2	92.0 ± 0.4	90.8 ± 0.6	91.2 ± 0.8

B. EXPERIMENT RESULTS AND ANALYSIS

1) COMPARATIVE EVALUATION

We compare App-Net with the following five different methods for evaluation:

- **RF**. This method represents the state-of-art mobile traffic classifier using traditional machine learning technique of random forest [2]. The statistical features it uses are derived from three packet length sequences (incoming, outgoing and bidirectional) within a biflow. For each of the sequences, 18 statistical values are computed. Then a feature selection is performed to have the top 40 values as the input.
- **CNN-784**. This method is evaluated to be the best-performing unimodal DL architecture for APP-ID in recent work [12], where 784 bytes from the application layer are chosen to be the input. The input data is treated as traffic images of size of $28 * 28 * 1$ and converted to IDX files before feeding into the model of 2D-CNN [47].
- **FS-Net**. This method represents a flow sequence network that combines stacked bidirectional GRUs with autoencoders to learn features from packet sequences in raw TLS flows [32]. It achieves the best results in terms of TLS traffic classification comparing to other state-of-the-art methods.
- **MIMETIC**. This method is represented as the first multimodal DL framework for APP-ID which includes a 1D CNN and a GRU network [15]. The inputs for the CNN and GRU network are first 576 bytes of L7 payload and multiple protocol fields of first 12 packets respectively. Regardless of the implementation details, the overall

architecture is similar to App-Net except that it uses concatenation for feature fusion which is quite simple and two-stage training phase which is costly. Besides, it is not optimized for TLS traffic and consumes much more input data than App-Net.

- **App-CNN**. This is a variant of App-Net which abandons the LSTM path. It only uses the 1D CNN to learn from payload bytes of the initial data packet for APP-ID.
- **App-LSTM**. This is another variant of App-Net which abandons the CNN path. It only uses the bi-LSTM network to learn the sequences of packet length for APP-ID.

In Table 2, we report the comparative evaluation results of App-Net on our dataset, which includes the Top-K accuracy ($K \in \{1, 5\}$) and the macro average metrics of *Precision*, *Recall* and *F-measure*. Each evaluation is based on a stratified ten-fold cross-validation for a more stable performance. From the results we can see that all the DL models significantly outperform the baseline RF model. This suggests that DL methods which automatically learn features from raw traffic data are superior to traditional ML approaches which rely heavily on good designed features. Among the DL models, CNNs (CNN-784 and App-CNN) show better performance than RNNs (FS-Net and App-LSTM), especially on the precision-recall metrics. This reveals an interesting fact that a majority of mobile TLS traffic, though encrypted in the data field, can still be identified and associated to a certain app by the initial data packet, i.e., the *client hello* message during a handshake. The models of MIMETIC and App-Net are all multimodal architectures with hybrid networks. They show better performance than other unimodal networks which demonstrates the advantages of hybrid DL architectures given the heterogeneous information available from traffic data. However, with more advanced implementation and training logic, our proposed App-Net outperforms MIMETIC as well as other models, achieving a best F1 score of 91.2%.

To have a better view of the different DL implementations, we have analyzed the training complexity of App-Net as well as other baseline models in terms of trainable parameters and run time per epoch. The results are shown in Fig. 7. For the record, our experiments are all completed under the same hardware and software environment (8× Inter Core i7-9700K CPU@3.60GHz, Nvidia GeForce RTX 2070 with Ubuntu 18.04 and TensorFlow 1.12). In practice, a raw measurement of training time could be misleading, as it depends heavily on the hardware platform (i.e. CPU, GPU, RAM), software libraries used (i.e. optimized or not), etc. Thus comparative analysis should be performed. As we can see, CNN-784 has the most trainable parameters whereas the fewest training time. It is mainly because training a CNN tends to be much faster than training an RNN on Nvidia GPUs from which CNNs benefit more. MIMETIC has the fewest trainable parameters due to its simple implementation. However, the overall training time of it would surpass all others under the two-stage of training. As App-Net has applied larger hyperparameters (e.g., more filters) compared to MIMETIC to boost performance, it has much more trainable parameters.

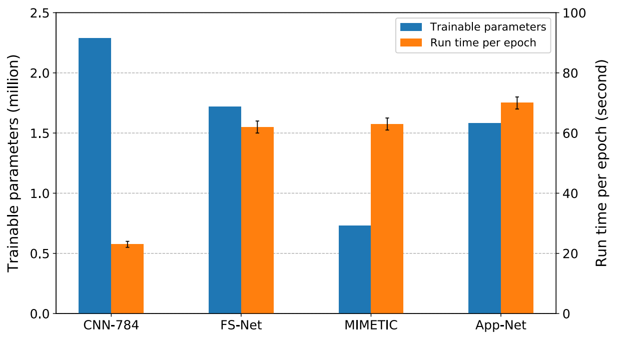


FIGURE 7. Training complexity analysis.

TABLE 3. 20 Apps most likely to be misclassified (Clustered into 5 app sets according to their owners).

	Owner	Apps
1	*Alibaba*	Tmall, Taobao, Xianyu, Feizhu, Juhuasuan, Eleme, Alipay
2	*Baidu*	Baidu, BaiduMap, BaiduDisk, Tieba
3	*Jingdong*	JD, Jddj, Jdjr
4	*Meituan*	Baidu, BaiduMap, BaiduDisk, Tieba
5	*Bytedance*	Toutiao, Douyin, Huoshan, Xigua

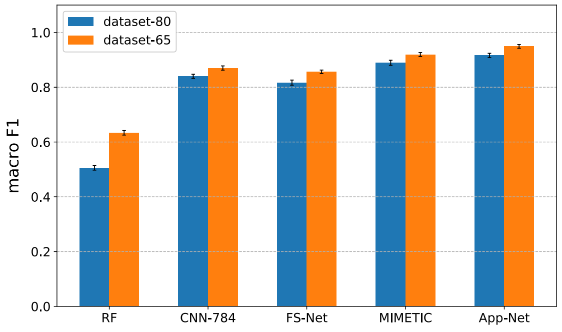
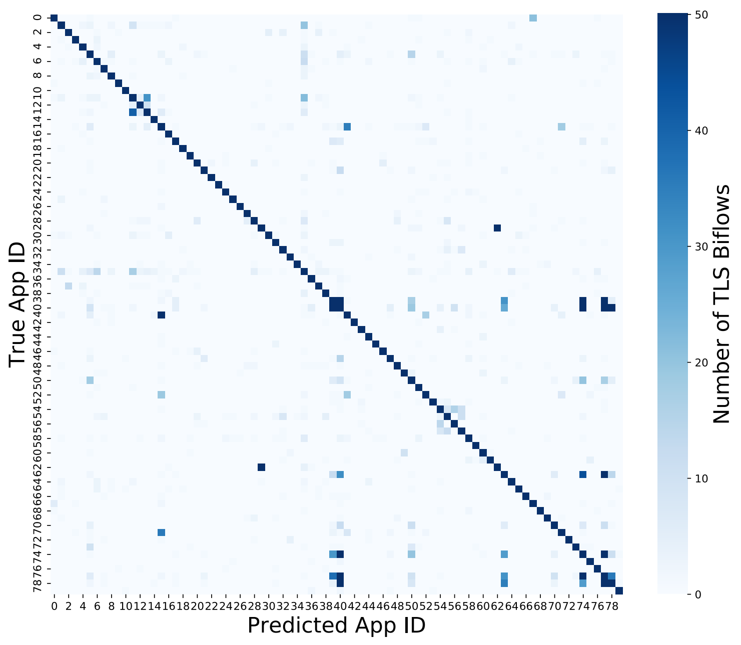


FIGURE 8. Performance comparison based on two datasets.

Despite that, the training complexity of App-Net is comparable to others.

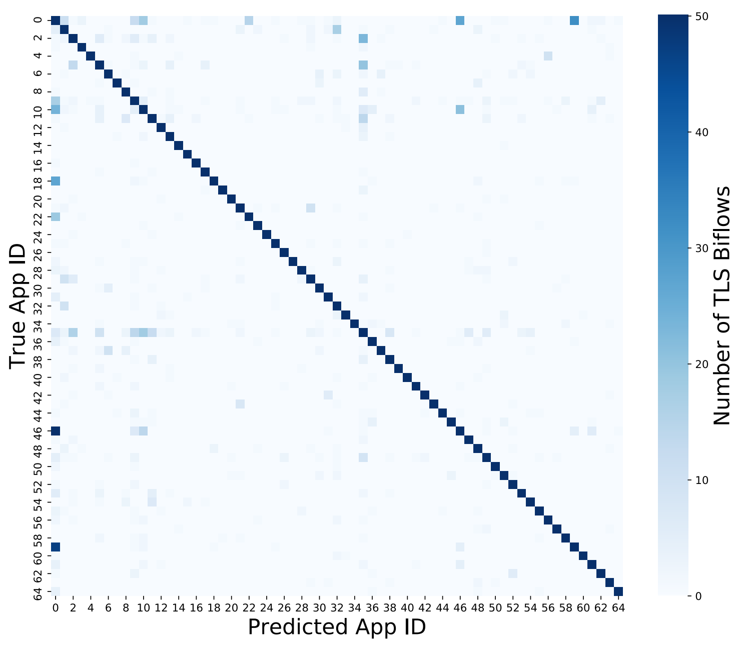
2) IMPROVING PERFORMANCE BY APP CLUSTERING

To further understand the performance of App-Net, we create a confusion matrix for all the prediction results which is depicted as a heatmap in Fig. 9. From this map we can discover the misclassification patterns by finding dark squares off the diagonal. For instance, the dark ones adjacent to the center indicate that a number of TLS biflows from app-39 (or app-40) are incorrectly classified as app-40 (or app-39). As we refer to the app name, we find that app-39 (*Tmall*) and app-40 (*Taobao*) are two highly related apps. They are both popular online shopping platforms in China belonging to Alibaba Group. For further details, we analyze the pcap trace files of the two apps with Wireshark and discover considerable information in common. For example, a certain TLS certificate with length 2390 bytes has been used in 284 TLS sessions of *Tmall* as well as 279 TLS sessions of *Taobao*. There are also dozens of TLS sessions between them sharing the same Server Name



0=PPVideo	27=Ximalaya	54=Baidu
1=QQBrowser	28=Dida	55=BaiduMap
2=QQZone	29=Dianping	56=BaiduDisk
3=QQMail	30=Kuaibao	57=Tieba
4=QQMusic	31=Tianya	58=Zhihu
5=UCBrowser	32=Haokan	59=163Music
6=QQ	33=Rugu	60=163News
7=Soul	34=Xhs	61=163Mail
8=TIM	35=WeChat	62=Meituan
9=EastMoney	36=Weibo	63=Juhuasuan
10=Amazon	37=Weishi	64=QQNews
11=Jd	38=Kuaishou	65=QQVideo
12=Jddj	39=Tmall	66=Mgtv
13=Jdjr	40=Taobao	67=Suning
14=RenRen	41=Douyin	68=Lizhi
15=Toutiao	42=Pdd	69=Huya
16=Egame	43=Tantan	70=Xiami
17=Youku	44=SohuNews	71=Xigua
18=Kg	45=SohuVideo	72=Douban
19=iFeng	46=Ctrip	73=Dingding
20=Qunar	47=Alipay	74=Xianyu
21=Koubei	48=Douyu	75=Momo
22=Hexin	49=Sina	76=Fengxing
23=Migu	50=Taopiao	77=Feizhu
24=Bilibili	51=DiDi	78=Eleme
25=Vipshop	52=Huoshan	79=Amap
26=Changba	53=iQiyi	

FIGURE 9. The confusion matrix of App-Net based on dataset-80. All the labels corresponding to the app IDs are listed on the right.



0=Alibaba	27=Changba	54=QQVideo
1=Baidu	28=Ximalaya	55=Mgtv
2=Jingdong	29=Dida	56=Suning
3=Meituan	30=Kuaibao	57=Lizhi
4=PPVideo	31=Tianya	58=Huya
5=QQBrowser	32=Haokan	59=Xiami
6=QQZone	33=Rugu	60=Douban
7=QQMail	34=Xhs	61=Dingding
8=QQMusic	35=WeChat	62=Momo
9=Bytedance	36=Weibo	63=Fengxing
10=UCBrowser	37=Weishi	64=Amap
11=QQ	38=Kuaishou	
12=Soul	39=Pdd	
13=TIM	40=Tantan	
14=EastMoney	41=SohuNews	
15=Amazon	42=SohuVideo	
16=Renren	43=Ctrip	
17=Egame	44=Douyu	
18=Youku	45=Sina	
19=Kg	46=Taopiao	
20=iFeng	47=DiDi	
21=Qunar	48=iQiyi	
22=Koubei	49=Zhihu	
23=Hexin	50=163Music	7
24=Migu	51=163News	
25=Bilibili	52=163Mail	
26=Vipshop	53=QQNews	

FIGURE 10. The confusion matrix of App-Net based on dataset-65. All the labels corresponding to the app IDs are listed on the right.

Indication (SNI) such as "**h5.m.taobao.com**", "**618.tmall.com**", "**gjusp.alicdn.com**", etc.

Through completely searching, we find 20 apps from the total 80 apps that are most likely to be misclassified, which is reported in Table 3. As we can see, those 20 apps can be further organized as 5 app sets. The apps in each set all share a same owner, thus are closely related with each other. Regardless of the results, products from one corporation are prone to have similar traffic patterns due to the shared resources and services, and hence are difficult to distinguish. In practice, when our task is not targeting every single app, it is intuitive to turn those naturally confusing apps into a whole to train a more robust classifier.

For a demo, suppose we treat each of the app set in Table 3 as one new app, the original dataset (**Dataset-80**) can be reorganized from 80 apps to 65 apps. We reevaluate App-Net as well as the baselines on the new dataset (**Dataset-65**), and the performance results are shown in Figure 8. It is obvious that with those 20 apps clustered into 5, the classification performance is improved significantly over all six models. In Figure 10, we also create a similar confusion matrix on the new dataset for App-Net. It directly shows that the misclassified examples have been greatly reduced compared with Figure 9. The experiment demonstrates how we can improve the performance of APP-ID with a strategy of app clustering. In analysis of Figure 10 we can still find a few misclassifications between apps, e.g., *Wechat* and *JD*, *Wechat* and *Zhihu*. Although *Wechat* does not associate with the other two apps directly, they are connected by some other ways. As one of the *JD*'s major shareholders, *Tencent* (owner of *Wechat*) has reserved a shopping portal for *JD* on *Wechat* which allows *Wechat* users to shop on *JD* directly. On the other hand, it provides cloud service not only to its own business but also to a large group of other companies like *Zhihu*, which is why the sessions from *Zhihu* include communications with *Tencent* cloud servers. This has revealed the intricate relationships among the modern apps and explains to a certain extent why it is hard to enable accurate mobile app identification via traffic classification.

VI. CONCLUSION

In this article we propose App-Net, a novel hybrid neural network which combines bi-LSTM and 1D-CNN in a multimodal way to identify mobile apps from encrypted TLS traffic. Through the planned RNN and CNN paths, App-Net can extract important features from the length sequence as well as the initial packet payloads in a TLS biflow at the same time. The end-to-end framework finally learns coordinated and joint representations which can identify mobile apps more effectively. The experiment results on the real-world dataset show that App-Net can achieve 93.2% accuracy and 91.2% F1 score for 80-class classification which outperforms many other state-of-the-art methods. We also find out that the classification performance can be directly improved by clustering the traffic of those naturally confusing apps in the dataset (i.e., apps owned by the same parent corporation).

In this work we focus on the problem of APP-ID from encrypted TLS traffic and use a deep hybrid neural network along with raw TLS data to tackle it. In the real-world scenario with highly-dynamic mobile traffic, periodic retraining is compelling to perform. Thus, simpler and faster architecture (e.g., with fewer filters) with a cost of little accuracy loss can be considered for reduced need of training time. On the other hand, as more secure protocols are being deployed, such as TLS 1.3 which carries fewer cleartext fields and QUIC which is always encrypted and with no cleartext, the proposed method may become less effective. We believe more advanced DL architectures are needed for the upcoming new challenges.

REFERENCES

[1] A. Dainotti, A. Pescape, and K. Claffy, "Issues and future directions in traffic classification," *IEEE Netw.*, vol. 26, no. 1, pp. 35–40, Jan. 2012.

[2] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 439–454.

[3] A. Tongaonkar, "A look at the mobile app identification landscape," *IEEE Internet Comput.*, vol. 20, no. 4, pp. 9–15, Jul. 2016.

[4] S. Miskovic, G. M. Lee, Y. Liao, and M. Baldi, "Appprint: Automatic fingerprinting of mobile applications in network traffic," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2015, pp. 57–69.

[5] Q. Xu, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, A. Nucci, and T. Andrews, "Automatic generation of mobile app signatures from traffic observations," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1481–1489.

[6] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao, "Samples: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 439–451.

[7] G. Ranjan, A. Tongaonkar, and R. Torres, "Approximate matching of persistent LExicon using search-engines for classifying mobile app traffic," in *Proc. IEEE INFOCOM 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[8] Y. Chen, W. You, Y. Lee, K. Chen, X. Wang, and W. Zou, "Mass discovery of Android traffic imprints through instantiated partial execution," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 815–828.

[9] 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 *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 433–441.

[10] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 1, pp. 63–78, Jan. 2018.

[11] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *J. Netw. Comput. Appl.*, vol. 103, pp. 131–145, Feb. 2018.

[12] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescape, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 2, pp. 445–458, Jun. 2019.

[13] X. Wang, S. Chen, and J. Su, "Real network traffic collection and deep learning for mobile app identification," *Wireless Commun. Mobile Comput.*, vol. 2020, pp. 1–14, Feb. 2020.

[14] H. Tang, Y. Cui, J. Wu, X. Yang, and Z. Yang, "Trigger relationship aware mobile traffic classification," in *Proc. Int. Symp. Quality Service*, 2019, pp. 1–10.

[15] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning," *Comput. Netw.*, vol. 165, Dec. 2019, Art. no. 106944.

[16] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Toward effective mobile encrypted traffic classification through deep learning," *Neurocomputing*, vol. 409, pp. 306–315, Oct. 2020.

[17] H. F. Alan and J. Kaur, "Can Android applications be identified using only TCP/IP headers of their launch time traffic?" in *Proc. 9th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2016, pp. 61–66.

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

[19] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent comparison of popular DPI tools for traffic classification," *Comput. Netw.*, vol. 76, pp. 75–89, Jan. 2015.

[20] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "NetworkProfiler: Towards automatic fingerprinting of Android apps," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 809–817.

[21] B. Bram and C. Brubaker. (2019). *An Update Android TLS Adoption*. [Online]. Available: https://android-developers.googleblog.com/2019/12/an-update-on-android-tls-adoption.html

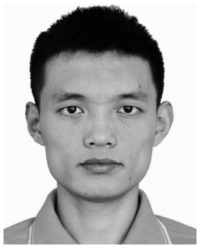
[22] *Preventing Insecure Network Connections*. Accessed: Mar. 20, 2020. [Online]. Available: https://developer.apple.com/documentation/security/preventing_insecure_network_connections

[23] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *Int. J. Netw. Manage.*, vol. 25, no. 5, pp. 355–374, 2015.

[24] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," Dept. Comput. Sci., Queen Mary Univ. London, London, U.K., Tech. Rep., 2013.

[25] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.

[26] K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham, "Adaptive encrypted traffic fingerprinting with bi-directional dependence," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 177–188.
[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[28] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, vol. 24, no. 11, pp. 1–10, 2015.
[29] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proc. IEEE Int. Conf. Intell. Secur. Informat. (ISI)*, Jul. 2017, pp. 43–48.
[30] 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. 18042–18050, 2017.
[31] 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 Comput.*, vol. 24, no. 3, pp. 1999–2012, Feb. 2020.
[32] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-Net: A flow sequence network for encrypted traffic classification," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, May 2019, pp. 1171–1179.
[33] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 781–789.
[34] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order Markov chains and application attribute bigrams," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1830–1843, Aug. 2017.
[35] 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 *Proc. IEEE/ACM 26th Int. Symp. Quality Service (IWQoS)*, Jun. 2018, pp. 1–10.
[36] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1723–1732.
[37] R. Wang, Z. Liu, Y. Cai, D. Tang, J. Yang, and Z. Yang, "Benchmark data for mobile app traffic research," in *Proc. 15th EAI Int. Conf. Mobile Ubiquitous Syst. Comput., Netw. Services*, Nov. 2018, pp. 402–411.
[38] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "MIRAGE: Mobile-app traffic capture and ground-truth creation," in *Proc. 4th Int. Conf. Comput., Commun. Secur. (ICCCS)*, Oct. 2019, pp. 1–8.
[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[42] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proc. ICML*, 2011, pp. 689–696.
[43] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019.
[44] A. Langley, *A Transport Layer Security (TLS) Clienthello Padding Extension*, document RFC 7685, Internet Requests for Comments, RFC Editor, Oct. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7685.txt
[45] H. Narasimhan, W. Pan, P. Kar, P. Protopapas, and H. G. Ramaswamy, "Optimizing the multiclass F-measure via biconcave programming," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 1101–1106.
[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[47] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017, pp. 712–717.



XIN WANG received the B.S. and M.S. degrees in networking engineering from the National University of Defense Technology, China, in 2013 and 2016, respectively, where he is currently pursuing the Ph.D. degree. His research interests include network security and traffic analysis.

SHUHUI CHEN received the Ph.D. degree from the School of Computer, National University of Defense Technology, China, in 2007. He is currently a Professor with the National University of Defense Technology. His research interests include network protocol and network security.

JINSHU SU (Senior Member, IEEE) received the B.S. degree in mathematics from Nankai University, Tianjin, China, in 1985, and the M.S. and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, China, in 1988 and 2000, respectively.

He is currently a Professor with the School of Computer Science, National University of Defense Technology. He leads the Distributed Computing and High Performance Router Laboratory and the Computer Networks and Information Security Laboratory, which are key laboratories of National 211 and 985 projects, China. He also leads the High Performance Computer Networks Laboratory, which is a key laboratory of Hunan, China. His research interests include Internet architecture, Internet routing, security, and wireless networks.