Computer Networks 179 (2020) 107372

Contents lists available at ScienceDirect

Computer Networks

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

App trajectory recognition over encrypted internet traffic based on deep neural network

Ding Li, Wenzhong Li*, Xiaoliang Wang, Cam-Tu Nguyen, Sanglu Lu

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, China

ARTICLE INFO

Keywords:
Encrypted internet traffic
Time series segmentation
Mobile app and activity classification

ABSTRACT

Despite the increasing popularity of mobile applications and the widespread adoption of encryption techniques, mobile devices are still susceptible to security and privacy risks. In this paper, we propose *ActiveTracker*, a new type of sniffing attack that can reveal the fine-grained trajectory of userâs mobile app usage from a sniffed encrypted Internet traffic stream. It firstly adopts a sliding window based approach to divide the encrypted traffic stream into a sequence of segments corresponding to different app activities. Then each traffic segment is represented by a normalized temporal-spacial traffic matrix and a traffic spectrum vector. Based on the normalized representation, a deep neural network (DNN) model which consists of an app filter and an activity classifier is developed to extract comprehensive features from the input and uncover the crucial app usage trajectory conducted by the user. By extensive experiments on real-world app usage traffic collected from volunteers and on our synthetic traffic data, we show that the proposed approach achieves up to 79.65% accuracy in recognizing app trajectory over encrypted traffic streams.

1. Introduction

The popularity of mobile applications (apps) is increasing dramatically in the past few years. People frequently use mobile apps for social interaction, online shopping, gaming, route navigation, enjoying music, watching videos, etc. According to the report [1], in the year of 2018, mobile apps accounted for 58% of worldwide Internet traffic and will continue to grow rapidly.

Due to the broadcast nature of wireless communications, mobile devices are susceptible to security and privacy risks. Malicious attacks such as sniffing may reveal users' sensitive information [2–6]. For example, the traffic classification techniques [7,8], by inspecting the headers (e.g., protocol type, IP address, port, etc) of the IP packets and the payloads, can infer the application types and the corresponding protocols (e.g., email, news, VoIP, etc). To enhance security, encryption techniques have been applied in different levels of the communication process [9]. For example, the Transport Layer Security (TLS) protocol has been widely used by many mobile apps to encrypt the application data to avoid the inspection of payload [10,11]. The Internet Protocol Security (IPsec) protocol can be used to encrypt data flows between a pair of hosts. The Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access (WPA) standard have been widely applied in wireless local area networks (WLANs) to prevent unauthorized access to the network. However, the recent researches [12–14] showed that, the information of mobile app usage can be inferred by examining the temporal-spacial patterns of the encrypted Internet traffic packets.

The works of app fingerprinting [15–18] tended to establish unique features for app distinction. The features were extracted from the traffic level, code level, and system level. For example, NetworkProfiler [15] automatically generated network profiles for identifying Android apps according to the HTTP headers in the traffic level. AppDNA [19] inspected the function-call-graph to form app fingerprint in the code level. POWERFUL [17] fingerprinted mobile apps by analyzing their power consumption patterns in the system level.

Recently, several works focused on in-app activity classification [12,13,20] that aimed to recognize the usage of different services within a particular app such as Whatsapp. Fu et al. proposed a system for classifying service usages of mobile messaging apps by jointly modeling user behavioral patterns, network traffic characteristics, and temporal dependencies [12], and developed an online analyzer to improve feature extraction and achieve in-app activity classification in real-time [13]. However, the existing works only focused on identifying the activity within a particular app, and they lacked the ability to recognize both app and in-app activity in a fine-grained level.

* Corresponding author.
E-mail addresses: liding@smail.nju.edu.cn (D. Li), lwz@nju.edu.cn (W. Li).

https://doi.org/10.1016/j.comnet.2020.107372
Received 15 February 2020; Received in revised form 7 June 2020; Accepted 16 June 2020
Available online 17 June 2020

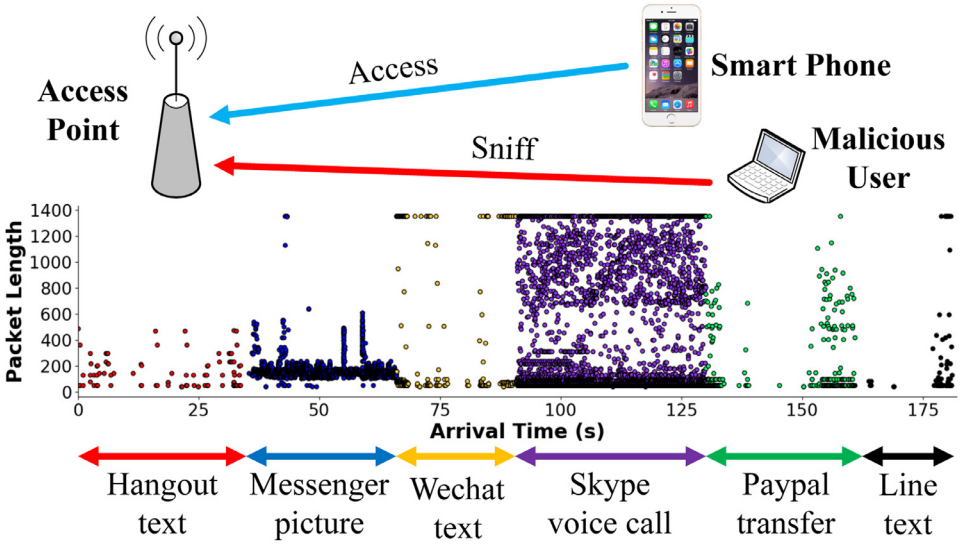


Fig. 1. An example of app trajectory recognition.

In this paper[1], we address a more challenging task: uncovering the trajectory of user's mobile app usage from a continuous encrypted Internet traffic stream. Specifically, we focus on the *app trajectory recognition problem*: inferring *which* apps are used to conduct *what* activities by analysing the encrypted Internet traffic stream sniffed from a user. Fig. 1 illustrates an example that a malicious attacker sniffs the encrypted traffic of a user via a public access point. As shown in the figure, there is a clear pattern (e.g., the packet size, the packet interval, etc.) in the encrypted traffic stream when the user conducts different activities with different apps. By exploring the patterns, a well-designed algorithm can uncover the trajectory of mobile app usage in a fine-grained level. In other words, the described technology can be considered as a new form of attack: an adversary can sniff the encrypted traffic and infer user's sensitive information such as "sending pictures with Skype and transferring money with PayPal".

The conventional works of app fingerprinting and in-app activity classification cannot solve the app trajectory recognition problem directly. The reason and technical challenges are discussed as follows. Firstly, the conventional approaches were designed for recognizing either app or activity, but not both. The combination of apps and activities forms a more complicated classification task, which yields low recognition accuracy with the conventional approaches (as shown in the performance analysis in Section 6). Secondly, the conventional approaches used hand-crafted features for classification. The extraction of features heavily relies on human experience, and the hand-crafted features are not thorough enough to differentiate similar activities on different apps (e.g., text messaging with Skype and text messaging with WeChat), which leads to poor performance as shown in Section 6. Thirdly, to uncover the trajectory of app usage from a continuous encrypted traffic stream, a method is needed to correctly partition the traffic stream into segments representing different activities, which has not been well studied in the past.

To address these challenges, we propose ActiveTracker, a novel deep learning framework to uncover the trajectory of app usage from the encrypted Internet traffic stream. Inspired by the great success of deep learning techniques in different research areas such as street-level imagery [22–25] and trajectory prediction [26–28], we exploit the ability of deep learning in automatic feature extraction from complex high-dimensional input signals and end-to-end learning to capture non-linear dependencies for app trajectory recognition. In the proposed deep learning framework, it first adopts a sliding window based approach to divide the traffic stream into a sequence of segments, where each segment corresponds to an app activity. The traffic segment is then normalized and represented by a temporal-spacial traffic matrix and a traffic spectrum vector. Using the normalized data as input, a deep neural network (DNN) model is proposed for activity recognition. Combining the recognition results of the sequence of traffic segments, the trajectory of app usage can be uncovered, which may lead to the leakage of sensitive personal information of the mobile users.

Table 1 highlights the differences of ActiveTracker and the existing works on app fingerprinting and in-app activity classification. The main results and contributions of this paper are summarized as follows:

- We design a novel sliding window based approach for encrypted traffic stream segmentation, which is able to accurately partition an encrypted Internet traffic stream into multiple single-activity substreams.
- We propose a DNN-based classification model for activity recognition from traffic segments. The proposed model uses convolutional neural network to extract features from the traffic segments automatically, and achieves high accuracy in activity recognition. To the best of our knowledge, we are the first to solve the problem of app trajectory recognition over encrypted Internet traffic streams.
- We conduct extensive experiments based on real-world Internet traffic collected from volunteers. The results show that the proposed approach achieves up to 79.65% accuracy in uncovering app trajectory from a long traffic stream. Our work will draw people's attention to privacy protection of mobile app communications.

The rest of the paper is organized as follows. Section 2 summarizes the related works. Section 3 formulates the research problem of app trajectory recognition over encrypted traffic streams. Section 4 introduces our proposed solution in detail. Section 5 introduces our data collection method and the statistics of the collected traffic. Section 6 evaluates our solution framework with extensive experiments. Finally, we conclude our paper in Section 7.

2. Related work

We summarize the related work into three categories: Internet traffic classification, app fingerprinting, and in-app activity classification.

[1] This paper is an extended version of the conference paper published in [21]. In this paper, significant modifications have been made to improve the practicability of the research problem, enhance the solution framework and algorithm with rationale, and provide additional experimental results with detailed analysis.

Table 1
Comparison of ActiveTracker and the existing works.

	App recognition	Activity recognition	App trajectory recognition	Feature	Classifier
App fingerprinting	✓	×	×	hand-crafted	SVM, random forest, etc.
In-app activity classification	×	✓	×	hand-crafted	SVM, random forest, etc.
Active Tracker	✓	✓	✓	automatic	Deep neural network

2.1. Internet traffic classification

The conventional traffic classification focused on distinguishing the traffic generated by different Internet protocols. Zhang et al. designed a nearest neighbor-based method to address the problem of very few training samples [29]. Wang et al. proposed a semi-supervised traffic clustering scheme called SBCK [30] that made decisions with consideration of some background information in addition to the observed traffic statistics. Lotfollahi et al. proposed Deep Packet [31], which embedded stacked autoencoder and convolutional neural network to identify traffic types and end-user apps. Kohout et al. proposed to identify applications in network traffic by exploiting unique communication patterns of applications and communication protocols [32]. Li et al. designed a novel neural network called BSNN [33] to predict application protocols. Besides the above works, some researchers focused on devising the classification methods specifically for multimedia traffic: Dong et al. propsed a novel feature selection method to extract flow statistical features of video traffic flows, and then developed a hierarchical k-Nearest Neighbor (k-NN) classification algorithm based on the selected features [34]; Dias et al. presented a real-time classification module for video streaming traffic [35]; more recently, Wu et al. proposed a novel feature selection and instance purification method based on consistency measure for multimedia traffic classification [36]. Anderson et al. analyzed six common machine learning algorithms, and showed how they each performed on the problem of detecting malicious, encrypted network sessions [37]. However, the above works mainly focused on Web traffic classification, and did not consider the traffic generated by mobile apps.

2.2. App fingerprinting

App fingerprinting aims to extract unique features from traffic level, code level, and system level to identify the app usage. Dai et al. proposed a fingerprint-based technique, called NetworkProfiler [15], which automatically generated network profiles for identifying Android apps in the HTTP traffic. Specifically, they chose the combination of HOST field within the HTTP header and invariant patterns within the HTTP header as the fingerprint of an app. Xu et al. built a system called Flow Recognition (FLOWR) [38], which learned the apps' distinguishing features via traffic analysis. It focused on key-value pairs in HTTP headers and identified the pairs suitable for app signatures. Yao et al. [16] generated conjunctive rules from HTTP flow header to identify app. Ranjan et al. [39] transformed app identification into an information retrieval problem, and used lexical similarity as a metric for classification task. However, the works above assumed that mobile apps run over HTTP, and thus their methods were not applicable over encrytped Internet traffic. To solve the problem of identifying apps in encrypted traffic streams, [40–42] adopted machine learning techniques to avoid the inspection of payloads. Moreover, Sengupta et al. devised a novel set of bit-sequence based features to help differentiate Android apps [11]. Besides machine learning techniques, researchers also developed other methods for app fingerprinting over encrypted traffic. Chen et al. proposed MAAF [43] which utilized information in DNS traces and handshake certificates. Korczyński et al. first proposed to employ first-order homogeneous Markov chains to model possible sequences of the SSL/TLS message types [44]. More recently, Shen et al. proposed a method based on the second-order homogeneous Markov chains to model message type transitions in SSL/TLS sessions for each specific application [10]. However, the above works only focused on recognizing the apps, ingoring the identification of the activities conducted with the apps.

2.3. In-app activity classification

In-app activity classification aims to recognize the usage of different activity within a particular app such as Whatsapp. Xu et al. identified traffic from distinct apps based on HTTP signature using anonymized network measurements from a tier-1 cellular carrier [45]. Fu et al. developed CUMMA [12], which classified in-app activities by jointly modeling user behavioral patterns, network traffic characteristics, and temporal dependencies. In their follow-up work [13], they designed an efficient feature selection framework to select most discriminative features of network traffic to meet the online efficiency requirement, and a random forest classifier with filtering (FRF) to classify in-app activities. In [46], Fu el al. devised a multi-label multi-view logistic classification method which first constructed the packet-length view and the time-delay view to exploit their mutual agreement, and then learned a multi-label multi-view logistic classifier to overcome the interference from mixed usage traffic. In their extended work [20], they further incorporated the idea of multi-view learning into multi-label logistic predictive model to improve the prediction accuracy.

In summary, the existing works focused on either app recognition or activity recognition over Internet traffic, but none of them can identify both app and activity at the same time. To the best of our knowledge, our work is the first to apply deep learning techniques to solving the app trajectory recognition problem, which hasn't been addressed in the past.

3. Problem formulation

In this section, we introduce the adversary model for recognizing app trajectory from encrypted traffic and formulate the problem of mobile app traffic segmentation and classification.

3.1. Adversary model

We consider the scenario that an adversary aims to uncover the trajectory of app usage of a mobile user, as illustrated in Fig. 1. Since wireless communications are broadcast, the attacker can sniff the encrypted Internet traffic of the target user on the same WLAN to collect data streams for further analysis. A number of sniffing tools such as *Wireshark* or *aircrack-ng* can be used to collect the wireless traffic between hosts and the access point. Such attack can be applied to most encrypted wireless networks such as the airport's WiFi and the coffee-shops' networks.

3.2. Definitions and assumptions

Here we provide some definitions and assumptions used in the rest of this paper.

Definition 1 (Encrypted Internet traffic stream)**.** An encrypted Internet traffic stream is defined as a sequence of packets $F = \{p_1, p_2, \dots, p_N\}$, where p_i $(1 \le i \le N)$ is the information of the ith observed packet represented by $p_i = \langle T_i, L_i, O_i \rangle$, where T_i is the timestamp of the packet; L_i is the length of the packet; and $O_i \in \{0, 1\}$ represents the direction of the packet (sent or received).

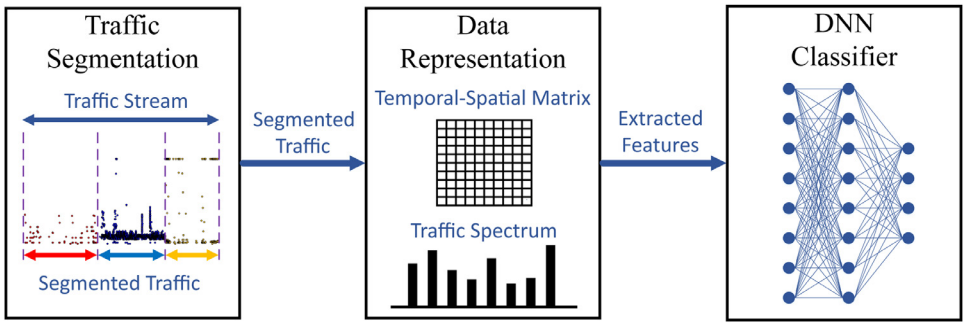


Fig. 2. The framework of DNN based app trajectory recognition.

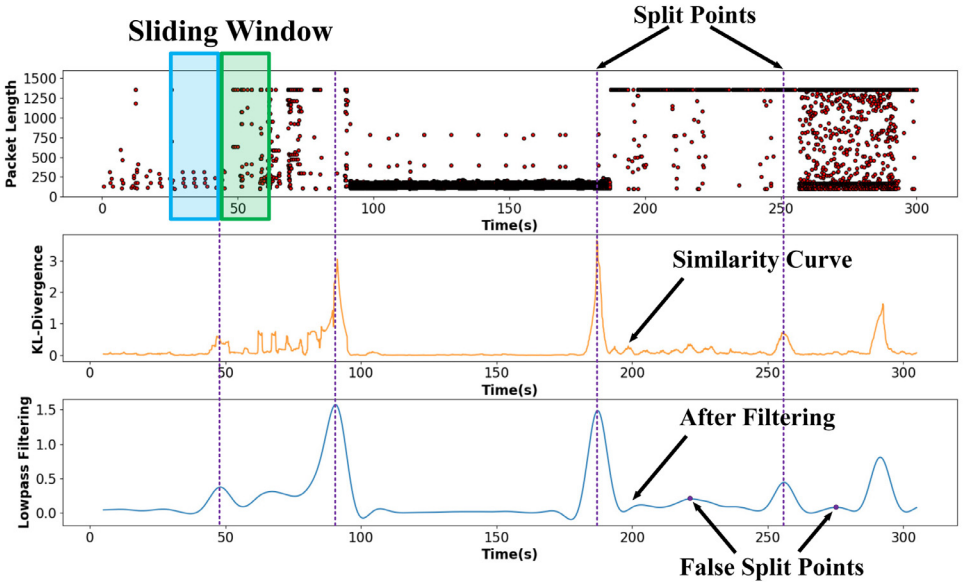


Fig. 3. Illustration of the sliding window mechanism.

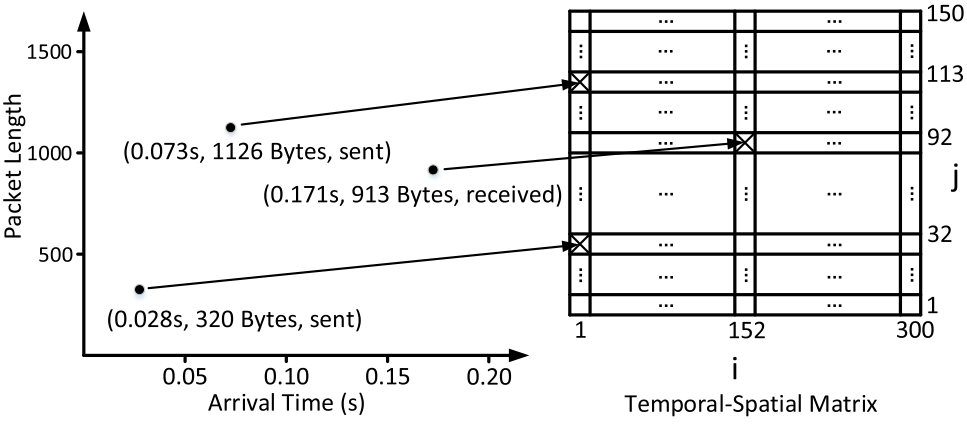


Fig. 4. Illustration of temporal-spatial matrix construction.

We assume the adversary can observe only very limited information from the encrypted packet, i.e., only the timestamp, packet length, and direction are observable to the attacker.

The length of an Internet traffic stream is defined as the time interval between the first and the last packet, i.e., $length(F) = T_N - T_1$.

Definition 2 (Encrypted traffic segment)**.** An encrypted traffic segment is defined as a continuous subsequence of an encrypted Internet traffic stream. For example, $F(T_i, T_j) = \{p_i, p_{i+1}, \ldots, p_j\}$ is an encrypted traffic segment of F.

Normally, an encrypted traffic segment corresponds to some app activity of the user. We assume that the activity should last for at least 15 s. Therefore the length of $F(T_i, T_j)$ should be longer than 15 s. If the length of $F(T_i, T_j)$ is too short, there won't be enough statistical information for activity recognition.

Definition 3 (App activity recognition)**.** The app activity recognition task is to find a mapping from an encrypted traffic segment to the app activity: $F(T_i, T_j) \rightarrow \langle app, activity \rangle$, where the app activity is represented by a tuple $\langle app, activity \rangle$. The task aims to recognize both the name of the app and the activity conducted with it.

Definition 4 (App trajectory recognition)**.** The app trajectory recognition task tends to map an encrypted Internet traffic stream F to a sequence of app activities: $F \rightarrow \langle T_1, app_1, activity_1 \rangle, \langle T_2, app_2, activity_2 \rangle, \ldots, \langle T_K, app_K, activity_K \rangle$. In other words, it aims to uncover which app is used for what activity at some moment by analysing the encrypted Internet traffic stream.

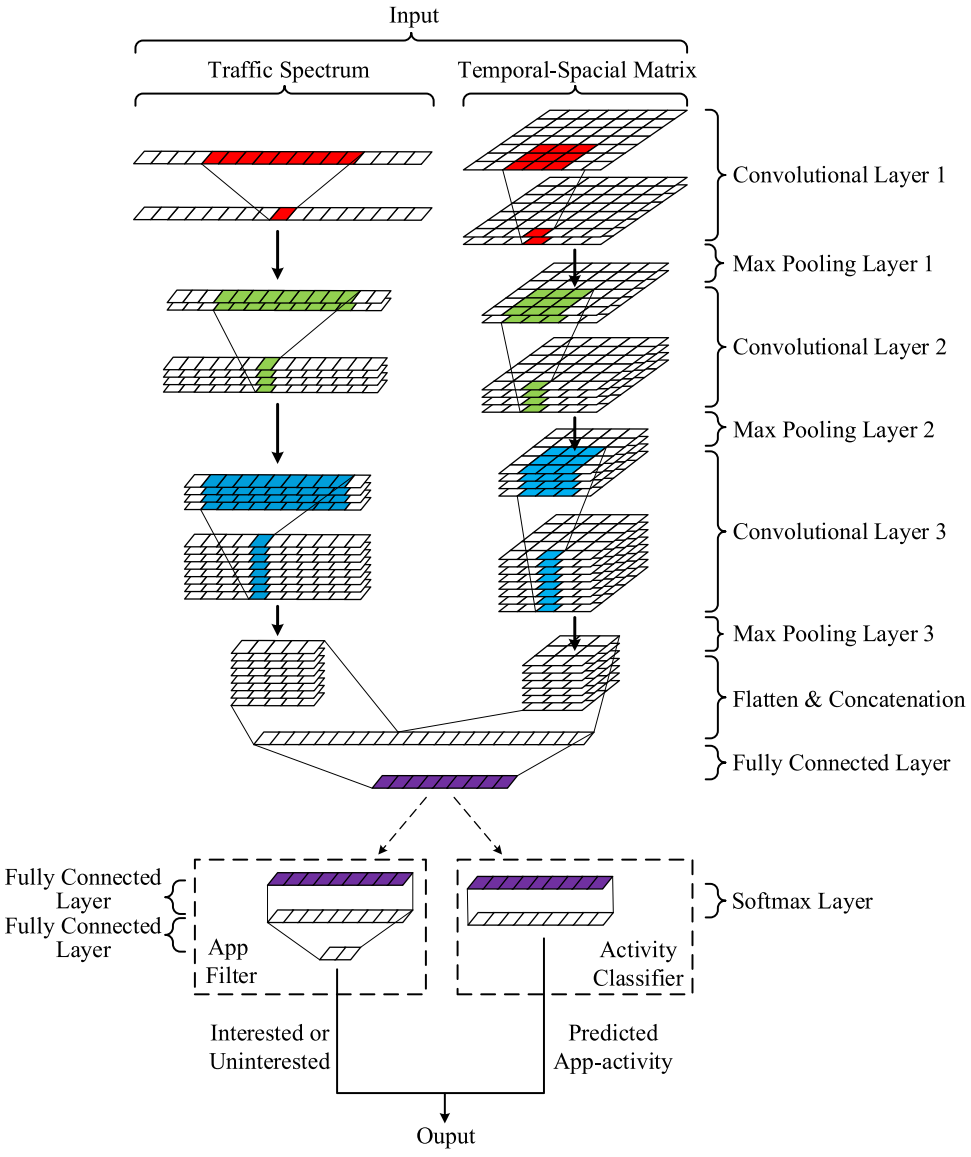


Fig. 5. The proposed DNN structure.

3.3. Problem statement

Given a sniffed encrypted Internet traffic stream, the goal of the attacker is to recognize the app trajectory of the stream. To achieve this goal, we can first partition the traffic stream into segments, then recognize the app activity of each segment using a classification algorithm, and finally combine the results to form the app trajectory. Specifically, the app trajectory recognition problem can be solved by considering the following sub-problems:

3.3.1. Encrypted internet traffic stream segmentation

Given an encrypted Internet traffic stream, the objective of the first sub-problem is to find a set of split points to partition the stream into a sequence of encrypted traffic segments, such that the packets within a segment are generated from the same app activity usage.

3.3.2. Encrypted traffic segment classification

Given an encrypted traffic segment, the objective of the second sub-problem is to identify the name of the app as well as the in-app activity from the traffic.

Next, we will propose an approach to uncover app trajectory from the encrypted Internet traffic stream based on deep neural network (DNN).

4. App trajectory recognition based on DNN

In this section, we first overview the framework of app trajectory recognition based on DNN. Then, we in detail introduce three major components of our framework: a novel sliding window based approach for encrypted traffic segmentation, a method for data representation, and a DNN classification model for app activity recognition.

4.1. Framework

Fig. 2 shows the framework of DNN based app trajectory recognition from encrypted traffic, which consists of three major components.

4.1.1. Traffic segmentation

Given a continuous encrypted Internet traffic stream, the first step is to partition the stream into segments. We adopt a sliding window based approach to divide the stream into a sequence of segments, where each segment corresponds to an app activity.

4.1.2. Data representation

Given the traffic segments, we transform them into normalized data as the input of the DNN classifier. Specifically, we represent each traf-

fic segment by a temporal-spacial traffic matrix and a traffic spectrum vector.

4.1.3. Segment classification

Using the normalized data as input, we propose a DNN model for app activity recognition. The DNN model uses a 2D convolutional neural network (CNN) and a 1D CNN to extract the features from different domains and combine these features to recognize the app activity and uncover the app trajectory.

The three components are presented in detail below.

4.2. Sliding window based traffic segmentation

We propose a novel sliding window based approach for encrypted traffic segmentation. Given a continuous encrypted traffic stream, the task of traffic segmentation is to find the *split points* that partition the stream into multiple traffic segments that correspond to different app activities. Intuitively, different app activities present different statistical patterns in their traffic regarding the packet lengths and packet intervals. Based on the intuition, we devise a sliding window based approach to search the split points that divide the traffic stream into segments with high statistical difference in their packet distributions.

The proposed approach includes four steps: sliding window mechanism, similarity curve generation, low-pass filtering, and split points identification, which are explained below.

4.2.1. Sliding window mechanism

The sliding window mechanism is illustrated in Fig. 3. We use a sliding window with length L to include a set of packets from the traffic stream. The middle point of the window divides the set of packets within the window into two parts: the left part S_l and the right part S_r. We compute the similarity of the packet statistical characteristics of the two parts: $Sim(S_l, S_r)$. If S_l and S_r belong to the traffic of different activities, the similarity between them should be low. By moving the sliding window and observing the change of similarities, the split points can be identified following the method introduced below.

4.2.2. Similarity curve generation

To assess the similarity of the packet statistical characteristics in S_l and S_r, we adopt the *Kullback-Leibler (K-L) divergence* [47] as the assessment metric for the two packet sets. The K-L divergence, also known as relative entropy, is formally formulated as follows:

$$D_{KL}(S_l||S_r) = \sum_i S_l(i) log \frac{S_l(i)}{S_r(i)}, \tag{1}$$

where $S_l(i)$ and $S_r(i)$ are the discrete probability distributions of the left part and the right part accordingly.

In mathematical statistics, the K-L divergence is a measure of how one probability distribution is different from another probability distribution. Considering that the probability distributions (e.g. on packet length) of two packet sets that are generated by different activities should be very different, K-L divergence would be a good metric for the similarity assessment between S_l and S_r. Specifically, K-L divergence is the expectation of the logarithmic difference between the probability distributions $S_l(i)$ and $S_r(i)$. If the probability distributions $S_l(i)$ and $S_r(i)$ are exactly the same, the K-L divergence equals 0. The larger K-L divergence value means the more difference between them.

In practice, we consider the similarity between $S_l(i)$ and $S_r(i)$ regarding their distributions on packet length and packet interval and the similarity $Sim(S_l, S_r)$ can be computed by the mean of their K-L divergences:

$$Sim(S_l, S_r) = \frac{D_{KL}^{length}(S_l||S_r) + D_{KL}^{interval}(S_l||S_r)}{2} \tag{2}$$

We move the sliding window step by step along the encrypted traffic stream, and compute the similarity on each point, which forms a similarity curve as shown in Fig. 3. According to the figure, when the sliding window is within the same activity, the curve is flat. When the sliding window moves across different activities, the curve first ascends and then descends. The peak of the curve appears around the split point of the two adjacent activities, since the maximum dissimilarity between the two distributions should occur at the split point in theory. Such property can be employed to identify the split points in the traffic stream, which will be introduced below.

4.2.3. Low-pass filtering

Due to the randomness of the data packets, the similarity curve is not smooth and it fluctuates as illustrated in Fig. 3. It is hard to identify the split points by searching the peaks along the curve. To reduce the fluctuation and smooth the curve, we propose a low-pass filtering approach.

In signal processing, a low-pass filter allows through signals with frequencies lower than a certain cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. Low-pass filter is widely used in signal systems to remove noises with high frequencies.

In our work, we adopt a cutoff frequency of 0.015 Hz, since we assume an app activity lasts for at least for 15 s.

4.2.4. Split points identification

After low-pass filtering, the similarity curve becomes smooth as illustrated in Fig. 3. Apparently, the split points correspond to the peaks of the similarity curve. There are still some local peaks (e.g., the false split points illustrated in Fig. 3) that could cause confusion. However, the values of these peaks are small, and most of them are less than 0.2, which means that the traffic streams around such small peaks are similar and they should not be considered as split points. This inspires us to design a heuristic approach for split points identification: we first detect the local maximum points on the similarity curve, and then compare the detected points with a predefined threshold (e.g., 0.2). If the similarity value on a local maximum point is greater than the predefined threshold, it will be identified as a split point. Using the above heuristics, we can obtain a set of split points which partition the continuous traffic stream into a set of traffic segments.

Algorithm 1 summarizes the proposed sliding window based approach for encrypted traffic segmentation.

Algorithm 1 Sliding window based approach for encrypted traffic segmentation.

Input:

An encrypted Internet traffic stream $F = \{p_1, p_2, \dots, p_N\}$.

Output:

Split points $SP = \{T_1, T_2, \dots, T_M\}$ such that the packets between two adjacent split points are generated by a single app activity.

Initial $SP = \emptyset$, threshold $\varphi = 0.2$;
Initial the length of sliding window $L = 10s$;
Set the sliding window at the beginning of F;
repeat
    Compute the similarity $s = Sim(S_l, S_r)$;
    Store the similarity s and the sliding window's current location T;
    Move the sliding window rightwards 0.1 s;
until the sliding window is at the end of F;
Generate the similarity curve FC using the stored s and T;
Feed FC into a low-pass filter, obtaining a smooth similarity curve SC;
Identify the split points by searching the local maximum points in SC and filter the false split points by discarding the points whose values are less than φ;
Add the identified split points to SP;
return SP;

4.3. Data representation

After the traffic segmentation, we can get a set of traffic segments with different sizes. To obtain a normalized representation of each traffic segment as the input of our proposed DNN classifier, we further propose a data representation method to normalize the traffic segments.

Since an app activity is assumed to be longer than 15 s, the segments with length less than 15 s will be discarded as they are unrecognizable. For each segment, we extract 15-second traffic from the middle of it. We then represent the extracted traffic stream with a *temporal-spacial traffic matrix M* and a *traffic spectrum vector V*.

4.3.1. Computation of the temporal-spatial traffic matrix M

The quantification of M tends to capture the temporal-spacial correlations of the encrypted traffic stream. In our paper, we represent M by a $k \times k$ 2D matrix (for the traffic segment of 15 s, the value of k is empirically set to 150 in our experiments) where each element M_{ij} $(1 \le i \le k, 1 \le j \le k)$ corresponds to the number of packets arriving within 0.1 s (from the $(i/10 - 0.1)$th to the $(i/10)$th second) and whose size falls in the jth bin (since the maximum transmission unit for Ethernet is 1500 bytes, we set $j = \lceil \frac{packet\ size \times k}{1500} \rceil$ in practice).

The direction of packets is also useful for differentiating different activities, since some activities (e.g. video call, voice call) always alternatively send and receive packets, while other activities (e.g. send text messages, location sharing) do not. To make use of the packet direction to improve the recognition accuracy, we extend the size of M. Specifically, we extend the temporal-spacial matrix M to the size of $2k \times k$, where element M_{ij} $(1 \le i \le k, 1 \le j \le k)$ corresponds to the number of *sent packets*, and element M_{ij} $(k+1 \le i \le 2k, 1 \le j \le k)$ corresponds to the number of *received packets*. In other words, the direction of a packet is indicated by the value of i (whether the value of i is greater than k) in the extended temporal-spatial traffic matrix.

To better illustrate how to construct a temporal-spatial traffic matrix, we give an example which is presented in Fig. 4. In the beginning, the values of all elements in the temporal-spatial matrix M are 0. For each packet in the traffic segment, we first find its corresponding position (i.e. calculate the coordinate (i, j)) in the matrix, and add one to the value in that position. For instance, for a sent packet which arrives at 0.073 s and whose length is 1126 bytes, its corresponding position in M is (1,113), since it arrives between 0 to 0.1 s and its length falls in the $\left\lceil \frac{1126 \times 150}{1500} \right\rceil = 113th$ bin. Thus, the value of $M_{1,113}$ (marked by a " × " symbol in the matrix as shown in Fig. 4) should be added by 1.

4.3.2. Computation of the traffic spectrum vector V

The traffic spectrum vector V is computed as follows. We regard the packet length as a signal of arrival time, and regard each packet as a sampling on this signal. Obviously, these samplings are generated with unstable time intervals. To obtain the traffic spectrum, we first conduct Hermite interpolation [48] to obtain an equally spaced (0.01 s) data sequence. Then, by using *Discrete-Time Fourier Transform (DTFT)* [49], we can represent the 15-second traffic as a vector of length 1500 which contains frequency domain features.

4.4. Deep neural network model

4.4.1. Neural network architecture

Inspired by the great success of DNNs in image recognition, natural language processing, computer version, etc., we design a deep convolutional neural network (CNN) architecture for app activity recognition. It has been shown that deep CNNs are powerful in autonomous feature extraction from matrix data where features extracted in shallower layers of the CNN will be fed to the successive convolutional layers in order to form more abstract features. Compared with hand-crafted features, the features extracted by CNN are more comprehensive, which would be exploited to differentiate similar activities on different apps from encrypted traffic in our system. Specifically, we propose a 2D CNN structure to extract features of a traffic stream from its temporal-spacial traffic matrix, and a 1D CNN to extract features from sequential traffic spectrum vector. The structure of the proposed deep neural network is illustrated in Fig. 5.

Table 2
The configuration of DNN model.

Network Configuration	
Input (1*1500, frequency domain)	Input (300*150, time domain)
Conv (filter=1*9, channel=2)	Conv (filter=3*3, channel=2)
Max Pooling	Max Pooling
Conv (filter=1*9, channel=4)	Conv (filter=3*3, channel=4)
Max Pooling	Max Pooling
Conv (filter=1*9, channel=8)	Conv (filter=3*3, channel=8)
Max Pooling	Max Pooling
Fully Connected	
Fully Connected	Softmax
Fully Connected	

According to the figure, the proposed architecture consists of a 2D CNN (right part of the figure) and a 1D CNN (left part of the figure). The temporal-spacial traffic matrix respectively goes through three convolutional layers and three max pooling layers, so does the traffic spectrum vector. Then, the features extracted by both 1D CNN and 2D CNN are concatenated into a feature vector. This feature vector then goes through a fully-connected layer. The fully-connected layer is able to decide the significance of each feature and assign high weights to important features. The output of the fully-connected layer is fed into an *app filter* that can determine whether the encrypted traffic is from an interested app or uninterested app, and an *activity classifier* that infers the specific in-app activities generated by the user. Finally, combining the results of app filter and activity classifier, the model outputs the recognized app activity with the highest probability.

4.4.2. Design of app filter and activity classifier

In this section, we discuss the design of app filter and activity classifier illustrated in Fig. 5.

Since typically hundreds of apps are installed on a smartphone, the adversary may only be interested in tracking the activities of a small number of apps. To achieve this, we apply an app filter to distinguish the interested apps from the encrypted traffic. The app filter works as a binary classifier, whose structure is illustrated in Fig. 5. It consists of two fully-connected layers. It takes the feature vector from DNN as input to infer whether the encrypted traffic is from the interested app or the uninterested app.

The activity classifier is applied to achieve fine-grained classification of the app activities, whose structure is illustrated in Fig. 5. It takes the feature vector from DNN as input and uses a softmax layer to form the classifier. The softmax layer applies a softmax function [50] on the input feature vector to produce the probability distribution over the predicted labels. It is assumed that there are m types of activities, and the predicted label is $y \in [1, 2, \ldots, m]$. Given an input $\boldsymbol{x}$, the probability $P(y|\boldsymbol{x})$ is estimated as follows,

$$\begin{bmatrix} P(y=1|\boldsymbol{x}) \\ \vdots \\ P(y=m|\boldsymbol{x}) \end{bmatrix} = \frac{1}{\sum_{p=1}^{m} e^{w^p N(\boldsymbol{x})}} \begin{bmatrix} e^{w^1 N(\boldsymbol{x})} \\ \vdots \\ e^{w^m N(\boldsymbol{x})} \end{bmatrix} \tag{3}$$

where $N(x)$ is the feature vector formed by the DNN and w is the parameters of the classifier that need to be learned. The predicted activity class $\hat{y}$ is the maximum of these probabilities,

$$\hat{y} = \underset{y}{\operatorname{argmax}}\ P(y|\boldsymbol{x}). \tag{4}$$

To train the model, we annotate the training data as two classes: the set of *interested app* $\mathcal{C}$ and the set of *uninterested app* $\mathcal{U}$. For interested apps, we further annotate the specific activity that a user conducted

Table 3
The interested apps and activity types.

#	Location	Picture	Text	Video Call	Voice Call	Transfer
PayPal						✓
Hangouts		✓	✓	✓	✓	
Line			✓		✓	
Messenger		✓	✓	✓	✓	
QQ				✓	✓	
Skype		✓		✓	✓	
Wechat	✓	✓	✓	✓	✓	✓
Uninterested		✓	✓			✓

Table 4
Statistics of the collected internet traffic.

Class	App	Activity	Records	Packets	Traffic	Traffic/min
1	PayPal	transfer	593	104.3K	38.2M	218.9K
2	Hangouts	picture	611	1451.1K	1074.7M	3.1M
3	Hangouts	text	728	163.0K	35.8M	267.3K
4	Hangouts	video call	608	2180.4K	1076.1M	6.5M
5	Hangouts	voice call	574	437.8K	89.9M	585.8K
6	Line	picture	832	408.6K	259.3M	503.8K
7	Line	text	707	61.9K	10.8M	49.08K
8	Messenger	picture	395	194.2K	118.3M	6.5M
9	Messenger	text	493	79.6K	15.24M	62.78K
10	Messenger	video call	208	441.5K	219.4M	4.2M
11	Messenger	voice call	146	73.7K	17.47M	461.1K
12	QQ	video call	584	2834.9K	2077.2M	18.96M
13	QQ	voice call	562	726.5K	128.0M	1.2M
14	Skype	picture	831	4651.7K	4109.3M	29.37M
15	Skype	video call	569	1682.8K	935.9M	5.28M
16	Skype	voice call	656	850.4K	163.0M	851.4K
17	Wechat	location	673	276.3K	156.3M	777.4K
18	Wechat	picture	963	1956.4K	1515.1M	8.4M
19	Wechat	text	617	41.7K	8.11M	23.04K
20	Wechat	transfer	578	75.6K	21.5M	103.9K
21	Wechat	video call	570	1673.5K	349.2M	2.5M
22	Wechat	voice call	629	1048.9K	184.9M	1.2M

with the app. The annotated traffic data forms the training set for the classifier.

We adopt the Entropic Open-Set Loss $J_E(\boldsymbol{x})$ function [51] for the proposed DNN model, which is defined as follows:

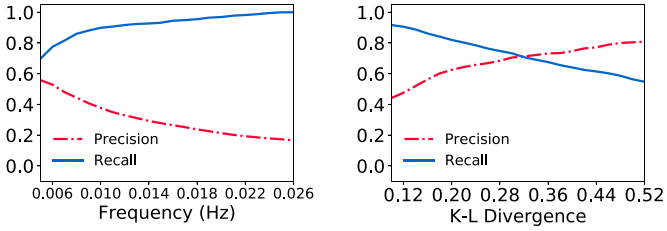
$$J_E(\boldsymbol{x}) = \begin{cases} -log\,P(y=c|\boldsymbol{x}) & \text{if } \boldsymbol{x} \text{ is from class } c \in C \\ -\frac{1}{|C|}\sum_{c=1}^{|C|} log\,P(y=c|\boldsymbol{x}) & \text{if } \boldsymbol{x} \text{ belongs to } \mathcal{U} \end{cases} \quad (5)$$

where $\boldsymbol{x}$ denotes a training sample in the training set.

The rationale of the Entropic Open-Set Loss is explained as follows. It is a piecewise function. If the input sample $\boldsymbol{x}$ belongs to an interested class, by minimizing $-log\,P(y=c|\boldsymbol{x})$, the probability that $\boldsymbol{x}$ is correctly recognized as an activity is maximized. If the input sample $\boldsymbol{x}$ belongs to an uninterested class, by minimizing $-\frac{1}{|C|}\sum_{c=1}^{|C|} log\,P(y=c|\boldsymbol{x})$, it forces the model to have equal probability to recognize $\boldsymbol{x}$ as any of the labeled activities. Therefore, it is easy to distinguish interested and uninterested apps by the probability distribution of softmax.

With the training set and the loss function, we train the DNN model alternatively. We first train the activity classifier using the Entropic Open-Set Loss function, which maximizes the probability that an activity is correctly recognized and enlarges the differences of probability distributions of interested and uninterested classes. We then fixed the parameters of DNN, and use the output feature vector to train the app filter, which intends to distinguish the interested and uninterested apps.

During model inference, the input encrypted traffic is first checked by the app filter. If it is classified as an interested class, the activity classifier will further examine it, and output the predicted app activity for the encrypted traffic.



(a) Influence of cutoff frequency. (b) Influence of threshold.

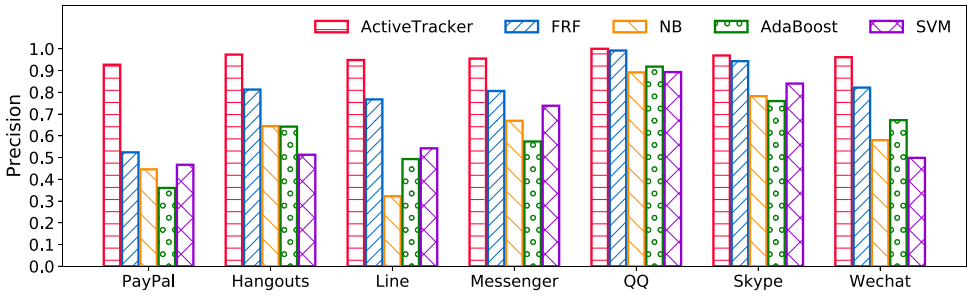
Fig. 6. The performance of traffic segmentation under different system parameters.

Table 5
Performance of app filter.

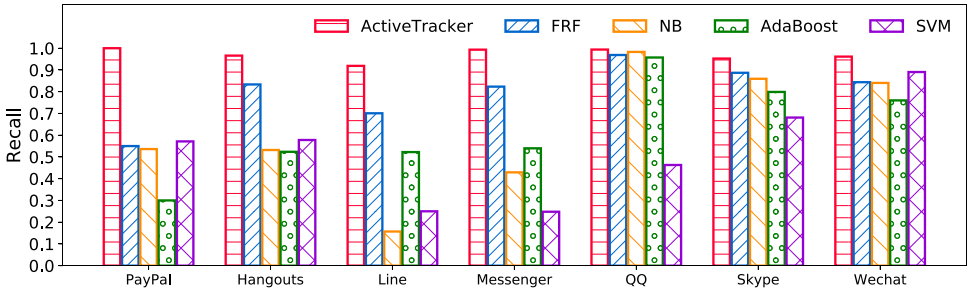
Traffic Type	Precision	Recall	F-score	Overall Acc.
Interested app	99.26%	94.12%	96.62%	94.03%
Uninterested app	61.82%	93.15%	74.32%	

4.4.3. Summary of model parameters

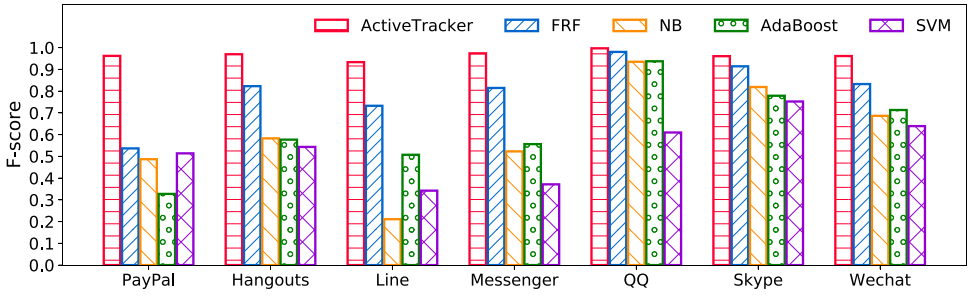
In summary, the main parameters of each layer of the proposed DNN model are described in Table 2.



(a) Precision



(b) Recall



(c) F-score

Fig. 7. Performance of app classification (real-world dataset).

5. Data collection and processing

To evaluate the performance of our framework, we conducted experiments based on both real-world collected smartphone traffic dataset and synthetic dataset. The dataset collection and precessing are described as follows.

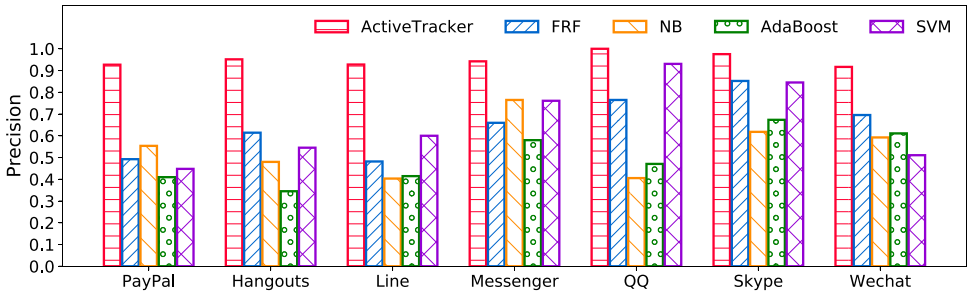
5.1. Real-world smartphone traffic dataset

We collected a dataset of encrypted Internet traffic from smartphones. We recruited 18 volunteers who were required to use a number of apps with our specially configured handsets (Samsung Galaxy S7, Huawei G9 and Huawei P8 smartphones). The smartphones are connected to a virtual Wi-Fi access point (AP) operated by a laptop (Dell XPS13-7390). A well-known sniffing tool, *WireShark*, is run on the laptop to crawl the packet information of the smartphones from the AP. To ensure the robustness of the proposed framework, besides collecting traffic data in the lab environment, we also collect data under different network environments including library, student dormitory and home.

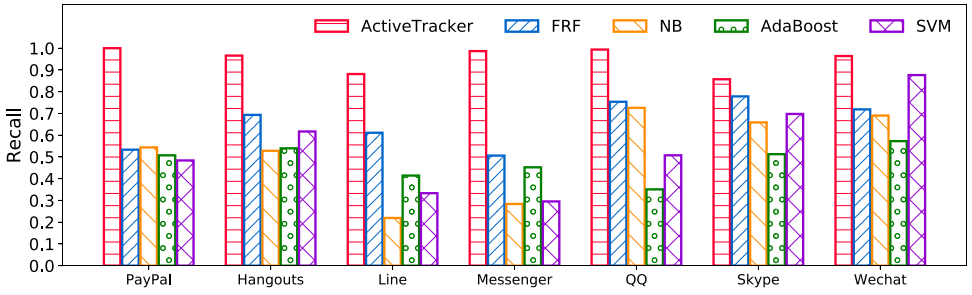
We are particularly interested in tracking 7 popular mobile apps: Paypal, Hangouts, Line, Messenger, QQ, Skype and Wechat. We focus on 6 types of in-app activities including location (sharing the current location of the user), picture (sending a photo to a friend), text (sending a text to a friend), video call, voice call, and money transfer. The interested apps and activities in our experiments are summarized in Table 3.

During data collection, the volunteer are required to use different apps to conduct different activities for a duration, and label the generated traffic with both app and activity. For example, a volunteer may use Wechat to text with his friends in the first hour, and then use Skype to make a video call in the next hour. Their apps, activities and the corresponding time durations are logged as ground truth. Besides the interested 7 apps, the volunteers also use other apps (such as Alipay, Instagram, etc) with different activities during data collection, and such apps are labeled as “uninterested” in the dataset. In the end, we form the training set with 80% random samples from the collected mobile traffic for model training, and form the test set with the rest 20% samples for performance evaluation.

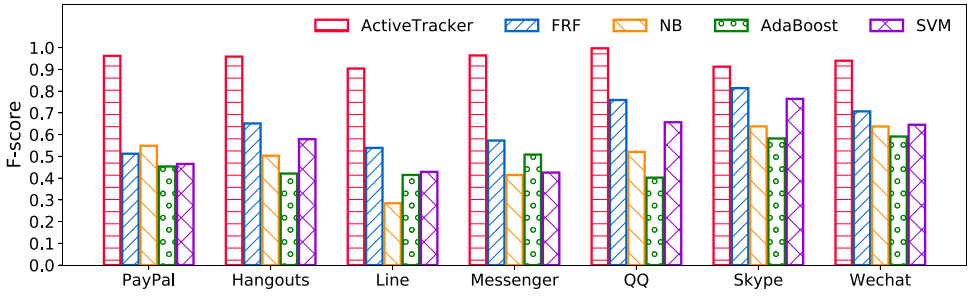
The basic statistics of the collected app activity dataset are illustrated in Table 4.



(a) Precision



(b) Recall



(c) F-score

Fig. 8. Performance of app classification (synthetic dataset).

5.2. Synthetic dataset

Since collecting real-world smartphone traffic is time-consuming, manpower-expensive, and privacy-sensitive, the data collection can only be performed by a small number of users. In order to test the performance of the proposed model in a larger dataset, we propose a method to generate a *synthetic dataset* based on the collected real-world data. Specifically, we generate pseudo app-activity trajectories by concatenating activities from different users as follows. A pseudo app-activity trajectory *Trace* is generated by $Trace = \{act_1, act_2, \ldots, act_m\}$, where act_i $(i = 1, 2, \ldots, m)$ is a random activity of the smartphone traffic drawn from one the of the 18 volunteers' usage data, and m is a random integer between 10 to 20 representing the number of activities in the trajectory. To increase the diversity of the dataset, for each activity act_i, we add a white Gaussian noise [52] to its traffic data. With the proposed method, we form a synthetic dataset with 10,000 app-activity trajectories, which are used for performance evaluation in the following sections.

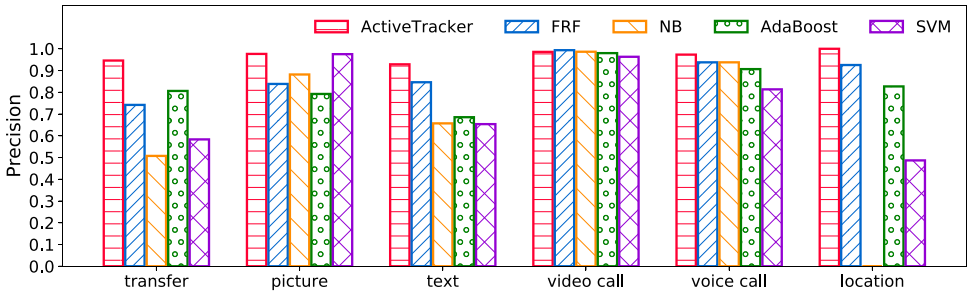
6. Performance evaluation

To validate the effectiveness of our proposed framework, we conducted extensive experiments on the collected real-world traffic data as well as the synthetic traffic data. All experiments were performed on a single machine with Intel Core i5-6600 processors (4 cores / 4 threads), 8 GB of memory, and NVIDIA GeForce GTX 1070 GPU. To implement our proposed DNN model, we used the PyTorch library [53,54].

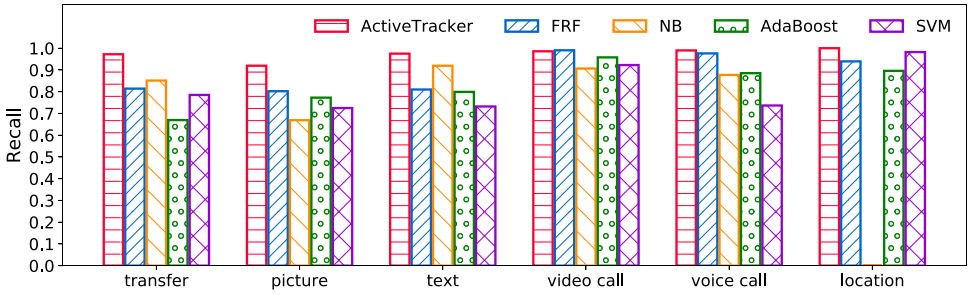
6.1. Performance metrics and baselines

Similar to the work of [12,13], we adopt the widely used metrics for performance evaluation of segmented traffic classification: accuracy, precision, recall, and F-score. Their definitions can be found in [12], and they are omitted due to page limit.

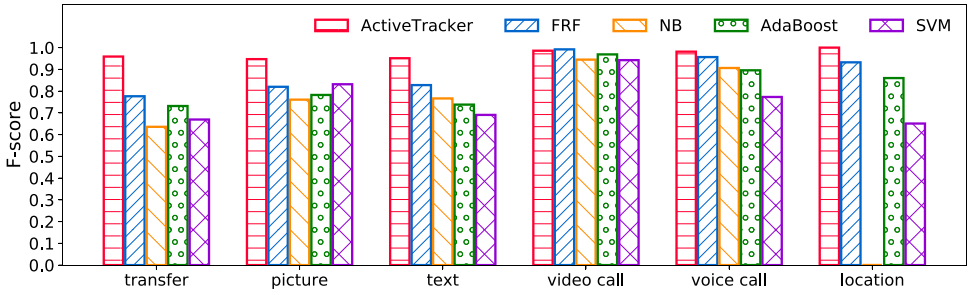
In order to confirm the effectiveness of our proposed DNN classifier, we compare our method with the random forest classifier with fitering called FRF in [13], which is the state-of-the-art solution for in-app



(a) Precision



(b) Recall



(c) F-score

Fig. 9. Performance of in-app activity classification (real-world dataset).

activity recognition. In addition, we also compare our method with several shallow machine learning algorithms including Naive Bayesian (NB) [55], Adaboost Classifier [56], and Support Vector Machine (SVM) [57].

6.2. Performance analysis

6.2.1. Performance of the traffic segmentation algorithm

We first show the performance of the proposed sliding window based approach for encrypted traffic segmentation. Two system parameters will influence the performance of the algorithm: the cutoff frequency of low-pass filtering and the threshold of K-L divergence for identifying split points.

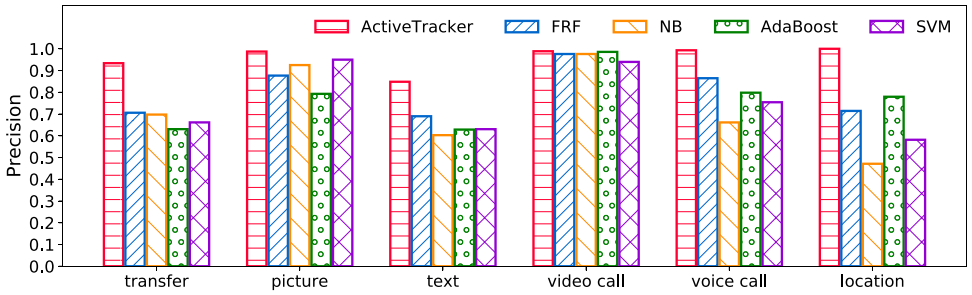
Fig. 6 a shows the performance under different cutoff frequencies. With the increase of cutoff frequency, the recall approaches 1, which means almost all the true split points can be identified by the algorithm. However, the precision decreases with the increase of cutoff frequency, which means more false split points are misclassified. As a trade-off, we set the cutoff frequency as 0.015 such that the recall is above 0.8, and the precision remains above 0.5.

Fig. 6 b shows the influence of tuning the threshold of K-L divergence. When the threshold is small, the recall is high but the precision is low. Increasing the threshold will decrease recall and improve the precision. As a trade-off, we set the threshold as 0.2 such that the recall is above 0.8, and the precision achieves above 0.6.

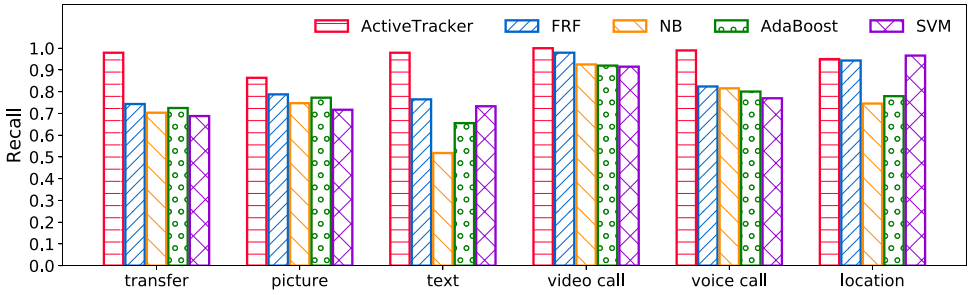
In the task of traffic segmentation, recall is more important than precision. Higher recall means more split points are correctly identified. The system tolerates false split points (lower precision), since a false split point simply divides the same activity into two segments, which will not influence the identification of app activity.

6.2.2. Performance of app filter

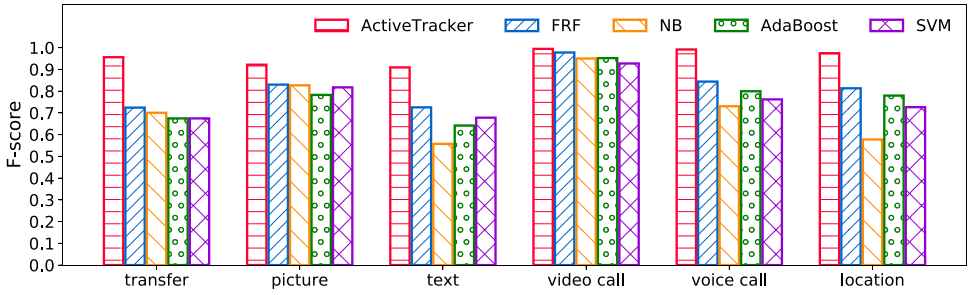
We test the ability of app filter to recognize the traffic of uninterested apps. The results are shown in Table 5. As shown in the table, the proposed app filter works well in recognizing samples from different types of apps. Specifically, a F-score of 96.62% is achieved in terms of recognizing interested apps. In the task of recognizing uninterested apps, a high recall of 93.15% is achieved. Although the precision of this task is relative low (61.82%), we argue that recall is more important than precision in this task. High recall means that most uninterested apps are



(a) Precision



(b) Recall



(c) F-score

Fig. 10. Performance of in-app activity classification (synthetic dataset).

rejected to avoid false information, and only a small fraction of the trajectory of interested apps are lost, which is tolerable in the system. In summary, an overall accuracy of 94.03% is achieved.

6.2.3. Performance of app recognition

We compare ActiveTracker and baseline methods on pure app recognition. This task only focuses on identifying apps from the traffic segments without considering the activities.

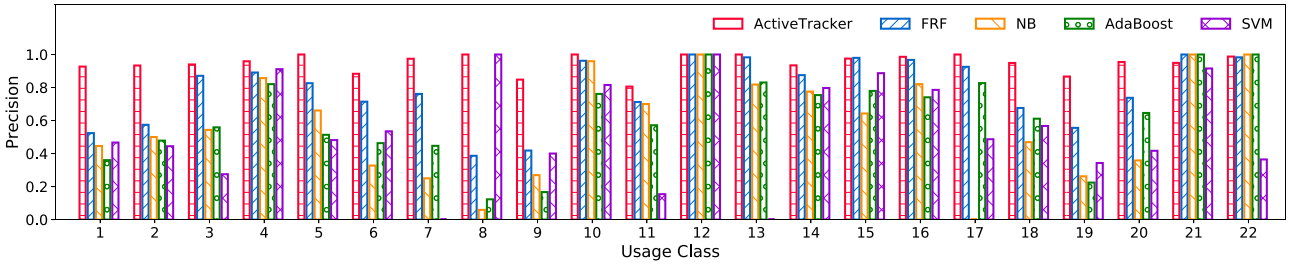
Fig. 7 a–c show the experiment results on real-world dataset in terms of precision, recall and F-score respectively. As shown in these figures, among all the classifiers, ActiveTracker achieves the highest precision, recall and F-score. As the state-of-the-art method, FRF is only good at recognizing QQ. Meanwhile, the other three shallow machine learning methods do not perform well on this task, and Naive Bayesian has the worst performance. In detail, the recall of recognizing Line achieved by Naive Bayesian is only 15.70%. In addition, all the baseline methods are not good at recognizing PayPal, Hangouts, Line and Messenger. In contrast, our model achieves the highest F-scores (over 93%) on recognizing all apps.

Fig. 8 shows the experimental results on the synthetic dataset. As can be seen, all the baseline methods perform much worse on the synthetic dataset. For example, on recognizing QQ, FRF is able to achieve a F-score of 98.02% on the real-wolrd dataset, while it can only achive a F-score of 75.91% on the synthetic dataset. In contrast, our model still maintain the highest recognition ability, achieving a F-score of over 90% for recognizing all apps.

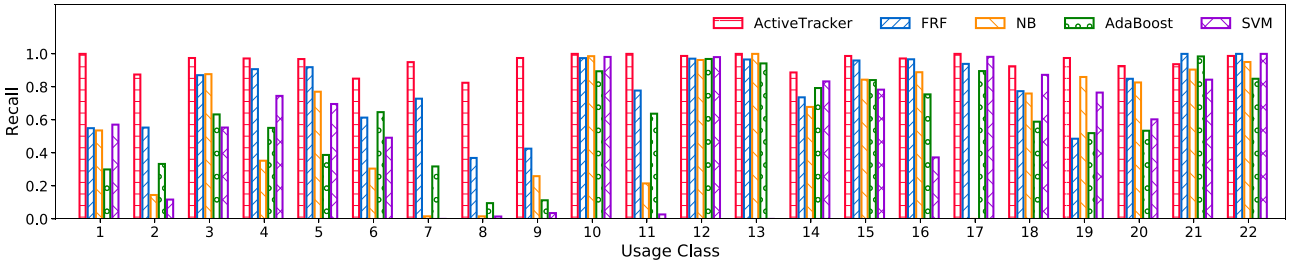
6.2.4. Performance of in-app activity recognition

We compare ActiveTracker and baseline methods on in-app activity recognition. This task focuses on identifying in-app activity from the traffic segment.

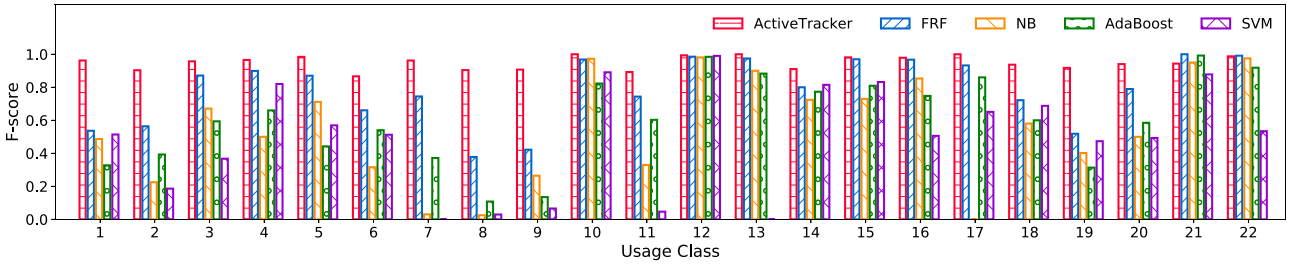
According to the experimental results on the real-world dataset shown in Fig. 9, FRF is expert in identifying video call, voice call, and location sharing, achieving the F-score of 99.19%, 95.65% and 93.23% respectively. However, FRF is not good at identifying other three activities. In detail, it only achieves a F-score of 77.64% on identifying transfer activity, a F-score of 81.99% on recognizing picture activity, and a F-score of 82.77% on identifying text activity. In this task, Naive Bayesian has the worst performance since it is not able to recognize location ac-



(a) Precision



(b) Recall



(c) F-score

Fig. 11. Performance of app-activity classification (real-world dataset).

tivity. AdaBoost Classifier and SVM are good at recognizing video call, but they both do not perform well on recognizing other five activities. In contrast, our model is able to recognize all activities with high precision and recall. In the task of recognizing voice call and location, although the performance of FRF is already very high, our model's performance is still higher, achieving the F-score of 98.13% and 100% respectively. This indicates that our proposed model is able to capture more unique features of the traffic generated by different activities compared with the baseline methods.

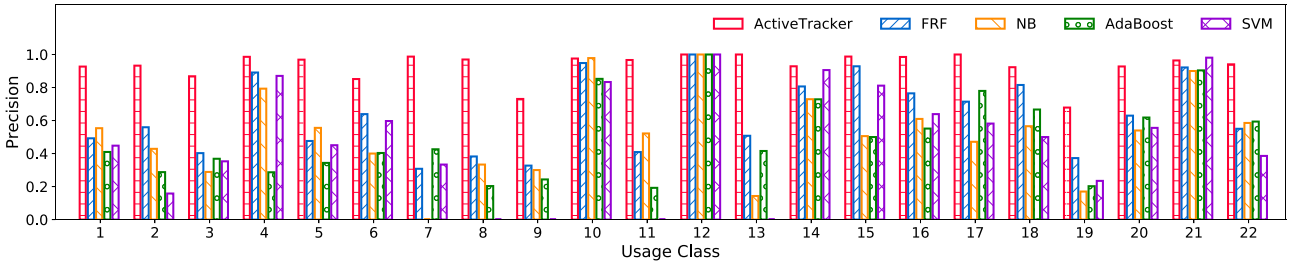
Fig. 10 presents the experimental results on the synthetic dataset. As presented in the figure, video call is the most recognizable activity, since all the methods achieves a high precision, a high recall and a high F-score on video call. However, all the baseline methods have worse performance on the synthetic dataset. Our proposed classifier achieves significant higher performance on synthetic dataset compared to the baselines, which indicates that the proposed model has good robustness.

6.2.5. *Performance of app-activity recognition*

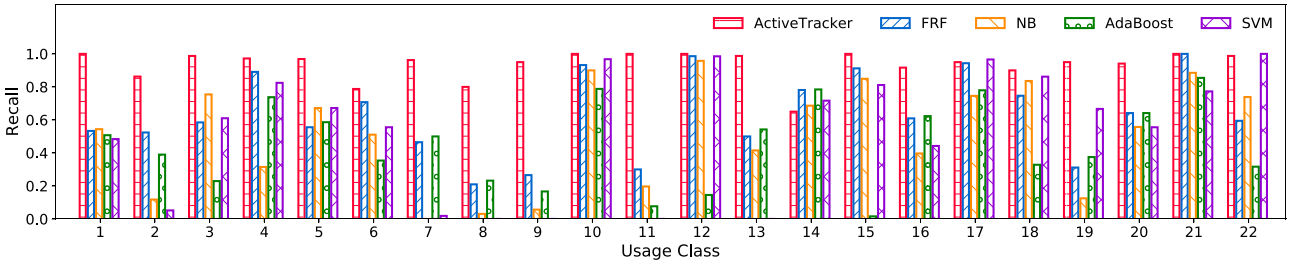
We compare ActiveTracker and baseline methods on recognizing both app and activity. This task is more difficult since it needs to identify the combination of apps and activities in a fine-grained level.

As the experimental results on real-world dataset shown in Fig. 11, our model again achieves high precision, recall and F-score for all app-activity classes. In contrast, no baseline methods perform well on this task. For example, none of the baseline methods can well recognizing sending text messages with Line (class 7), sending pictures with Messenger (class 8), sending text messages with Messenger (class 9). This is because the hand-crafted features used in FRF cannot fully reflect the minor differences between similar activities on different apps, which in turn indicates that our proposed model indeed has the ability to find unique patterns for individual app activity.

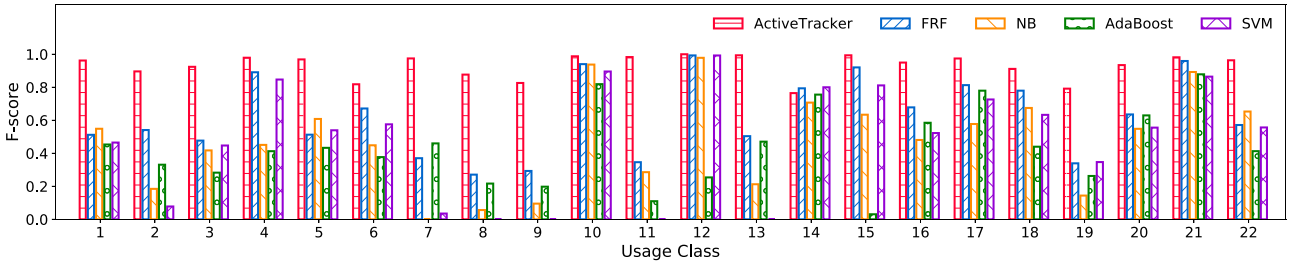
Particularly, we pay attention to the ability of recognizing the sensitive activities such as money transfer from the encrypted traffic. In Fig. 11, usage class 1 and 20 correspond to the money transfer activity using PayPal and WeChat accordingly. According to the figure, our model achieves a precision of 92.68% and a recall of 100% when identifying money transfer in PayPal, and a precision of 95.45% and a recall of 92.64% when identifying money transfer in WeChat. Among all the baseline methods, FRF has the best performance on recognzing money transfer activity. However, compared with our model, precision and recall achieved by FRF are much lower. In detail, it only achieves a precision of 52.38% and a recall of 55.0% when identifying money transfer in PayPal, and a precision of 73.77% and a recall of 84.91% when identifying money transfer in WeChat. The ability of precisely recognizing sensitive activities could enable malicious attackers to spy on people's money transfer by sniffing an access point in public space and cause a potential threat to people's property.



(a) Precision



(b) Recall



(c) F-score

Fig. 12. Performance of app-activity classification (synthetic dataset).

Fig. 12 shows the experimental results on the synthetic dataset. Comparing the results presented in Figs. 11 and 12, we observe that the baseline models are very sensitive to noises. For example, in Fig. 11, Adaboost Classifier have good perfomance on recognizing QQ's video call (class 12). However, in Fig. 12, its recall and F-score degrades significantly. Meanwhile, our proposed model still performs well on the synthetic dataset, which shows great robustness.

6.2.6. *Confusion matrix*

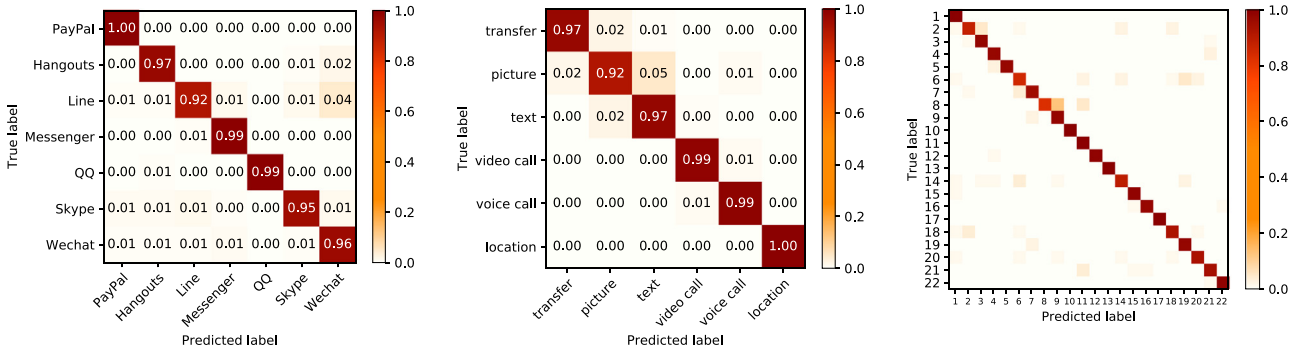
We draw the confusion matrices of the tasks of app recognition, in-app activity recognition, and app-activity recognition as illustrated in Fig. 13. As shown in Fig. 13a, we notice that most apps can be correctly identified except Line, where 8% of the traffic segments generated by Line are falsely classified, and 4% of them are recognized as the traffic segments generated by Wechat. Analysing the confusion matrix depicted in Fig. 13b, we realize that 8% of traffic segments generated by picture activity are incorrectly classified as the traffic segments generated by text activity. To find out the cause of misclassifications, we further analyse the confusion matrix shown in Fig. 13c. We notice that most misclassifications appear in recognizing *sending picture activity*. Specifically, most misclassification occurs in recognizing Hangout-picture (class 2), Line-picture (class 6), Messenger-picture (class 8) and Wechat-picture (class 18). A possible explanation is that within the same app, the traffic segments generated by sending picture activity are very similar to the traffic segments generated by sending text activity. For instance, our model classifies some samples from Hangouts-picture (class 2) as Hangouts-text (class 3), and identifies some samples from Messenger-picture (class 8) as Messenger-text (class 9).

Table 6
Overall accuracy (real-world dataset).

Classifier	App	Activity	App+Activity
ActiveTracker	**96.50**%	**96.71**%	**95.10**%
FRF	82.88%	89.47%	79.47%
Naive Bayesian	64.32%	80.60%	59.55%
AdaBoost Classifier	65.38%	84.09%	61.52%
SVM	57.27%	78.71%	54.09%

6.3. *Comparison of overall accuracy*

Table 6 shows the overall accuracy of our model and all the baseline methods on the real-world dataset for the three tasks mentioned above. As can be seen, compared with all the baseline methods, our proposed classifier achieves much higher overall accuracy on all three tasks with great improvement. Specifically, FRF achieves an overall accuracy of 82.88% on app recognition and an overall accuracy of 89.49% on in-app activity recognition. In addition, the performance of



(a) App recognition (b) In-app activity recognition (c) App-activity recognition

Fig. 13. Confusion matrices of three classification tasks (real-world dataset).

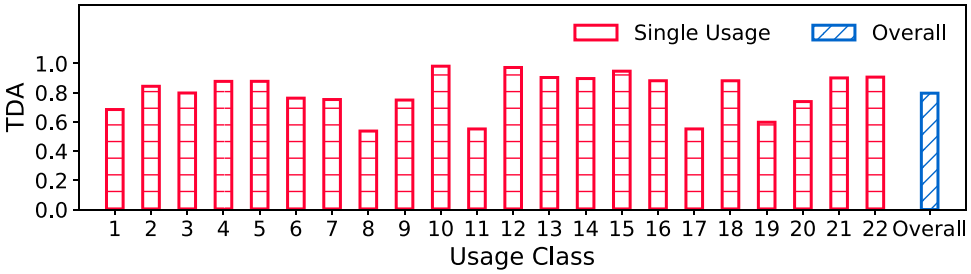


Fig. 14. Time duration accuracy of app trajectory recognition (real-world dataset).

Table 7
Overall accuracy (synthetic dataset).

Classifier	App	Activity	App+Activity
ActiveTracker	**94.54%**	**95.66%**	**93.20%**
FRF	66.97%	83.71%	63.33%
Naive Bayesian	54.17%	75.76%	50.83%
AdaBoost Classifier	50.00%	79.02%	45.68%
SVM	59.55%	78.56%	54.92%

other three baseline methods is even worse. For example, SVM only achieves an overall accuracy of 57.27% on app recognition task. On app-activity recognition task, the overall accuracy achieved by the baseline method is even lower (79.47% for FRF, 59.55% for Naive Bayesian, 61.52% for AdaBoost Classifier and 54.09% for SVM), which indicates that recognizing both app and activity is indeed a more complicated and challenging task. In contrast, the proposed ActiveTracker achieves a high overall accuracy on both app recognition task (96.50%) and in-app activity recognition task (96.71%). For the most challenging app-activity recognition task, our model can still achieve an overall accuracy of 95.10%, only slightly lower than the accuracy achieved in other two task. This illustrates that ActiveTracker has the ability to differentiate the similar activities on different apps with high accuracy.

As presented in Table 7, the experimental results on the synthetic dataset show that the accuracy achieved by all the baseline methods is much lower than that on the real-world dataset. In contrast, for all three recognition tasks, the overall accuracy achieved by our model is only slightly lower than the accuracy achieved on the real-world dataset. This shows that our model works well in a large-scale synthetic dataset with noise and diverse activities.

6.4. Performance of app trajectory recognition

Last we test the performance of recognizing continuous app trajectory from an encrypted traffic stream using the proposed approach. Time duration accuracy (TDA) is adopted as the evaluation metric which evaluates the total time durations that are correctly labeled. TDA is formally formulated as follows:

$$TDA = \frac{length(F \cap \hat{F})}{length(F)} \tag{6}$$

where $length(F)$ denotes the duration time of the whole traffic stream, and $length(F \cap \hat{F})$ captures the time durations (segments) in which the app activity is correctly identified. The results are shown in Fig. 14. As can be seen, a high time duration accuracy is achieved for most app activities. Specifically, video calls (class 10, 12, 15, 21) and voice calls (class 13, 22) can be easily uncovered with TDA of over 90%. As for uncovering money transfer activity, our framework achieves a TDA of 68.46% on Paypal, and a TDA of 90.09% on Wechat, which indicates that our framework indeed has the ability to uncover the sensitive activity such as money transfer over an encrypted traffic stream. Overall, our mothod achieves a TDA of 79.65% on app trajectory recognition.

6.5. Scalability of DNN model

Last we test the scalability of our proposed DNN model. The goal of this experiment is to check whether significant performance degradation will occur when more and more apps are added into the model. We rerun the experiments in 6.2.3 (app recognition), 6.2.4 (activity recognition), 6.2.5 (app activity recognition) with different number of apps (varying from 2 apps to 7 apps) and test the overall accuracy of each task.

As presented in Fig. 15, no significant performance degradation is observed when the number of apps increases from 2 to 7. In detail, when recognizing both app and activity with only 2 apps, our DNN model is able to achieve a very high overall accuracy of 97.06%. For the same recognition task with 7 apps, the overall accuracy achieved by our DNN model is still high (95.10%). We can expect that the model will maintain the same level of accuracy if the number of apps are small, but we cannot promise the performance on a large number of apps (e.g., 100 apps). Theoretically the accuracy will decrease since a large number of apps will increase the difficulty of classification. Note that the attacker will

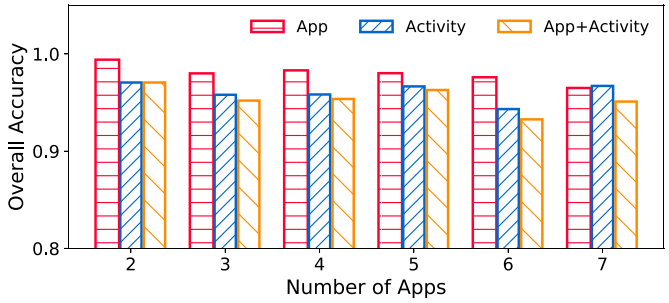


Fig. 15. Scalability of DNN model (real-world dataset).

mainly focus on attacking a few number of apps, so the proposed model works well on revealing activities from a small number of apps.

7. Conclusion

In this paper, we propose *ActiveTracker*, a framework to recognize app trajectory over encrypted Internet traffic streams. First, the incoming Internet traffic of mobile apps is segmented into several single-activity subsequences by a sliding window based approach. Then, each traffic segment is represented by a temporal-spacial traffic matrix and a traffic spectrum vector. Using the normalized data as input, we propose a deep neural network (DNN) model to combine the features from different domains for app trajectory recognition. Extensive experiments based on real-world encrypted mobile traffic as well as the synthetic traffic show that the proposed framework achieves high accuracy in app trajectory recognition. Our work will rise people's attention on the privacy protection of mobile app communications.

Declaration of Competing Interest

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

Acknowledgements

This work was partially supported by the National Key R&D Program of China (Grant No. 2017YFB1001801), the National Natural Science Foundation of China (Grant Nos. 61972196, 61672278, 61832008, 61832005), the open Project from the State Key Laboratory of Smart Grid Protection and Operation Control "Research on Smart Integration of Terminal-Edge-Cloud Techniques for Pervasive Internet of Things", the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Sino-German Institutes of Social Computing.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107372.

References

[1] Mobile vs. desktop usage in 2019.
[2] K.P. Dyer, S.E. Coull, T. Ristenpart, T. Shrimpton, Peek-a-boo, i still see you: why efficient traffic analysis countermeasures fail, in: IEEE Symposium on Security and Privacy (S & P), 2012, pp. 332–346.
[3] A. Mendoza, G. Gu, Mobile application web API reconnaissance: web-to-mobile inconsistencies & vulnerabilities, in: IEEE Symposium on Security and Privacy (S & P), 2018, pp. 756–769.
[4] G. Yang, J. Huang, G. Gu, A. Mendoza, Study and mitigation of origin stripping vulnerabilities in hybrid-postmessage enabled mobile applications, in: IEEE Symposium on Security and Privacy (S & P), 2018, pp. 742–755.
[5] A. Possemato, A. Lanzi, S.P.H. Chung, W. Lee, Y. Fratantonio, Clickshield: are you hiding something? Towards eradicating clickjacking on android, in: ACM SIGSAC Conference on Computer and Communications Security (CCS), 2018, pp. 1120–1136.
[6] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, Y. Zhang, Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms, in: USENIX Security Symposium, 2019, pp. 1133–1150.
[7] T. Karagiannis, A. Broido, M. Faloutsos, K.C. Claffy, Transport layer identification of p2p traffic, in: ACM SIGCOMM Internet Measurement Conference (IMC), 2004, pp. 121–134.
[8] S. Sen, O. Spatscheck, D. Wang, Accurate, scalable in-network identification of p2p traffic using application signatures, in: International Conference on World Wide Web (WWW), 2004, pp. 512–521.
[9] A. Pham, I. Dacosta, E. Losiouk, J. Stephan, K. Huguenin, J. Hubaux, Hidemyapp: hiding the presence of sensitive apps on android, in: USENIX Security Symposium, 2019, pp. 711–728.
[10] M. Shen, M. Wei, L. Zhu, M. Wang, Classification of encrypted traffic with second-order Markov chains and application attribute bigrams, IEEE Trans. Inf. Forensics Secur. 12 (8) (2017) 1830–1843.
[11] S. Sengupta, N. Ganguly, P. De, S. Chakraborty, Exploiting diversity in android TLS implementations for mobile app traffic classification, in: The World Wide Web Conference (WWW), 2009, pp. 1657–1668.
[12] Y. Fu, H. Xiong, X. Lu, J. Yang, C. Chen, Service usage classification with encrypted internet traffic in mobile messaging apps, IEEE Trans. Mob. Comput. (TMC) 15 (11) (2016) 2851–2864.
[13] J. Liu, Y. Fu, J. Ming, Y. Ren, L. Sun, H. Xiong, Effective and real-time in-app activity analysis in encrypted internet traffic streams, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2017, pp. 335–344.
[14] T. Bao, H. Cao, E. Chen, J. Tian, H. Xiong, An unsupervised approach to modeling personalized contexts of mobile users, in: IEEE International Conference on Data Mining (ICDM), 2010, pp. 38–47.
[15] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, D. Song, Networkprofiler: towards automatic fingerprinting of android apps, in: IEEE International Conference on Computer Communications (INFOCOM), 2013, pp. 809–817.
[16] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, Z.M. Mao, Samples:self adaptive mining of persistent lexical snippets for classifying mobile application traffic, in: ACM International Conference on Mobile Computing and Networking (MobiCom), 2015, pp. 439–451.
[17] Y. Chen, X. Jin, J. Sun, R. Zhang, Y. Zhang, Powerful: mobile app fingerprinting via power analysis, in: IEEE Conference on Computer Communications (INFOCOM), 2017, pp. 1–9.
[18] C. Liu, Z. Cao, G. Xiong, G. Gou, S. Yiu, L. He, Mampf: encrypted traffic classification based on multi-attribute Markov probability fingerprints, in: IEEE/ACM International Symposium on Quality of Service (IWQoS), 2018, pp. 1–10.
[19] S. Xu, L. Zhang, A. Li, X. Li, C. Ruan, W. Huang, Appdna: app behavior profiling via graph-based deep learning, in: IEEE Conference on Computer Communications (INFOCOM), 2018, pp. 1475–1483.
[20] Y. Fu, J. Liu, X. Li, H. Xiong, A multi-label multi-view learning framework for in-app service usage analysis, ACM Trans. Intell. Syst. Technol. 9 (4) (2018) 1–24.
[21] D. Li, W. Li, X. Wang, C. Nguyen, S. Lu, Activetracker: uncovering the trajectory of app activities over encrypted internet traffic streams, in: IEEE International Conference on Sensing, Communication, and Networking (SECON), 2019, pp. 1–9.
[22] Y. Ye, W. Zeng, Q. Shen, X. Zhang, Y. Lu, The visual quality of streets: a human–centred continuous measurement based on machine learning algorithms and street view images, Environ. Plann. B 46 (8) (2019) 1439–1457.
[23] F. Zhang, L. Wu, D. Zhu, Y. Liu, Social sensing from street-level imagery: a case study in learning spatiotemporal urban mobility patterns, ISPRS J. Photogramm. Remote Sens. 153 (2019) 48–58.
[24] F. Zhang, D. Zhang, Y. Liu, H. Lin, Representing place locales using scene elements, Comput. Environ. Urban Syst. 71 (2018) 153–164.
[25] D. Anguelov, C. Dulong, D. Filip, C. Früh, S. Lafon, R. Lyon, A.S. Ogale, L. Vincent, J. Weaver, Google street view: capturing the world at street level, IEEE Comput. 43 (6) (2010) 32–38.
[26] A.B. Adege, H. Lin, L. Wang, Mobility predictions for IoT devices using gated recurrent unit network, IEEE Internet Things J. 7 (1) (2020) 505–517.
[27] L. Wang, L. Zhang, Z. Yi, Trajectory predictor by using recurrent neural networks in visual tracking, IEEE Trans. Cybern. 47 (10) (2017) 3172–3183.
[28] K. Lin, M. Chen, J. Deng, M.M. Hassan, G. Fortino, Enhanced fingerprinting and trajectory prediction for IoT localization in smart buildings, IEEE Trans. Autom. Sci. Eng. 13 (3) (2016) 1294–1307.
[29] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, Y. Guan, Network traffic classification using correlation information, IEEE Trans. Parallel Distrib.Syst. 24 (1) (2013) 104–117.
[30] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, G. Wei, L.T. Yang, Internet traffic classification using constrained clustering, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2932–2943.
[31] M. Lotfollahi, R.S.H. Zade, M.J. Siavoshani, M. Saberian, Deep packet: a novel approach for encrypted traffic classification using deep learning, CoRR (2017) abs/1709.02656.
[32] J. Kohout, T. Pevný, Network traffic fingerprinting based on approximated kernel two-sample test, IEEE Trans. Inf. Forensics Secur. 13 (3) (2018) 788–801.
[33] R. Li, X. Xiao, S. Ni, H. Zheng, S. Xia, Byte segment neural network for network traffic classification, in: 26th IEEE/ACM International Symposium on Quality of Service (IWQoS), 2018, pp. 1–10.
[34] Y. Dong, J. Zhao, J. Jin, Novel feature selection and classification of internet video traffic based on a hierarchical scheme, Comput. Netw. 119 (2017) 102–111.
[35] K.L. Dias, M.A. Pongelupe, W.M. Caminhas, L. de Errico, An innovative approach for real-time network traffic classification, Comput. Netw. 158 (2019) 143–157.
[36] Z. Wu, Y. Dong, H. Wei, W. Tian, Consistency measure based simultaneous feature se-

lection and instance purification for multimedia traffic classification, Comput. Netw. 173 (2020).
[37] B. Anderson, D.A. McGrew, Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2017, pp. 1723–1732.
[38] Q. Xu, Y. Liao, S. Miskovic, Z.M. Mao, M. Baldi, A. Nucci, T. Andrews, Automatic generation of mobile app signatures from traffic observations, in: IEEE International Conference on Computer Communications (INFOCOM), 2015, pp. 1481–1489.
[39] G. Ranjan, A. Tongaonkar, R. Torres, Approximate matching of persistent lexicon using search-engines for classifying mobile app traffic, in: IEEE International Conference on Computer Communications (INFOCOM), 2016, pp. 1–9.
[40] V.F. Taylor, R. Spolaor, M. Conti, I. Martinovic, Robust smartphone app identification via encrypted network traffic analysis, IEEE Trans. Inf. Forensics Secur. 13 (1) (2017) 63–78.
[41] M. Conti, L.V. Mancini, R. Spolaor, N.V. Verde, Analyzing android encrypted network traffic to identify user actions, IEEE Trans. Inf. Forensics Secur. 11 (1) (2016) 114–125.
[42] C. Liu, L. He, G. Xiong, Z. Cao, Z. Li, Fs-net: a flow sequence network for encrypted traffic classification, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 1171–1179.
[43] Y. Chen, T. Zang, Y. Zhang, Y. Zhou, Y. Wang, Rethinking encrypted traffic classification: a multi-attribute associated fingerprint approach, in: 27th IEEE International Conference on Network Protocols (ICNP), 2019, pp. 1–11.
[44] M. Korczynski, A. Duda, Markov chain fingerprinting to classify encrypted traffic, in: IEEE Conference on Computer Communications (INFOCOM), 2014, pp. 781–789.
[45] Q. Xu, J. Erman, A. Gerber, Z.M. Mao, J. Pang, S. Venkataraman, Identifying diverse usage behaviors of smartphone apps, in: ACM SIGCOMM Internet Measurement Conference (IMC), 2011, pp. 329–344.
[46] Y. Fu, J. Liu, X. Li, X. Lu, J. Ming, C. Guan, H. Xiong, Service usage analysis in mobile messaging apps: a multi-label multi-view perspective, in: F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, Z. Zhou, X. Wu (Eds.), 16th IEEE International Conference on Data Mining (ICDM), 2016.
[47] J.M. Joyce, Kullback-Leibler Divergence, Springer, pp. 720–722.
[48] D. Salomon, Curves and Surfaces for Computer Graphics, Springer, 2006.
[49] J. Unpingco, Python for Signal Processing, Springer, 2014.
[50] J.S. Bridle, Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters, in: Advances in Neural Information Processing Systems (NIPS), 1989, pp. 211–217.
[51] A.R. Dhamija, M. Günther, T.E. Boult, Reducing network agnostophobia, in: Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 9157–9168.
[52] A.V. Balakrishnan, R.R. Mazumdar, On powers of gaussian white noise, IEEE Trans. Inf. Theory 57 (11) (2011) 7629–7634.
[53] Pytorch. [Online]. Available: https://pytorch.org/.
[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: an imperative style, high-performance deep learning library, in: Annual Conference on Neural Information Processing Systems (NeurIPS), 2019, pp. 8024–8035.
[55] T.F. Chan, G.H. Golub, R.J. LeVeque, Updating formulae and a pairwise algorithm for computing sample variances, in: COMPSTAT 5th Symposium Held at Toulouse, 1982, pp. 30–41.
[56] J. Zhu, H. Zou, S. Rosset, T. Hastie, Multi-class adaboost, Stat. Interface 2 (2009) 349–360.
[57] C.J.C. Burges, B. Schölkopf, Improving the accuracy and speed of support vector machines, in: Advances in Neural Information Processing Systems(NIPS), 1996, pp. 375–381.

Ding Li received his B.E. degree in computer science and technology from Beijing University of Posts and Telecommunications and his M.S. degree in computer science from University of California, Santa Cruz. He is now a Ph.D. candidate in the Department of Computer Science and Technology, Nanjing University. His research interests include wireless network measurement, mobile computing, and deep learning.

Wenzhong Li receives his B.S. and Ph.D degree from Nanjing University, China, both in computer science. He was an Alexander von Humboldt Scholar Fellow in University of Goettingen, Germany. He is now a full professor in the Department of Computer Science, Nanjing University. Dr. Li's research interests include distributed computing, data mining, mobile cloud computing, wireless networks, pervasive computing, and social networks. He has published over 100 peer-review papers at international conferences and journals, which include INFOCOM, UBICOMP, IJCAI, ACM Multimedia, ICDCS, IEEE Communications Magazine, IEEE/ACM Transactions on Networking (ToN), IEEE Journal on Selected Areas in Communications (JSAC), IEEE Transactions on Parallel and Distributed Systems (TPDS), IEEE Transactions on Wireless Communications (TWC), etc. He served as Program Co-chair of MobiArch 2013 and Registration Chair of ICNP 2013. He was the TPC member of several international conferences and the reviewer of many journals. He is the principle investigator of three fundings from NSFC, and the co-principle investigator of a China-Europe international research staff exchange program. Dr. Li is a member of IEEE, ACM, and China Computer Federation (CCF). He was also the winner of the Best Paper Award of ICC 2009 and APNet 2018.

Xiaoliang Wang received the Ph.D. degree from the Graduate School of Information Sciences, Tohoku University, Japan, in 2010. He is currently an Associate Professor with the Department of Computer Science and Technology, Nanjing University. His research interests include network system design and human–computer interaction.

Cam-Tu Nguyen obtained her bachelor and master degrees from Vietnam National University in 2005 and 2008, respectively. She received her Ph.d. in information science from Tohoku, University, Japan in 2011. From 2012 to 2016, she worked as Postdoctor at Nanjing University, China. She has published several papers in leading journals and conferences such as IEEE TKDE, ACM Tweb, AAAI, IJCAI, CIKM. Her research interest includes Machine Learning, Natural Language Understanding.

Sanglu Lu received her B.S., M.S., and Ph.D. degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the deputy director of State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, pervasive computing, and wireless networks. She has published more than 100 papers in referred journals and conferences in the above areas. She was the principle investigator of many national fundings including National Key R&D Program of China, the National Natural Science Foundation of China, the Key R&D Program of Jiangsu Province, China, etc. She is a member of IEEE and ACM.