# ActiveTracker: Uncovering the Trajectory of App Activities over Encrypted Internet Traffic Streams

Ding Li, Wenzhong Li, Xiaoliang Wang, Cam-Tu Nguyen, Sanglu Lu
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Email: liding@smail.nju.edu.cn, lwz@nju.edu.cn

***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 user's 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) classification algorithm is developed to recognize the crucial activities conducted with different apps by the user. We show by extensive experiments on real-world app usage traffic collected from volunteers that the proposed approach achieves up to 78.5% accuracy in recognizing app trajectory over encrypted traffic streams.**

***Index Terms*—Time Series Segmentation; Encrypted Internet Traffic; Mobile App and Service Classification.**

## I. 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 2017, mobile apps accounted for 2/3 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][3][4]. For example, the traffic classification techniques [5][6][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. 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. 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 [9][10][11] showed that, the information of mobile app usage can be inferred by examining the temporal-spacial patterns of the encrypted Internet traffic packets.

This work was partially supported by the National Key R&D Program of China (Grant No. 2018YFB1004704), the Key R&D Program of Jiangsu Province, China (Grant No. BE2018116), the National Natural Science Foundation of China (Grant Nos. 61672278, 61832008, 61832005), the science and technology project from State Grid Corporation of China (Contract No. SGSNXT00YJJS1800031), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Sino-German Institutes of Social Computing. The corresponding author is Wenzhong Li (Email: lwz@nju.edu.cn) and Sanglu Lu (sanglu@nju.edu.cn).

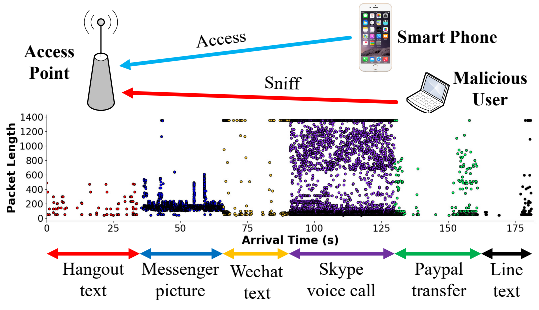


Fig. 1. An Example of App Trajectory Recognition.

The works of app fingerprinting [12][13][14][15][16] tend to establish unique features for app distinction. The features are extracted from the traffic level, code level, and system level. For example, NetworkProfiler [12] automatically generated network profiles for identifying Android apps according to the HTTP headers in the traffic. AppPrint [13] used parts of the HTTP URLs or strings from HTTP headers as a fingerprint. AppDNA [17] inspected the function-call-graph to form app fingerprint in the code-level. POWERFUL [16] fingerprinted mobile apps by analyzing their power consumption patterns in the system-level.

Recently, several works focus on in-app activity classification that aims to recognize the usage of different services within a particular app such as Whatsapp [9][10][18]. Fu et al. proposed in-app activity classification by jointly modeling user behavioral patterns, network traffic characteristics, and temporal dependencies. In their follow-up work [10], they improved the processing speed of this classifier by selecting most discriminative traffic patterns to meet the online efficiency requirement. A multi-label multi-view logistic classification method was developed in [18] to overcome the pain of mixed-usage traffic flows. However, the existing works focus on identifying the activity within a particular app, and they lack the ability to recognize both app and in-app activity in a fine-grained level.

In this paper, 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 explained below. First, the conventional approaches were designed for recognizing either app or activity (service), 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 V). Second, 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 V. Third, to uncover the app trajectory from a continuous encrypted traffic stream, it needs a method to correctly partition the traffic stream into segments representing different user activities, which has not been well studied in the past.

To address these challenges, we propose ActiveTracker, a novel framework to uncover the trajectory of app activities from the encrypted Internet traffic stream. 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) classifier is proposed for activity recognition. Combining the recognition results of the sequence of traffic segments, the trajectory of app activities can be uncovered, which may lead to the leakage of sensitive personal information of the mobile users.

Tabel I 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.

TABLE I
COMPARISON OF ACTIVETRACKER AND THE EXISTING WORKS.

| | App Recognition | Activity Recognition | 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 learning |

- 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 sub-streams.
- We propose a DNN-based classifier for activity recognition from traffic segments. The proposed classifier uses convolutional neural network to extract features from the traffic segmentation automatically, and achieves high accuracy in activity recognition.
- To the best of our knowledge, we are the first to solve the problem of uncovering the trajectory of app activities 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 78.5% accuracy in uncovering app activity trajectory from a long traffic stream. Our work will draw people's attention to privacy protection of mobile app communications.

## II. Problem Formulation

### A. Adversary Model

We consider the scenario that an adversary aims to uncover the trajectory of app activities 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.

### B. 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 $F = \{p_1, p_2, \cdots, p_N\}$, where $p_i$ $(1 \leq i \leq N)$ is the information of the $i$-th observed packet represented by $p_i = \langle T_i, L_i \rangle$, where $T_i$ is the timestamp of the packet and $L_i$ is the length of the packet.

We assume the adversary can observe only very limited information from the encrypted packet, i.e., only the timestamp and packet length 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, $F(T_i, T_j) = \{p_i, p_{i+1}, \cdots, p_j\}$ 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 seconds. Therefore the length of $F(T_i, T_j)$ should be longer than 15 seconds. 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: $F \rightarrow \langle T_1, app_1, activity_1 \rangle, \langle T_2, app_2, activity_2 \rangle, \cdots, \langle T_K, app_K, activity_K \rangle$. In other words, it aims to uncover which app is used for what activity at some moment by analysing the encrypted Internet traffic stream.

### C. Problem Statement

Given a sniffed encrypted Internet traffic stream, the goal of the attacker is to uncover the trajectory of the app activities 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 usage trajectory. Specifically, the app usage trajectory recognition problem can be solved by considering the following sub-problems:

*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.

*2) Encrypted Traffic Segment Classification:* Given an encrypted traffic segments, 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 usage trajectory from the encrypted Internet traffic stream based on Deep Neural Network (DNN).

## III. App Trajectory Recognition based on DNN

In this section, we first give a brief overview 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 traffic segmentation, a method for data representation, and the proposed DNN classifier for encrypted traffic segment classification.

Fig. 2 shows our solution framework, which consists of three major components.

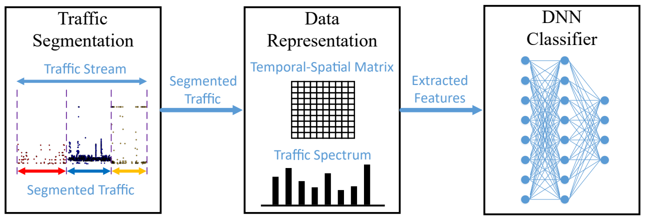


Fig. 2. The Solution Framework.

### A. Solution Framework

*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.

*2) Data Representation:* Given the traffic segments, we transform them into normalized data as the input of the DNN classifier. Specifically, we represent each traffic segment by a temporal-spacial traffic matrix and a traffic spectrum vector.

*3) Segment Classification:* Using the normalized data as input, we propose a DNN model for 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 activity and uncover the app usage trajectory.

The three components are presented in detail below.

### B. Sliding Window based Traffic Segmentation

We propose a novel sliding window based approach for traffic segmentation. Given a continuous encrypted traffic stream, the task of traffic segmentation is to find the "split points" that partition the stream into segments corresponding to different app activities. Intuitively, different 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 formulation, similarity curve generation, low-pass filtering, and split points identification, which are explained below.

*1) Sliding Window Formulation:* The sliding window approach 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 as introduced below.

*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* [19] as the distance 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 and right parts accordingly. 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.

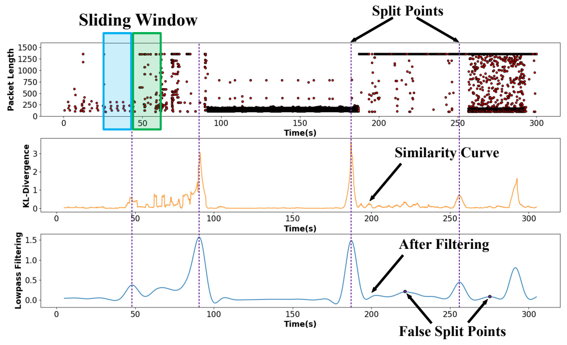


Fig. 3. Illustration of the Sliding Window Approach.

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

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

We move the sliding window step by step along the encrypted traffic stream, and compute the similarities 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 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.

*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 high frequency noise.

In our work, we adopt a cutoff frequency of 0.015Hz, since an app activity should last for at least for 15 seconds.

*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, such peaks are small, and most of their values 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 a heuristic approach for split points identification: we first detect the extreme points (maximum) on the similarity curve, and then compare the detected points with a predefined threshold (e.g., 0.2). If the similarity value on the extreme point is greater than the threshold, it is identified as a split point.

Using the above heuristics, we can obtain a set of split points which partition the continuous traffic stream into segments.

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

**Algorithm 1** Sliding Window based Traffic Segmentation

```
Input:
    An encrypted Internet Traffic Stream $F = \{p_1, p_2, \cdots, p_N\}$.
Output:
    Split points $P = \{t_j, j = 1, 2, \cdots, M\}$ such that the packets between two adjacent split points are generated by a single app activity.
1:  initial $P = \varnothing$ and $C = \varnothing$;
2:  initial the length of time window $L = 10s$, and set the sliding window at the beginning of the traffic stream
3:  repeat
4:      compute the similarity $s = Sim(S_l, S_r)$;
5:      add $s$ to $C$;
6:      move the sliding window rightwards 0.1 second;
7:  until no packet is included by the sliding window;
8:  generate the similarity curve $FC$ using elements in $C$;
9:  feed $FC$ into a low-pass filter, obtaining a smooth similarity curve $SC$;
10: identify the split points in $SC$ and add them to $P$;
11: return $P$;
```

## C. Data Representation

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

Since a user activity is assumed to be longer than 15 seconds, the segments with length less than 15 seconds 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* and a *traffic spectrum vector*.

The temporal-spacial traffic matrix $M$ is a $150 \times 150$ 2D matrix, where each element $M_{ij}$ $(1 \leq i \leq 150, 1 \leq j \leq 150)$ corresponds to the number of packets which arrive between $(i/10 - 0.1)$ second and $(i/10)$ second and whose lengths are between $(j * 100 - 99)$ bytes and $(j * 100)$ bytes. The quantification of $M$ tends to capture the temporal-spacial correlations of the encrypted traffic stream.

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 function. Obviously, these samplings are generated with unstable time intervals. To obtain the traffic spectrum, we first conduct Hermite interpolation [20] to obtain equally spaced (0.01 second) data sequence. Then, by using *Discrete-Time Fourier Transform (DTFT)* [21], we can obtain a vector of length 1500 which contains frequency domain features of the extracted traffic stream.

### D. Deep Neural Network Classifier

Inspired by the fact that 2D CNN is expert in image recognition and that images are usually represented by matrices, we believe that 2D CNN can also be ultilized to extract features of a traffic stream from its temporal-spacial traffic matrix. In addition, since 1D CNN is good at extracting features from sequential data, we can also use it to extract features from traffic spectrum vector.

Similar to other deep learning models, feature extraction is vital in CNN. Specifically, features extracted in shallower layers of the CNN will be fed to the successive convolutional layers in order to extract more abstract features. Thus, compared with hand-crafted features, the features extracted by CNN are more comprehensive, which would be a great help to differentiating similar activities on different apps.

Considering the above mentioned advantages of CNN, we propose a deep neural network (DNN) classifier for app activity identification. As illustrated in Fig. 4, the proposed architecture consists of a 2D CNN and a 1D CNN. 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. Finally, the output of the fully-connected layer is fed into a softmax layer, and the softmax layer outputs a probability distribution of app usage. The usage with the highest probability are chosen as the predicted usage.

Table II describes the main parameters of each layer of the proposed DNN classifier.

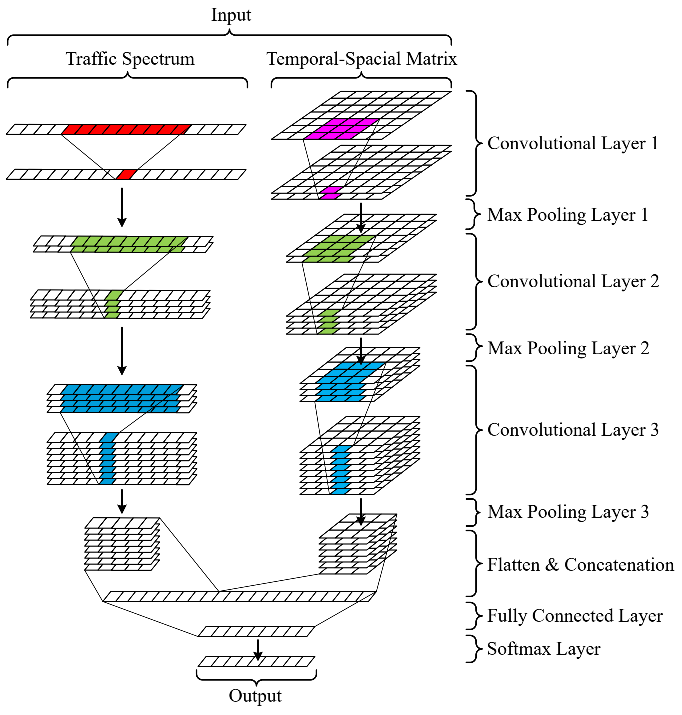


Fig. 4. The Structure of the DNN Classifier.

TABLE II
THE CONFIGURATION OF DNN CLASSIFIER.

| **Network Configuration** | |
|---|---|
| Input (1*1500, frequency domain) | Input (150*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 | |
| Softmax | |

## IV. DATA COLLECTION

To evaluate the performance of our framework, we collect real-world data of encrypted Internet traffic from 7 popular mobile apps: Paypal, Hangouts, Line, Messenger, QQ, Skype and Wechat. We use these apps to conduct different activities including location (sharing the current location of the user), picture (posting a photo), text (sending text to a friend), video call, voice call, and money transfer. The apps and activities for data collection are summarized in Table III.

To collect traffic flow data, we recruited 3 volunteers who are required to use these apps with our specially configured handsets. Specifically, we use Samsung Galaxy S7 and Huawei G9 Youth as our experimental handsets. In addition, we set up a firewall on the handset which grants Internet permission to the target apps and blocks the traffic from other apps. The smartphones are connected to a virtual Wi-Fi access point (AP) operated by a laptop. A well-known sniffing tool, WireShark, is run on the laptop to crawl the packet information of the smartphones from the AP.

TABLE III
THE INTERESTED APP AND ACTIVITY TYPES.

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

During data collection, the volunteer conducted a particular activity with an app for a duration, and labeled the activity in the dataset. For example, a volunteer may use WeChat to text with others in the first hour, and then use Skype to make a video call in the next hour. Their activities and time durations are logged as ground truth. The collected traffic is labeled to form a training set for model learning and a test set for performance verification.

To further ensure the robustness of our framework, besides collecting traffic data in our lab, we also collect data under different network environments including in the library and student dormitory.

The basic statistics of the collected app activity dataset are illustrated in Tabel IV.

TABLE IV
STATISTICS OF THE COLLECTED INTERNET TRAFFIC.

| # | app | Activity | Rec. | Packets | Traffic | Tra/min |
|---|---|---|---|---|---|---|
| 1 | PayPal | transfer | 182 | 33.6K | 12.9M | 231K |
| 2 | Hangouts | picture | 265 | 1065.9K | 785.1M | 3.39M |
| 3 | Hangouts | text | 468 | 71.8K | 18.38M | 63.2K |
| 4 | Hangouts | video call | 162 | 257.1K | 123.4M | 3.04M |
| 5 | Hangouts | voice call | 162 | 90K | 30.76M | 777.1K |
| 6 | Line | picture | 400 | 307.4K | 200.7M | 845.3K |
| 7 | Line | text | 340 | 32.7K | 5.42M | 18.68K |
| 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 | 269 | 903K | 584.8M | 8.69M |
| 13 | QQ | voice call | 262 | 219.3K | 37.86M | 564.59K |
| 14 | Skype | picture | 296 | 102.8K | 54.72M | 397.4K |
| 15 | Skype | video call | 196 | 691.8K | 441.9M | 9.01M |
| 16 | Skype | voice call | 164 | 239.8K | 40.47M | 0.99M |
| 17 | Wechat | location | 673 | 276.3K | 156.3M | 777.4K |
| 18 | Wechat | picture | 446 | 475.8K | 306.65M | 852.7K |
| 19 | Wechat | text | 617 | 41.7K | 8.11M | 23.04K |
| 20 | Wechat | transfer | 142 | 19.6K | 6.69M | 47.67K |
| 21 | Wechat | video call | 243 | 690.5K | 140.5M | 2.3M |
| 22 | Wechat | voice call | 301 | 514.9K | 85.83M | 1.13M |

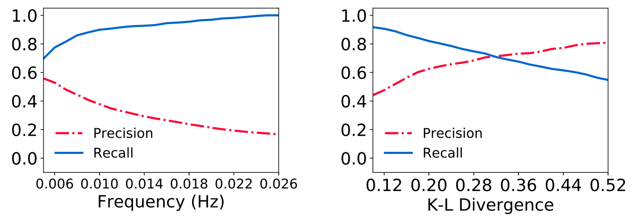
## V. PERFORMANCE EVALUATION

To validate the effectiveness of our proposed framework, we conducted extensive experiments on the collected traffic data. All experiments were performed on a single machine with Intel Core i7-7700 processors (4 cores / 8 threads), 8 GB of memory, and NVIDIA GeForce GTX 1050 GPU.

### A. Performance Metrics and Baselines

Similar to the work of [9][10], 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 [22], and they are omitted due to page limit.

In order to confirm the effectiveness of our proposed DNN classifier, we compare our classifier with the random forest classifier with fitering called FRF in [10], which is the state-of-art approach for in-app activity recognition.



(a) Influence of Cutoff Frequency. (b) Influence of Threshold.

Fig. 5. The Performance of Traffic Segmentation under Different System Parameters.

### B. Performance Analysis

*1) Performance of the Traffic Segmentation Algorithm:* We first show the performance of the proposed sliding window based traffic segmentation algorithm. 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.

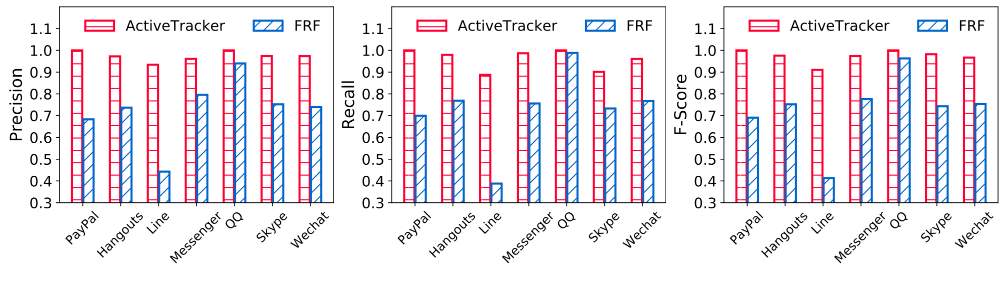
Fig. 5(a) shows the performance under different cutoff frequencies. With the increase of cutoff frequency, the recall approaches 1, which means almost all split points can be identified by the algorithm. However, the precision decreases with the increase of cutoff frequency, which means more false split points are included. 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. 5(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 usage.

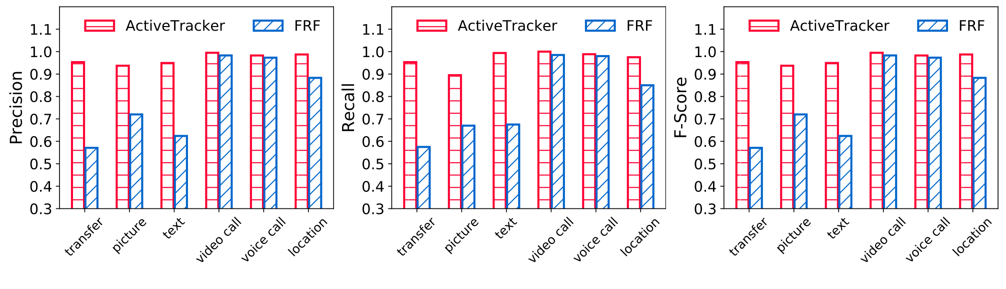
*2) Performance of App Recognition:* We compare Active-Tracker and FRF on pure app recognition. This task only focuses on identifying app from the traffic segment without considering the activities.

Fig. 6(a), 6(b), and 6(c) show the experiment results in terms of precision, recall and f-score respectively. As shown in these figures, our model achieves high precision, recall and f-score on recognizing all apps, while FRF is only good at recognizing QQ app. Both models have the worst performance



(a) Precision (b) Recall (c) F-score

Fig. 6. Performance of App Classification.



(a) Precision (b) Recall (c) F-score

Fig. 7. Performance of In-app Activity Classification.

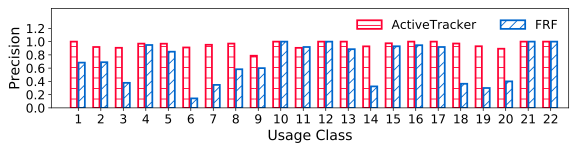
on recognizing Line app among all seven apps. This is because some traffic streams generated by the Line app is very similar to the traffic generated by the Wechat app. However, under such circumstances, our model still achives a relatively high f-score of 91.03% on recognizing Line, while FRF only achives a low f-score of 41.30%.

*3) Performance of In-app Activity Recognition:* We compare ActiveTracker and FRF on in-app activity recognition. This task focuses on identifying in-app activity from the traffic segment.

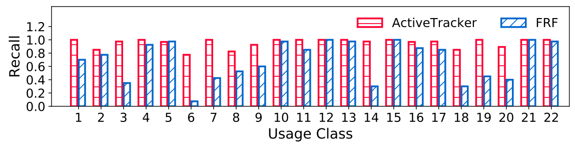
According to the experiment results shown in Fig. 7, FRF is expert in identifying video call and voice call, achieving the f-score of 98.3% and 97.3% respectively. However, FRF is not good at identifying other activities. For example, FRF only achieves a f-score of 57.1% on identifying transfer activity, and a f-score of 62.4% on recognizing text activity. In contrast, our model is able to recognize all activities with high precision and recall. In the task of recognizing video call and voice call, although the performance of FRF is already very high, our model's performance is still higher, achieving the f-score of 99.48% and 99.28% 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 method.

*4) Performance of App-activity Recognition:* We compare ActiveTracker and FRF 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.

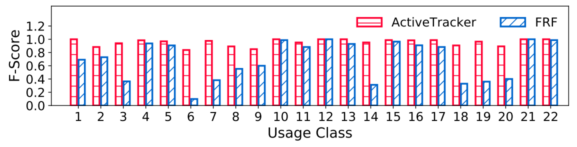
As shown in Fig. 8, our model again achieves high precision, recall and f-score for all app-activity classes. In contrast, the performance of FRF is much worse than that of our model. Specifically, it is hard for FRF to recognize text activity and



(a) Precision



(b) Recall



(c) F-score

Fig. 8. Performance of app-Activity Classification.

picture activity. 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 each app activity.

Particularly, we pay attention to the ability of recognizing the sensitive activities such as money transfer from the encrypted traffic. In Fig. 8, 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 100% and a recall of 100% when identifying money transfer in PayPal, and a precision of 93% and a recall of 100% when identifying money transfer in WeChat. In contrast, FRF has much lower precision and recall in recognizing money transfer, which is about 68.3% and 70.0% in PayPal, and 40.0% and 40.0% in WeChat. The ability of precisely recognizing sensitive activities could enable malicious attackers to spy people's money transfer by sniffing an access point in public space and cause a potential threat to people's property.

### C. *Comparison of Overall Accuracy*

Table V shows the overall accuracy of our model and FRF on the three tasks mentioned above. As can be seen, compared with FRF, our proposed classifier achieves much higher overall accuracy on all three tasks with great improvement. Specifically, FRF only achieves an overall accuracy of 74.32% on app recognition and an overall accuracy of 81.25% on in-app activity recognition. On app-activity recognition task, the overall accuracy achieved by FRF is even lower (69.55%), which indeed indicates that recognizing both app and activity is a more complicated and challenging task. In contrast, the proposed ActiveTracker achieves a high overall accuracy on both app recognition task (97.10%) and in-app activity recognition task (97.62%). For the most challenging app-activity recognition task, our model can still achieve an overall accuracy of 95.28%, 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.

TABLE V
OVERALL ACCURACY.

| Classifier | App | Activity | App+Activity |
|---|---|---|---|
| ActiveTracker | 97.10% | 96.62% | 95.28% |
| FRF | 74.32% | 81.25% | 69.55% |

### D. *Performance of App Trajectory Recognition*

Last we test the performance of uncovering continuous trajectory of app activities 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:

$$TDA = \frac{T(F \cap \hat{F})}{T(F)} \tag{3}$$

where $T(F)$ denotes the duration time of the whole traffic stream, and $T(F \cap \hat{F})$ captures the time durations (segments) in which the app activity is correctly identified. The results are shown in Fig. 9. As can be seen, a high time duration accuracy is achieved for most app activities. Specifically, video calls and voice calls can be easily uncovered with TDA approaching to 1. As for uncovering money transfer activity, our framework achieves a TDA of 86.32% on Paypal, and a TDA of 73.21% on Wechat, which indicates that our framework indeed has the ability to uncover the money transfer activity over a long encrypted traffic stream.

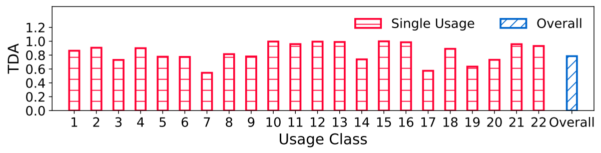


Fig. 9. Time Duration Accuracy.

## VI. Related Work

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

### A. *Internet Traffic Classification*

The conventional traffic classification focused on distinguishing the traffic generated by different Internet protocols. Bissias et al. proposed to gather profiles of specific websites from encrypted HTTP response streams, and use these profiles to infer the site being accessed of a trace of web traffic [23]. Draper-Gil et al. proposed a flow-based classification method to characterize encrypted and VPN traffic using only time-related features [24]. Wei et al. designed an end-to-end Internet traffic classifier based on one-dimensional convolution neural networks. Their proposed classifier identifies traffic flows by extracting patterns from the first 784 bytes of each flow [22]. Lotfollahi et al. [25] proposed Deep Packet, which embeds stacked autoencoder and convolutional neural network to identify traffic types and end-user apps.

### B. *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 [12], which automatically generates 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) [26], which learns the apps' distinguishing features via traffic analysis. It focuses on key-value pairs in HTTP headers and identifies the pairs suitable for app signatures. Miskovic et al. proposed AppPrint [13], a system that uses parts of the HTTP URLs or strings from HTTP headers, which are unique to the app as a fingerprint. [15] generates conjunctive rules from HTTP flow header to identify app. [27] transforms app identification into an information retrieval problem, and uses lexical similarity as a metric for classification task. However, the works above assume that mobile apps run over HTTP, and thus their methods are not applicable over encrytped Internet traffic. To solve the problem of identifying app in encrypted traffic streams, [14][28][29] adopts machine learning techniques, and [30][31] try to improve the classification accuracy through multi-classification.

### C. In-app Service Classification

In-app service classification aims to recognize the usage of different service 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 [32]. Fu et al. developed CUMMA [9], which classifies in-app service usages by jointly modeling user behavioral patterns, network traffic characteristics, and temporal dependencies. In their follow-up work [10], they improve the processing speed of this classifier by selecting most discriminative traffic patterns.

Different from existing works, our work addresses the problem of uncovering the trajectory of app activities. 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 before.

## VII. Conclusion

In this paper, we propose *ActiveTracker*, a framework to uncover the trajectory of app activities over encrypted Internet traffic streams. First, the incoming Internet traffic of mobile apps is segmented into several single-usage subsequences by a sliding window based approach. Then, each segmented traffic stream 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 usage trajectory recognition. Extensive experiments based on real-world encrypted mobile traffic show that the proposed approach achieves high accuracy in app usage trajectory recognition. Our work will rise people's attention on the privacy protection of mobile app communications.

## References

[1] "https://www.stonetemple.com/mobile-vs-desktop-usage-study/."

[2] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *Security and Privacy*, 2012, pp. 332–346.

[3] Tim, Frank, Mario, Schmitt, Jens, Martinovic, and Ivan, *Who do you sync you are?: smartphone fingerprinting via application behaviour*, 2013.

[4] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Can't you hear me knocking:identification of user actions on android apps via traffic analysis," vol. 194, no. 2-4, pp. 297–304, 2015.

[5] B. Raahemi, W. Zhong, and J. Liu, "Peer-to-peer traffic identification by mining ip layer data streams using concept-adapting very fast decision tree," in *IEEE International Conference on TOOLS with Artificial Intelligence*, 2008, pp. 525–532.

[6] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Local Computer Networks, 2005. Anniversary. the IEEE Conference on*, 2005, pp. 250–257.

[7] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of p2p traffic," in *ACM SIGCOMM Conference on Internet Measurement 2004, Taormina, Sicily, Italy, October*, 2004, pp. 121–134.

[8] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *International Conference on World Wide Web*, 2004, pp. 512–521.

[9] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Transactions on Mobile Computing*, vol. 15, no. 11, pp. 2851–2864, 2016.

[10] H. Xiong, H. Xiong, H. Xiong, H. Xiong, H. Xiong, and 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*, 2017, pp. 335–344.

[11] T. Bao, H. Cao, E. Chen, J. Tian, and H. Xiong, "An unsupervised approach to modeling personalized contexts of mobile users," in *IEEE International Conference on Data Mining*, 2010, pp. 38–47.

[12] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 809–817.

[13] S. Miskovic, G. M. Lee, Y. Liao, and M. Baldi, *AppPrint: Automatic Fingerprinting of Mobile Applications in Network Traffic*. Springer International Publishing, 2015.

[14] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *IEEE European Symposium on Security and Privacy*, 2016, pp. 439–454.

[15] 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," pp. 439–451, 2015.

[16] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," in *INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE*, 2017, pp. 1–9.

[17] S. X. Xue, L. Zhang, A. Li, X.-Y. Li, C. Ruan, and W. Huang, "Appdna: App behavior profiling via graph-based deep learning," 2018.

[18] Y. Fu, J. Liu, X. Li, and H. Xiong, "A multi-label multi-view learning framework for in-app service usage analysis," *Acm Transactions on Intelligent Systems and Technology*, vol. 9, no. 4, pp. 1–24, 2018.

[19] J. M. Joyce, *Kullback-Leibler Divergence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722.

[20] F. P. Miller, A. F. Vandome, and J. Mcbrewster, *Hermite Interpolation*. Springer New York, 2006.

[21] J. Unpingco, *Discrete-Time Fourier Transform*. Springer International Publishing, 2014.

[22] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *IEEE International Conference on Intelligence and Security Informatics*, 2017, pp. 43–48.

[23] G. D. Bissias, M. Liberatore, D. D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted HTTP streams," in *Privacy Enhancing Technologies, 5th International Workshop, PET 2005, Cavtat, Croatia, May 30-June 1, 2005, Revised Selected Papers*, 2005, pp. 1–11.

[24] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," in *The International Conference on Information Systems Security and Privacy*, 2016, p. 94–98.

[25] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," 2017.

[26] 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 *Computer Communications*, 2015, pp. 1481–1489.

[27] G. Ranjan, A. Tongaonkar, and R. Torres, "Approximate matching of persistent lexicon using search-engines for classifying mobile app traffic," in *IEEE INFOCOM 2016 - the IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

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

[29] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 114–125, 2016.

[30] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, 2017.

[31] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescape, "Traffic classification of mobile apps through multi-classification," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2018.

[32] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *ACM SIGCOMM Conference on Internet Measurement Conference*, 2011, pp. 329–344.