Xu and Cheng *Cybersecurity (2024)7:1*
https://doi.org/10.1186/s42400-023-00185-6

Cybersecurity

RESEARCH

Open Access

# F3l: an automated and secure function-level low-overhead labeled encrypted traffic dataset construction method for IM in Android

Keya Xu[1] and Guang Cheng[1,2,3*]

**Abstract**

Fine-grained function-level encrypted traffic classification is an essential approach to maintaining network security. Machine learning and deep learning have become mainstream methods to analyze traffic, and labeled dataset construction is the basis. Android occupies a huge share of the mobile operating system market. Instant Messaging (IM) applications are important tools for people communication. But such applications have complex functions which frequently switched, so it is difficult to obtain function-level labels. The existing function-level public datasets in Android are rare and noisy, leading to research stagnation. Most labeled samples are collected with WLAN devices, which cannot exclude the operating system background traffic. At the same time, other datasets need to obtain root permission or use scripts to simulate user behavior. These collecting methods either destroy the security of the mobile device or ignore the real operation features of users with coarse-grained. Previous work (Chen et al. in Appl Sci 12(22):11731, 2022) proposed a one-stop automated encrypted traffic labeled sample collection, construction, and correlation system, A3C, running at the application-level in Android. This paper analyzes the display characteristics of IM and proposes a function-level low-overhead labeled encrypted traffic datasets construction method for Android, F3L. The supplementary method to A3C monitors UI controls and layouts of the Android system in the foreground. It selects the feature fields of attributes of them for different in-app functions to build an in-app function label matching library for target applications and in-app functions. The deviation of timestamp between function invocation and label identification completion is calibrated to cut traffic samples and map them to corresponding labels. Experiments show that the method can match the correct label within 3 s after the user operation.

**Keywords** Encrypted traffic, Deep learning, Android, Labeled dataset

## Introduction

Accurate and efficient traffic classification, especially fine-grained, can assist Internet Service Providers (ISPs) in providing reasonable resource allocation and optimization for different Internet services. It is also the primary method for regulators to protect cyber security. However, encryption technology has been widely used in traffic transmission to deal with privacy risks, making the traditional Deep Packet Inspection (DPI) method of analyzing plaintext payload no longer applicable (Yang and Liu 2019). The Google Transparency Report "Percentage of pages loaded over HTTPS in Chrome by platform" shows that among all Chrome users, the proportion of web pages loaded using the HTTPS protocol reached 99% in December 2022, and this proportion is as high as 97% on the Android. Encrypted traffic has been widely used in various applications such as IM (Instant Messaging), game, and shopping and has become an unavoidable

*Correspondence:
Guang Cheng
gcheng@njnet.edu.cn
[1] School of Cyber Science and Engineering, Southeast University, Nanjing, China
[2] Jiangsu Province Engineering Research Center of Security for Ubiquitous Network, Nanjing, China
[3] Purple Mountain Laboratories, Nanjing, China

problem in network traffic analysis. As of June 2022, the number of mobile Internet users in China has reached 1.047 billion, and the proportion of Internet users using mobile phones to access the Internet has reached 99.6% (CNNIC 2022). Meanwhile, as of the fourth quarter of 2022, the market share of the Android system in mobile networking devices is as high as 71.1%, occupying a significant position in the mobile operating system (Statista 2022). In recent years, the number of mobile applications has shown explosive growth. The design of many popular mobile applications is evolving in integration and complexity, leading to the demand for fine-grained functional-level mobile encrypted traffic classification.

As machine learning and deep learning methods have become the mainstream of traffic classification (Velan et al. 2015), the labeled datasets, regarded as the basis of these two methods, are significant. The volume and coverage of the dataset will directly affect the generalization and robustness of models; the imbalance of samples in the dataset or excessive noise will affect the accuracy of the classification results, resulting in unacceptable traffic classification errors. In the current field of encrypted traffic classification, most of the labeled datasets used are a small number of public datasets, such as the ISCX VPN-nonVPN dataset (Draper-Gil et al. 2016) released in 2016, or private datasets collected by researchers themselves. However, the public labeled datasets have old samples that have yet to be updated in time, the number of samples is small, and the coverage needs to be improved. Their collection method with PC and WLAN devices throughout construction cannot eliminate background traffic. The accuracy of the trained models needs to be improved. It is difficult to apply them to the real Internet traffic classification scenario with rapid development and change, even using models trained by private datasets lacking updates. There are fewer public datasets on the mobile terminal, and the private datasets collected by the researchers have different sample collection methods and need a unified operating standard. Most private datasets are not disclosed due to the risk of leaking user privacy, making it difficult for other researchers to verify whether there is traffic mixing. Unbalanced samples may lead to problems such as a sharp decline in model versatility and robustness. At the same time, although the Android system is implemented based on Linux, it has a different security policy from Linux. Some operations that can be efficiently completed in the Linux system need to obtain root permission in the Android system. However, obtaining root permission means irreversible damage to the Android system, causing unknown risks and threatening user privacy and security.

To solve these problems, based on the A3C system (Chen et al. 2022), this paper proposes a function-level low-overhead labeled encrypted traffic datasets construction method for Android (F3L). With the IM and Android system analysis, this paper determines how to identify in-app functions in the foreground by monitoring UI (User Interface) controls and layouts. Field features of attributes of them are selected to build an in-app function label matching library to guarantee the accuracy and rapidity of identification. After that, since label matching takes time and causes partial traffic loss, the timestamp series generated from traffic samples are calibrated to make the function-level labels correspond entirely to the traffic.

The main contributions of this paper are as follows.

(1) This paper proposes that fine-grained encrypted traffic classification is an essential method for future network security and points out that the lack and impurity of current datasets bring difficulties for research. Hence, this paper proposes F3L to automatedly collect and label functional-level traffic samples in an Android system without root permission in a secure and efficient method way low consumption.
(2) Further, this paper considers the errors between label completion time and traffic generation time. Through enough experiments, this paper proves that the time threshold between user operation and F3L matching the in-app function label presents a fixed range of distribution in terms of application and in-app functions, which can be calibrated to eliminate.

## Related work

### Encrypted traffic classification

Encrypted traffic makes DPI methods for plaintext ineffective. In contrast, the encrypted traffic still has features that can be captured, so the researchers applied machine learning and deep learning methods to traffic classification. Currently, the research for encrypted traffic classification is divided into the following directions.

(1) *Feature* Extracting appropriate features from data and summarizing patterns is essential in machine learning and deep learning. Maonan et al. (2021) proposed a method based on ResNet and AutoEncoder to extract comprehensive information. Satrabhandhu and Tritilanunt (2021) focused on traditional machine learning and proposed a bidirection flow non-zero payload flow data extraction

scheme and bi-direction flow payload ratio feature. Shen et al. (2020) considered the cost of fine-grained classification, only using length features and traditional machine learning models to reduce overhead. Vasudevan et al. (2021) focused on application layer features, which a small number of could get better results. Chen et al. (2020) innovatively take the differences among encryption network protocol stacks into account, making feature extraction faster with multi-PDU lengths. Zhang et al. (2021) applied deep learning to feature engineering to explore deeper features in the traffic. Cai et al. (2021) mined the hidden topological information of the flow with Markov chains.

(2) *Model* With machine learning and deep learning development, more models are applied to encrypt traffic classification. Zhao et al. (2021) used HMM and LSTM. Aceto et al. (2021) explored multi-task traffic classification to optimize the performance of the deep learning model. Lin et al. (2022) designed the model structure combined with CNN and Bi-GRU. Wang et al. (2020) implemented a framework with CNN and SAE. Sun et al. (2020) proposed a method including the benefits of GCN and the autoencoder, so only a few labeled data were needed. Banihashemi and Aktharkavan (2022) designed an algorithm based on DNN. Zhou et al. (2021) proposed a 2D-CNN model and introduced image recognition into encrypted traffic classification. Yao et al. (2019) used RNN to model network traffic, while LSTM and HAM were introduced to assist classification.

(3) *Dataset* Nowadays, most encrypted traffic classification research adopts the ISCX VPN-onVPN dataset. However, the Internet is developing rapidly, and the dataset in the past has had serious data-drifting problems. The high classification accuracy depends on high-quality datasets. However, collecting traffic samples is often time-consuming, and it is not easy to ensure the balance and purity of the dataset. Therefore, some researchers tried to optimize data acquisition and solve the imbalance problem. Sharif and Moein (2021) proposed a cost-sensitive deep learning approach. Considering the same trouble, Wang et al. (2020) proposed the approach of data augmenting to generate samples to achieve data balancing.

Current research on encrypted traffic classification mainly focuses on machine learning and deep learning methods. With the development of models and algorithms, more and more new models are being applied in the field of encrypted traffic classification. However, as the basis of the supervised learning problem, the public dataset of encrypted traffic is still scarce. Most studies still use the ISCX VPN-nonVPN dataset released in 2016. However, the update speed of network protocols is breakneck, and old datasets cannot reflect the actual situation of the current internet. The research of improving and generating encrypted traffic labeled datasets can only improve the imbalance and other issues in the dataset. It cannot introduce new features of real internet encrypted traffic. Therefore, it is necessary to study accurate and efficient methods for constructing high-quality encrypted traffic labeled datasets.

#### Function-level labeled dataset construction

Although the mobile Internet has received widespread attention, gaps in related research still need to be filled. Currently, most mobile Internet encrypted traffic classification research focuses on the model optimization and feature engineering of machine learning and deep learning. However, the construction of datasets is the basis of machine learning and deep learning. Since the popularity of mobile networking devices is much later than Windows, the public datasets in this field are very scarce, and most of them are old and have yet to be updated in time. It is not easy to match the ever-changing mobile applications and their diverse in-app functions, so most researchers collect traffic by themselves and construct private datasets.

In the previous work, the A3C system implements application-level traffic collection in Android. Based on VPN Service, A3C changed the traffic path of the target application without root permission, and labeled the encrypted traffic with the application name. However, the sample label granularity of A3C is at the application-level and cannot separate the traffic of different in-app functions.

At present, researchers construct functional-level encrypted traffic labeled datasets on mobile terminals mainly in the following ways:

(1) Use tools such as Tcpdump to collect traffic in the root permission environment, monitor and record user operations and then replay them. Huabing et al. (2021) used dSploit, an Android system analysis tool, to collect traffic under root permission and selected traffic by adding packet filtering rules. Wei et al. (2012) built a multi-layer Android application analysis system. At the network layer, Tcpdump was suitable for Android collecting network traffic; system calls were monitored at the operating system layer; and user behavior such as clicking

and long-pressing the screen was monitored at the user layer with ADB to record, and network traffic was collected for analysis by replaying the recorded user behavior. Dai et al. (2013) proposed a heuristic UI path generation method to automatedly run the application in the simulator and automatedly explore the usage of the application, thereby simulating user operations and generating network interaction. The Android tool monkeyrunner was used to record and replay the coordinates and controls of the screen of the users' operations, and Tcpdump was used to capture the traffic. The labels of applications were identified by PID (Process Identification) of the applications by reading network calls. Among them, monitoring the user's operation on screen needs to obtain root permission.

(2) Cooperate with wireless AP (Access Point) and other devices, use tools such as Wireshark to collect traffic, and record labels manually. This method inevitably introduces irreducible background traffic and systematic errors into the dataset, so related research focuses more on reducing the noise of the datasets. Saltaformaggio et al. (2016) and Shafiq et al. (2016) reduced the noise of the sample datasets by only accessing specified devices and performing specified behaviors. Fu et al. (2016) recorded user behavior manually as labels and built a pure target application traffic environment by uninstalling non-target applications and setting Android firewalls. Deng et al. (2017) focused on the traffic research of WeChat and extracted the traffic belonging to WeChat through plaintext field matching in the cellular network traffic, but SNI (Server Name Indication) can tamper, and with the development and popularization of ESNI (Encrypted SNI) and ECH (Encrypted Client Hello) (IETF 2022), the methods of using plain text will be completely invalid. Yan et al. (2018) also took WeChat as the research target and reduced system errors by limiting the networking capabilities of other applications. Wu et al. (2021) manually recorded function switching when collecting function-level labeled traffic for labeling. Bartolec et al. (2020) researched on YouTube, using wireless AP to capture traffic and OCR (Optical Character Recognition) technology to identify video information to form labels. Loh et al. (2022) also collected YouTube traffic, combined with the wireless AP and ADB connection, and the information provided by the YouTube application to label the sample traffic. Afzal et al. (2021) took Signal Messenger as their research object and filtered the traffic of the specified application by setting up firewall rules, and its different functions in the application were operated manually.

(3) Simulate user behavior with scripts. Conti et al. (2015) used ADB to run scripts to automate the application operation, recorded the timestamp of each operation, marked the traffic after the timestamp as generated by the operation, and mapped for the corresponding function label accordingly. Regarding target application traffic filtering, the IP is filtered based on WHOIS, and only the flows related to the application were considered. Capturing traffic was carried out at the wireless AP through Wireshark. Afterward, Conti et al. (2015) made further improvements to the scripts, adding randomness to filling text in the edit box, not using static text, and using randomly selected from a large number of sentences instead. However, this type of scripting method still needs the features of real user operations. Bahuguna et al. (2021) used Appium to write scripts, operated specified application functions at fixed timestamps, used tshark to capture traffic, cut and formed labeled sample datasets according to specified timestamps, and obtained adequate flows based on filtered with SNI and DNS query. This type of method replaces real users with scripts, loses the real operation features of users, and obtains labeled datasets that do not conform to the actual usage scenario of the applications.

To sum up, the current construction method of function-level labeled datasets for encrypted traffic on mobile devices mainly has the following problems: (1) The acquisition of root permission destroys the native system of mobile devices, which has security risks; (2) The collection methods with wireless AP or other devices are complex, inefficient, and will introduce system errors that cannot be eliminated; (3) The real operation features of users are lost in the way of script simulation, which does not conform to the actual usage scenario of the applications.

## Analysis of IM in Android

As one of the most popular mobile operating systems today, Android has a complete architecture and components to give users an excellent visual and interactive experience. *Activity* is responsible for displaying the user interface, and all of the UI controls and layouts in the application are deployed in the *Activity*. Since an application may have multiple *Activity*s, in

**Table 1** Common UI controls in Android

| Name | Function |
|---|---|
| TextView | Display a text message |
| EditText | Allow users to input and edit content |
| Button | Display a button |
| ImageView | Display an image |
| ProgressBar | Display a progress bar |
| ProgressDialog | Pop up a dialog box and block other UI controls |
| AlertDialog | Pop up a dialog box and block other UI controls |
| ... | ... |

non-split-screen mode, only one *Activity* in the foreground can interact with the user and give user operation feedback. Regardless of complex situations such as split-screen mode and multi-screen collaboration, in the general usage scenario of the Android system, the life cycle of an *Activity* is divided into Running state, Paused state, Stopped state, and Killed state. When the *Activity* is in the foreground, it is visible to the user and can interact with the user, which means the *Activity* is Running. The user's operation can also be carried out on this *Activity* to generate calls to in-app functions and provide feedback to the user. Therefore, the identification and mapping of the function-level labels in the application should also be completed when the application and its *Activity* where the in-app functions are deployed are in the foreground.

In order to implement a variety of in-app functions, the Android system provides many UI controls to support a rich interactive experience, as shown in Table 1. A complex user interface can be formed by combining and arranging many UI controls natively provided by the Android system and customized by application developers. In order to make the display of the user interface logical and beautiful, the Android system provides layout as the container for UI controls. It forms a multi-layer nested hierarchical relationship with UI controls and has certain regularity. With the help of hierarchical nesting of layouts and UI controls, the application's interface in the Android system realizes a rich and diverse interaction mechanism with users, which can logically display various information and give users feedback on operations.

With the rapid development of the mobile Internet, many popular applications, such as Alipay, integrate many functions in a single application, providing users with many services, including online payment, financial planning, health code, and so on. Although these in-app functions belong to the same application, their traffic features are changeful and often even use different protocols for network communication. In IM, it is also widespread that different in-app functions of the same application use different transport and application layer protocols. Taking WeChat of the Android system as an example, it uses TCP, TLS, HTTP, MMTLS, and many other public and private protocols when transmitting text, pictures, files, and diverse information.

Different from the in-app functions integration method that provides different services by switching the entire foreground interface, the function switching in the chat window of the IM is more frequent. In extreme cases, whenever a user sends a message, it is possible to call a different network protocol using a different in-app function. Generally speaking, the in-app functions provided in the chat window of popular IM are shown in Table 2.

Compared with other types of applications, most of the functions of IM are frequently switched in daily usage scenarios. Hou et al. (2018) put forward a clear conditional hypothesis in their research: two different functions are performed sequentially, not concurrently. When the user uses the IM, the messages are also sent serially instead of in parallel. Therefore, this paper also assumes that the traffic generated by different in-app functions does not overlap; it belongs to the time interval of the function call in the application. The collected network traffic belongs only to the function and its background traffic that cannot be eliminated and will not be mixed with traffic generated by other functions. This assumption provides the possibility for labeling and mapping function-level samples. Looking further at various IM, it can be found that the visual feedback after the function call of this type of application follows the same interactive logic. The newly sent or received message is located at the bottom of the display area that belongs to this type of function in the *Activity* in Running state. When a new message appears, and there is no remaining display space in this area, the interface is automatedly scrolled up, and the new message is kept at the bottom of the display area.

**Table 2** Popular in-app functions in IM

| In-app function | Abbreviation |
|---|---|
| Send and receive text message | Text |
| Send and receive picture | Picture |
| Send and receive video | Video |
| Send and receive voice message | Voice |
| Send and receive file | File |
| Audio and video call | – |
| ... | ... |

## Function-level label identification and matching

In previous work (Chen et al. 2022), the A3C system was proposed to complete the automatic and pure application-level traffic collection. Based on A3C, this paper studies the function-level label of traffic to make samples more fine-grained. Cooperating with the A3C system to collect encrypted traffic, F3L is targeted to identify and cut the application-level traffic into samples with function-level labels. Since past researchers need to obtain root permission to monitor users' operation on the screen, F3L is based on Accessibility Service to monitor for changes in UI controls and layouts in the foreground to identify user operation, bypassing the limitation of root permission. In addition, F3L only listens instead of simulating user operation, and its listening content is still real user behavior, thus avoiding the loss of real user features.

### System overview

In order to identify the in-app functions in the IM with frequent switching in the Android system with non-root permission and correctly label the traffic samples in time, this paper proposes a function-level low-overhead labeled encrypted traffic datasets construction method for Android, F3L, which monitors the UI controls and layouts in the foreground without interfering normal usage. The overall architecture of the system is shown in Fig. 1. The in-app function label matching library is the core module of F3L. Different IM have different UI controls and layouts. Even if the same manufacturer develops them, different teams are responsible for the development and implementation. There are general differences in the audio-visual feedback and attributes of UI controls and layouts, which allows for building a specific in-app function label matching library. The feature fields of UI controls and layouts attributes are selected to build an in-app function label matching library, which can map the attribute changes of the UI controls and layouts with the corresponding in-app functions and accurately complete the function-level label matching within the shortest time threshold. The core of this module adopts the *Accessibility Service* interface in the Android system to provide the accessibility service. This interface has been supported since Android 1.6 and has been greatly improved in Android 4.0. It can collect interaction information about *Activity* in the foreground (Developers 2022).

The detailed process of the method is shown in Algorithm 1. After detecting and confirming that the *Activity* currently in the foreground belongs to the target application, the UI controls and layouts are scanned to form a snapshot. Unlike monitoring user operations on the screen, this step can be realized in a non-root environment. According to the in-app function label matching library, the content fields of each attribute of the UI controls and layouts are regularly matched with preset rules. Then the coordinates of the feature UI controls and layouts are read and compared to get the UI control and layout at the bottom of the *Activity* in the foreground. Since the latest in-app function is clear, the current in-app function-level label is obtained. Finally, the current label is compared with the last recorded label. If they are different, the latest in-app function in the target application has changed. Therefore, the new label and its timestamp must be recorded in the function-level label log to prepare for subsequent traffic cutting and mapping.

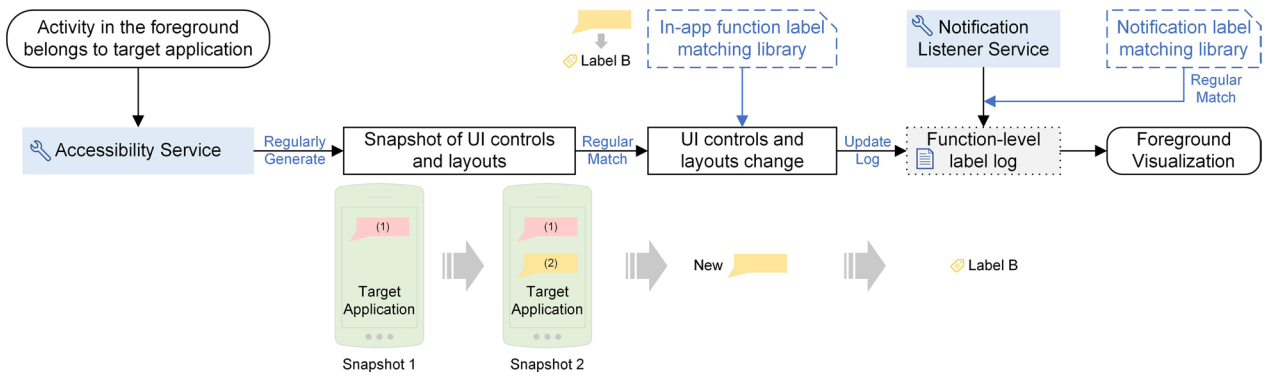



**Fig. 1** The architecture of F3L

**Algorithm 1:** In-app function label identification and matching

**Input:** Initialize ordinate of each functional UI control and layout $y_0, y_1, \cdots y_n \leftarrow 0$; Initialize the in-app function label $L_{old}$; In-app function label matching library $Lib_{func} \leftarrow [(feature_{func_0}, L_0), (feature_{func_1}, L_1), \cdots (feature_{func_n}, L_n)]$

**Output:** User behavior match in-app function-level label $L_{next}$

```
while True do
    // target application is running in the foreground
    if !feature_app.find().empty() then
        // scan to search for func_0 feature UI controls and layouts
        if !feature_func0.find().empty() then
            // the latest ordinate of UI control and layout of func_0
            y_0 ← feature_func0.findLast().bounds().bottom()
        end
        if !feature_func1.find().empty() then
            y_1 ← feature_func1.findLast().bounds().bottom()
        end
        ⋮
        if !feature_funcn.find().empty() then
            y_n ← feature_funcn.findLast().bounds().bottom()
        end
        y_x ← max(y_0, y_1, ··· y_n)
        L_new ← L_x        // latest label
        if L_new ≠ L_old then
            L_next ← L_new
        end
    end
end
return L_next
```

F3L applies to the function of the UI controls and layouts that can be obtained by scanning in the *Activity* in the foreground. Therefore, any in-app function that causes the changes in UI controls and layouts in the foreground can adopt the same principle for label identification and matching and only needs to add content fields of feature attributes to the in-app function label matching library to expand the preset rules.

However, as the audio and video call function attached to most IM, it reminds users of incoming calls in the form of floating windows or full-screen pop-up windows in many cases, and no UI control or layout can be directly scanned and obtained. It does not belong to the applicable scope of the in-app function label matching library. This paper proposes monitoring system notification as a supplement to F3L. The notification of the Android system is outside the UI and is used to display message reminders. In Android 4.3, Google provides the *Notification Listener Service* interface to allow listening to notifications, through which information such as the notification source's application and the notification content can be obtained. This paper also constructs an extensible notification label matching library, introducing the content of audio and video call notifications. A regular matching method is adopted to identify the audio and video call. The in-app function label is recorded in the function-level label log. This module does not need to confirm that the *Activity* in the foreground belongs to the target application. Even when the target application is in the background, F3L can also record this label.

#### Calibration of function-level label timestamp

For general users, sending and receiving messages in IM is a complex thinking activity. This paper assumes that the operation interval between different in-app functions should be greater than or equal to 1 s when users send and receive each message. Therefore, 500 *ms* is selected as the time interval for F3L to take every snapshot. Too high a frequency of snapshot generation will lead to unnecessary performance consumption. At the same time, too low a frequency will lead to the loss of some inappropriate sample labels. Since 500 ms is the median of 1 s, there must be a snapshot between two user operations, which allows for accuracy and performance.

This paper assumes that at timestamp $t_A$, the matched in-app function is $A$, and at subsequent timestamp $t_B$, the matched in-app function is $B$. Therefore, the traffic from

timestamp $t_A$ to $t_B$ is mapped to label $A$, and the traffic after $t_B$ is mapped to label $B$.

In an ideal state, F3L can immediately identify and update the function-level label after the user operates the in-app function. However, it takes time for the application to respond and give feedback. With a script to simulate user operations and A3C collecting traffic simultaneously, F3L identifies the in-app function labels. The three types of timestamps are recorded as shown in Fig. 2. Every green dot represents a packet. The darker the green is, the denser the packets are at this time point. So it is evident that after the user operates the functions in the target application, the application generates network interaction traffic first. After a while, the function-level label matching is completed. Hence, after the operation, the specified in-app function is called first, and the network interaction required by the in-app function starts, which leads to network traffic generation. Then in the foreground, UI controls, and layouts in the *Activity* are refreshed to give the user the necessary feedback on the operation. Currently, F3L can perceive the changes in the UI controls and layouts, compare them with the in-app function label matching library, and write the matching result into the function-level label log. Therefore, a time difference exists between user operation, network traffic generation, and label matching completion. To solve the problem that may lead to the dislocation of traffic cutting, this paper proposes to take advantage of the existence of Think Time (Microsoft 2012) to calibrate the timestamp of the function-level label. Otherwise, when an in-app function is triggered, the traffic may be omitted. Think Time is when users switch between different applications and functions and perform different operations in the load test. Since users need to spend a certain amount of time thinking before they operate in-app functions in actual usage scenarios, when the user switches to different in-app functions, there is a time interval that provides a time threshold for cutting labeled samples, which provides possibilities for accurate mapping of function-level labeled samples. Thanks to the high-speed Internet and manufacturers' demand for user experience, the time for feedback will be brief. Through enough experiments, the calibration threshold can be determined to match diverse in-app functions.

#### Feature attribute selection

This paper takes QQ, WeChat, and Telegram, three popular IM, as examples to analyze, considering text, picture, video, voice, and file, five in-app functions to introduce in-app function label matching library building steps. Both QQ and WeChat are Internet IM tools provided by Tencent, while Telegram is an IM that provides end-to-end encrypted communications.

Using the Android UI control analysis tool to scan the *Activity* of the three IM, it can find that their UI controls and layouts present a hierarchical pattern. The central part of the user interface of WeChat and Telegram is *RecycleView*, a component like a list. In contrast, QQ's central part is a custom component named *AbsListView*, which plays the same role. In order to support diversified visual effects and interactive experiences, these UI controls and layouts have complex multi-dimensional attributes, which can be used as features to identify different in-app functions, as shown in Table 3. These attributes include interactivity, visual effects, drawing order, and hierarchical relationship of the UI controls and layouts.

Because attributes are multi-dimensional and complex, even changing, this paper proposes that the following principles should be followed when selecting feature attributes:

(1) *Specificity* Since different in-app functions are in the same application, their corresponding UI controls and layouts must be similar. If the selected feature attributes lack specificity, it will lead to misidentification, making it impossible to obtain the correct function-level label.
(2) *Stability* First, feature attributes must be stable in the same application version. For example, bounds and depth will change with the sliding page, while the className and other attributes of the same type of UI controls will remain unchanged. Secondly, feature attributes should also be robust in different

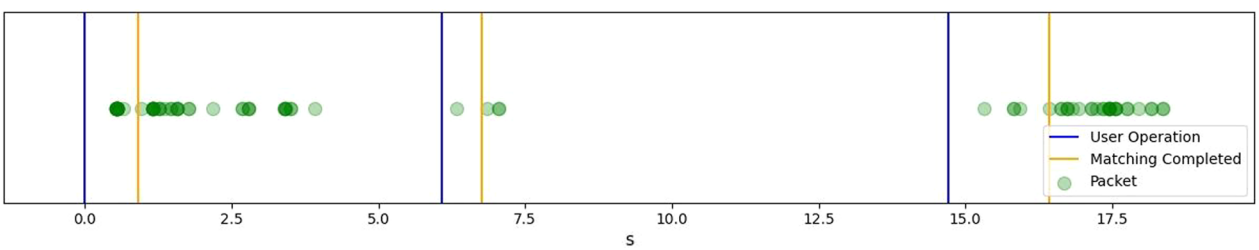



**Fig. 2** The process of application traffic transmission

**Table 3** Attributes of layouts and UI controls in Android

| Name | Function |
|---|---|
| className | The name of class |
| id | Identity document |
| desc | Description |
| text | Text content |
| bounds | Coordinates |
| depth | Layer |
| clickable | Whether to allow to click |
| longClickable | Whether to allow to long press |
| ... | ... |

versions of applications. For example, in multiple versions of QQ, the id of the dialog box is "qq_aio_ptt_time_tv". In contrast, in WeChat, the id of the dialog box is iterated with the version, which brings costs for updating the in-app function label matching library.

(3) *Availability* Compared with attributes such as id, which can directly regularly match content fields, the attributes of child UI controls involve the hierarchical nesting relationship between UI controls and layouts. A second search is required in the results of the first round of searches, which will burden the performance of F3L. Therefore, simple attributes that do not involve nested relationships should be selected as feature attributes preferentially unless a single attribute is not specific.

Finally, after updating iterations of multiple application versions, the relatively stable feature attributes of the five in-app functions in the three applications are selected, as shown in Table 4. Among the three IM, the five functions of QQ can all be identified accurately with a regular match through the content fields of the four attributes of className, id, desc, and text. The situation with WeChat is more complicated. According to the results of scanning and analysis, the id attributes of the layouts and UI controls in WeChat are volatile in the iterative update of the version. As the manufacturer continues to push new versions, the function label matching library in the application needs to be updated continuously, making the method's robustness not satisfied.

Moreover, due to the lack of a stable id in WeChat, when users operate the interface in the foreground, the UI controls and layouts that have nothing to do with the target function will also show a high degree of similarity

**Table 4** Feature attributes of in-app functions in matching library

| In-app functions | QQ | WeChat | Telegram |
|---|---|---|---|
| Text | id, className | longClickable, className, text | text, desc |
| Picture | id, className | desc, className | text, desc |
| Video | desc | clickable, className, childCount | text, desc |
| Voice | id, text | text | text, desc |
| File | text | className, desc | text, desc |

with those that serve as the feedback of the function. If selected feature fields are not specific, it may disturb the recognition results and reduce the accuracy. Therefore, compared with QQ, WeChat's function label matching library introduces attributes of child-controls and matching mechanisms related to application interaction to maintain the stability of feature attributes in the matching library and reduce the cost it needs to pay after the iteration of the application version. Compared with QQ and WeChat, which have complex structures of user interface and display rich visual effects, the UI controls and layouts in *Activity* of Telegram are more concise and have fewer nesting layers. The five functions in the chat window use the same UI controls and layouts packaging logic. The information received and sent by the user can also be directly obtained by reading the content field in the attributes of the UI controls and layouts to simplify the above algorithm. The feature UI control at the bottom of the chat window can be obtained directly to read its attributes, which can be matched with the feature content field in the matching library to determine whether to update the in-app function label.

## Experiments and results

In the actual usage scenario of IM daily, various in-app functions switch frequently. In order to ensure that the function-level labels are correct and the mapping with the traffic samples is entirely accurate and to provide support for the following encryption traffic classification method based on machine learning and deep learning, this paper verifies the accuracy and rapidity of F3L. This paper selects an Android smartphone equipped with a Qualcomm Snapdragon 730 G processor and 8GB of memory as the experimental device. Considering sending and receiving text messages, pictures, videos, voice messages, and files of three IM, which are WeChat, QQ, and Telegram, five popular in-app functions are included in the experiments.

**Accuracy of function-level labels identification**

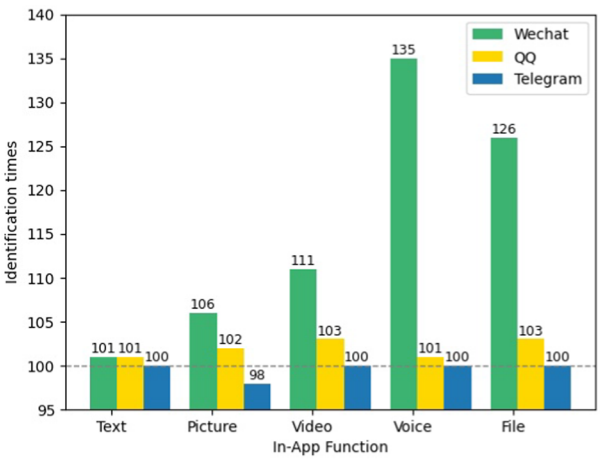
This paper adopts the method of manually sending five types of IM messages. It sends each type of message alternately 100 times in a fixed order of picture, video, text, file, and voice, records that each type of message is sent once as a round, and compares the result of label identification and matching with the real label to obtain its accuracy rate. The result is shown in Fig. 3a.

The experimental results show that after sending 100 rounds of messages, WeChat has many misidentifications. However, each message is accurately matched, and the locations of the misidentification phenomenon are shown in Fig. 3b. It is shown that the phenomenon mainly occurs when sending pictures, videos, and files. This is because when users send these three types of messages when viewing pictures, videos, and files, WeChat will play a sliding window animation to switch the *Activity*, and during the switching process, the UI controls and layouts related to sending and receiving messages in the chat window in the *Activity* are partially blocked, resulting in the incomplete layouts and UI controls scanned, so that the identification of the latest in-app function makes errors. The nature of recognition of UI controls and layouts is similar to that of OCR, and such problems are unavoidable errors due to the complexity of screen display content. Nevertheless, after the user sends a new message, the *Activity* on the interface for sending and receiving messages will return to the correct position and will not interfere with the following identification of the label of the in-app function.

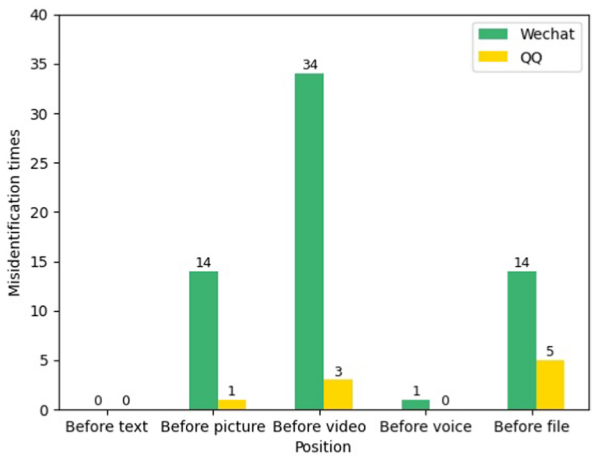
Due to the slow upload speed of videos, this paper simulates the operation of WeChat's four in-app functions by the script and carries out five rounds in the order of picture, text, file, and voice. After calibrating the label timestamp with the threshold in Sec. 5.2, the labels recorded in the function-level label log are shown in Table 5. After cutting the traffic collected by A3C, the number of packets corresponding to each function-level label is also shown below, where (*) means misidentification. In order to avoid errors caused by closing scripts, A3C and F3L at the end of the collection, the tail samples are discarded. It can prove that since the time threshold of wrong labels recorded by misidentification is mostly very short, and in WeChat when there is no operation of sending and receiving messages, there is often no network traffic interaction. So when traffic cutting and mapping are performed later, the corresponding traffic samples in the time threshold are empty and can be discarded directly, so such misidentification will not affect the construction of the encrypted traffic function-level labeled dataset.

Similar to the results of WeChat, a few misidentifications also occurred in QQ. The results are shown in Fig. 3 too. The reason is also that the UI animation blocks the UI controls and layouts, resulting in incomplete snapshots obtained by scanning, then identification goes wrong. However, the UI animation in QQ is smoother and faster than in WeChat, and the time threshold is smaller. The probability of misidentification is lower, and the impact on subsequent labeled sample cutting and mapping is also weaker.

Compared with WeChat and QQ, which have complex interfaces and rich animation effects, Telegram, due to the simplicity of its UI controls and layouts, makes the matching results of function-level labels in its applications extremely accurate, as shown in Fig. 3a, almost reaching 100%. According to the test, when the user's operation speed is too fast, which means after the




(a) Identification of in-app functions




(b) The distribution of misidentification

**Fig. 3** The matching results of function-level labels

**Table 5** WeChat function-level traffic sample

| Operation | Picture | Text | | | File | Voice | |
|---|---|---|---|---|---|---|---|
| Log1 | Picture | Text | | | File | Voice | |
| Packet number | 51 | 4 | | | 35 | 5 | |
| Log2 | Picture | Text | Voice* | | File | Voice | |
| Packet number | 49 | 4 | 0 | | 36 | 3 | |
| Log3 | Picture | Text | | | File | Voice | |
| Packet number | 52 | 4 | | | 36 | 5 | |
| Log4 | Picture | Text | Picture* | Text* | File | Voice | |
| Packet number | 46 | 4 | 0 | 0 | 36 | 5 | |
| Log5 | Picture | Text | | | File | Voice | Voice* |
| Packet number | 49 | 4 | | | 41 | – | – |

feedback with UI controls and layouts is generated, the user stays on the interface to be scanned and identified for less than the interval threshold between two scans of the screen, making the method unable to recognize the changes in the in-app functions in the two snapshots before and after the UI controls and layouts refreshed, resulting in failure to match function-level labels.

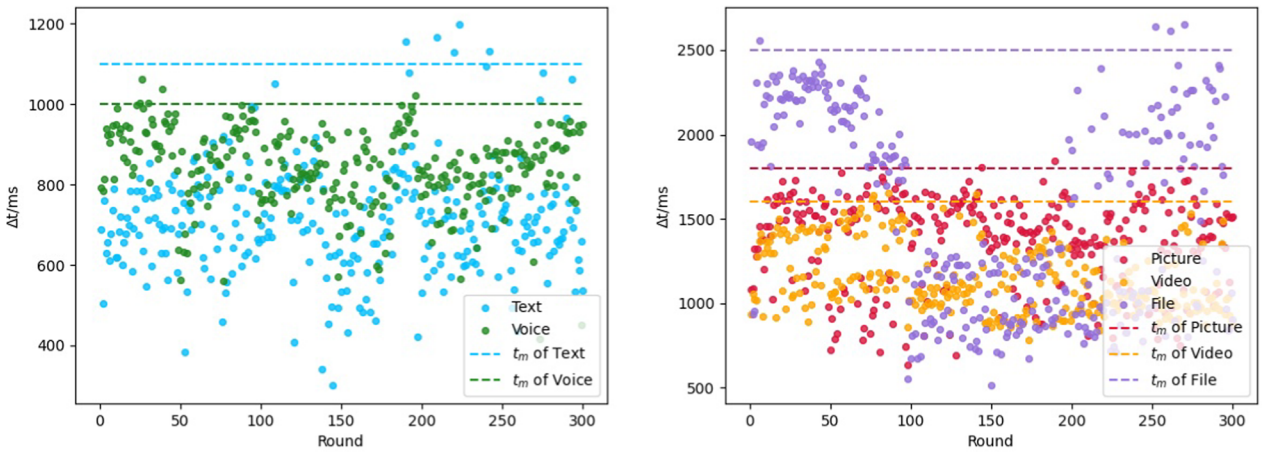
Therefore, according to the experimental results, the accuracy of label matching at the in-app function-level can lay the foundation for subsequent traffic cutting and mapping, thereby ensuring the accuracy of labeled samples. Since encrypted traffic classification is a typical supervised learning problem, and the dataset is its basis. In principle, the noiseless and accurate dataset collected by F3L is helpful for model training and prediction. As for experiments, it should be persuasive to compare with other function-level datasets, but the same type of public datasets in this area are rare.

#### Calibration threshold measurement and selection

In order to obtain the timestamps of user operations of applications and in-app functions, this paper uses automated scripts to simulate user operations. It records the timestamp $t_x$ of the five types of messages sent by the scripts. The timestamp $t_c$ of the function-level label matching completed and compares the difference between the two $\Delta t$ is used as the calibration threshold due to the applications and functions response and the time-consuming of label identification and matching. According to enough multiple experimental results, the function-level label timestamp calibration threshold $t_m$ is selected. In order to ensure the reliability and accuracy of the selected in-app functions calibration thresholds, pictures, videos, text messages, files, and voice messages are sent alternately in a fixed order, recording that each type of message is sent once as a round, and discarded 300 rounds of experiments were carried out. The results are shown in Fig. 4, 5 and 6. It can be seen that the $\Delta t$ of different in-app functions has a relatively stable distribution range, which provides feasibility for the selection of $t_m$. Because F3L is based on the UI controls and layouts in *Activity* in the foreground, the complexity of the UI controls and layouts will greatly affect the matching speed of function-level labels. Among them, since the information carried by the text and voice messages is relatively simple, $\Delta t$ is generally small. However, due to the greater information entropy, pictures, videos, and files also need to meet users' complex needs, such as clicking to view, clicking to play, and clicking to download, resulting in more complex UI controls and layouts for more feedback. More complex hierarchical relationships also require longer loading times. These reasons lead to the increase of $\Delta t$. Compared with WeChat and QQ, Telegram's UI controls and layouts have fewer layers and a more concise combination. Therefore, label identification and matching speed are higher than that of WeChat and QQ, and the distribution of $\Delta t$ is more concentrated.

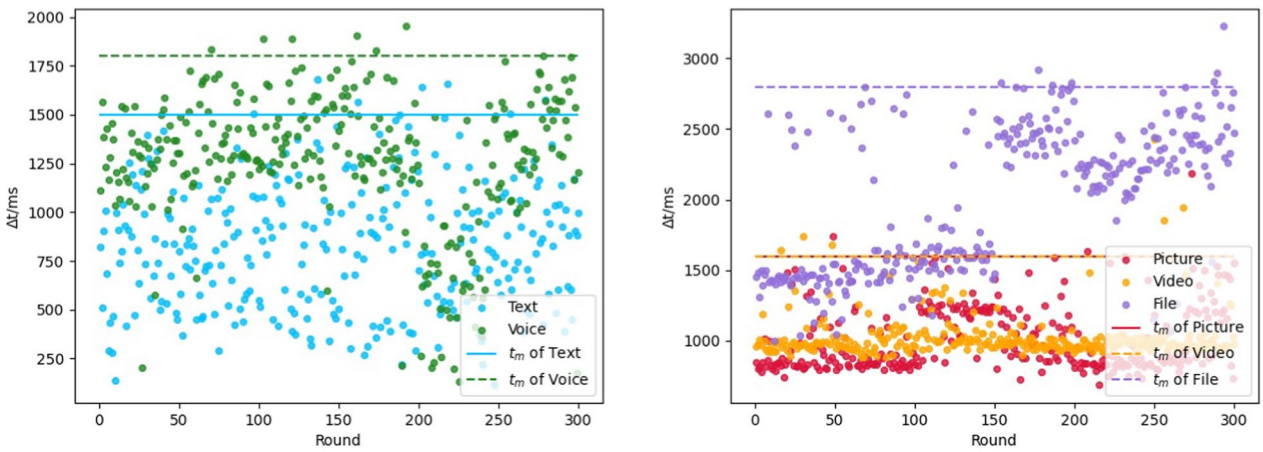
When selecting $t_m$, it is considered that when $t_m$ is too small, it may cause a large loss of head traffic during sample cutting and mapping, making the traffic corresponding to the in-app functions incomplete. When $t_m$ is too large, it may cause the tail traffic of the previous function to be incorrectly mapped to the current function-level label. After discarding the extreme values beyond the distribution range of most $\Delta t$, $t_m$ is chosen. In this paper, $C$ is defined as the ratio between the number of $\Delta t$ less than $t_m$ and the number of all $\Delta t$ measured through experiments. The final results of $t_m$ and $C$ of different applications and in-app functions are shown in Table 6. It can be seen that $C$ is average around 98%, proving that most $\Delta t$ can be calibrated.

The more concentrated the distribution of $\Delta t$, the larger the $C$ is. The simpler the UI controls and layouts are, the smaller the $t_m$ is. In addition, among the five




(a) text, voice (b) picture, video, file

**Fig. 4** $\Delta t$ distribution of WeChat




(a) text, voice (b) picture, video, file

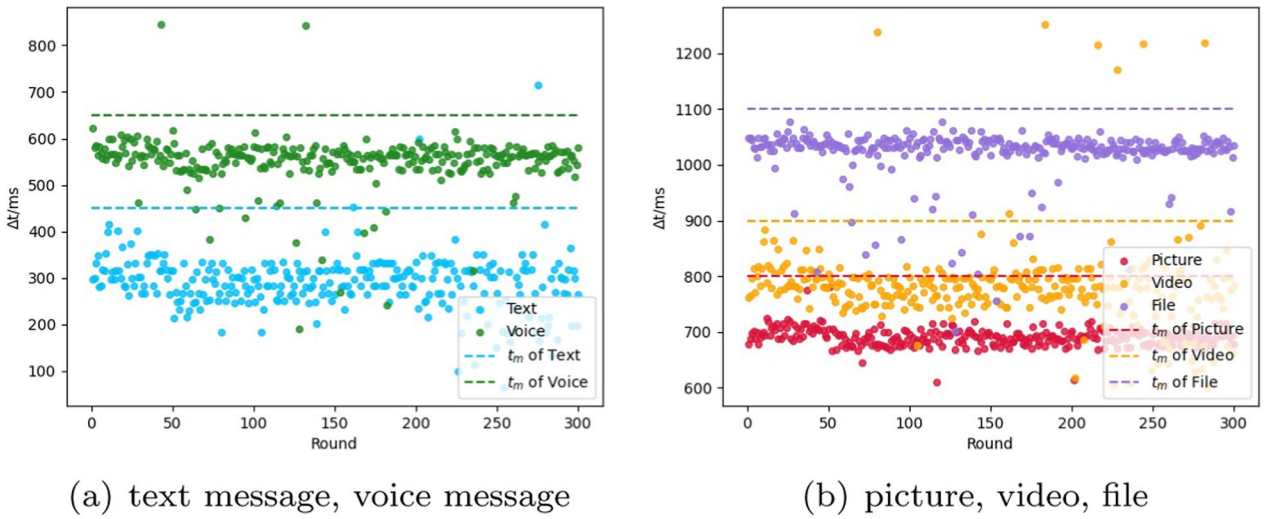
**Fig. 5** $\Delta t$ distribution of QQ

in-app functions of the three applications, the maximum $t_m$ is 2800 ms. Only the $t_m$ of WeChat and QQ file sending and receiving exceeds 2000 ms. The rest of the in-app functions can complete label matching within 2000 *ms*, proving that F3L is fast and efficient.

### Performance and cost

In daily usage scenarios, the performance of Android devices is often lower than that of PCs, and its storage space and computing power have a large gap compared with PCs, so the performance and cost of the method must be addressed. High system cost will cause the Android system to have the risk of freezing, affecting the normal use of applications running in the foreground. It will also reduce the efficiency of the Android system itself, even causing errors in the identification and matching results. For example, big games with high image rendering accuracy can only run on PCs. When on Android, it is often necessary to sacrifice image accuracy and loading time to adapt, leading to information loss.

This paper uses CPU usage and memory usage as indicators to evaluate the system performance. The specific experimental method enables F3L to run during the normal use of the foreground IM and records the CPU usage and memory usage of F3L every second. The test lasts for one minute. F3L is a tool that runs in the background of Android and monitors and identifies the UI controls and layouts of applications in the foreground. Since there is currently no function-level label identification tool of the same type, at the same




(a) text message, voice message (b) picture, video, file

**Fig. 6** $\Delta t$ distribution of Telegram

**Table 6** Function-level label timestamp calibration

| Application | WeChat | | QQ | | Telegram | |
|---|---|---|---|---|---|---|
| **Function** | $t_m$**/ms** | ***C*/%** | $t_m$**/ms** | ***C*/%** | $t_m$**/ms** | ***C*/%** |
| Text | 1100 | 98.3 | 1500 | 99.0 | 450 | 99.0 |
| Voice | 1000 | 98.3 | 1800 | 98.0 | 650 | 99.3 |
| Picture | 1800 | 99.3 | 1600 | 99.0 | 800 | 100 |
| Video | 1600 | 98.7 | 1600 | 98.0 | 900 | 97.7 |
| File | 2500 | 99.0 | 2800 | 97.7 | 1100 | 100 |

time, the Android device is sufficient to support the normal usage of IM in the foreground while playing music in the background. Hence, this paper proposes to compare three popular music applications supporting background running to verify that F3L can match function-level labels without affecting the foreground IM application. The results are shown in Fig. 7. It can be seen that the CPU usage and memory usage of F3L is similar to those of background music players. When F3L runs, the average CPU usage is 7.22%, and the average memory usage is 249.04MB.

Meanwhile, this paper also measured CPU and memory usage when WeChat, QQ, and Telegram were in the foreground. The results are shown in Fig. 8. Generally speaking, the total cost of IM and F3L is within the acceptable range of Android devices. Therefore, the resource occupation of F3L is small and will not affect the normal use of the target IM. Since the traffic generation environment conforms to the users' general usage scenarios, F3L ensures the authenticity and reliability of the labeled samples.

In addition, during the construction of the in-app function-level labeled dataset, A3C runs simultaneously with F3L. Their functional implementations are independent and do not affect each other. However, since A3C and F3L run simultaneously on an Android device, evaluating their performance and cost is also necessary. The experimental method enables F3L and A3C to run simultaneously during the normal use of the foreground IM and records the CPU usage and memory usage of F3L every second. The test lasts for one minute. The experimental results are shown in Fig. 9. It can be seen that excluding the performance peak caused by turning on A3C in the first second, the average CPU usage of A3C during normal operation is only 0.11%, and the average memory usage is 61.00 MB, which is much lower than that of F3L and other background music players. Therefore, combining A3C and F3L for automated encrypted traffic labeled dataset construction is undoubtedly low-overhead.

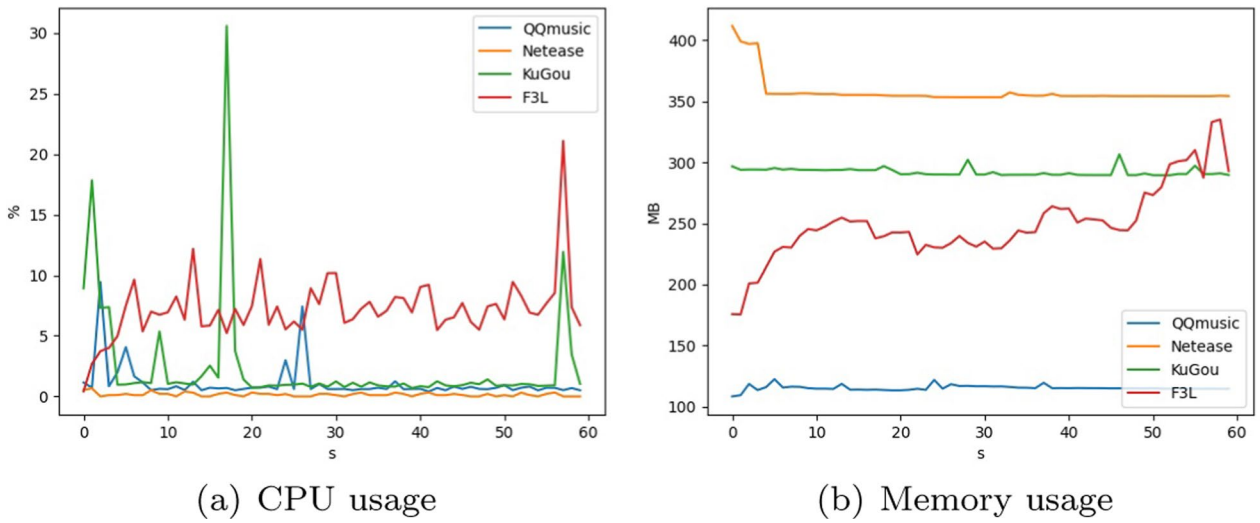



**Fig. 7** Performance and cost of applications in the background

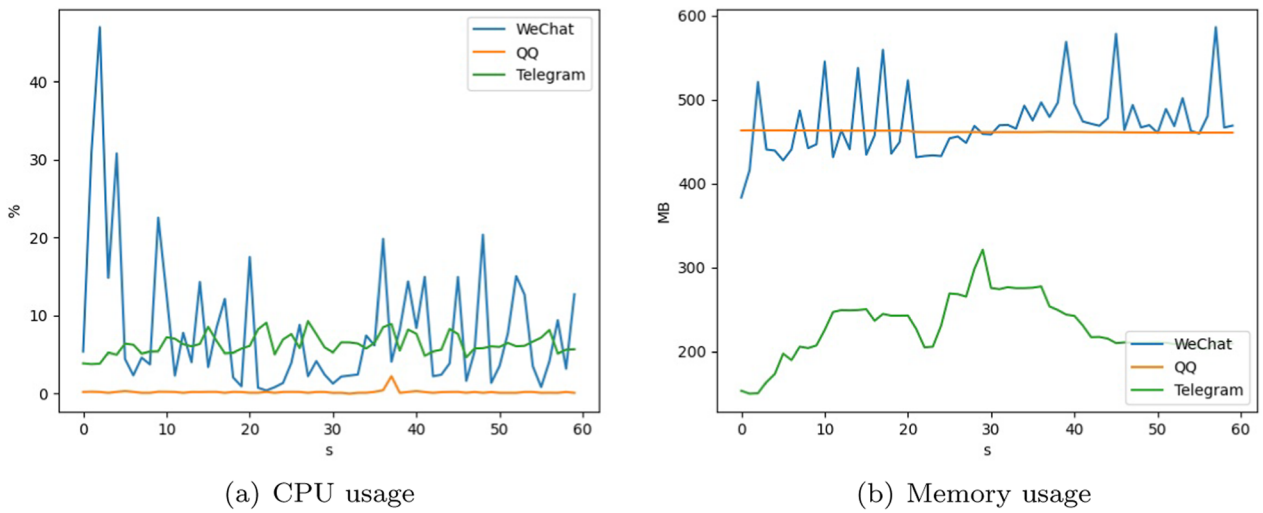


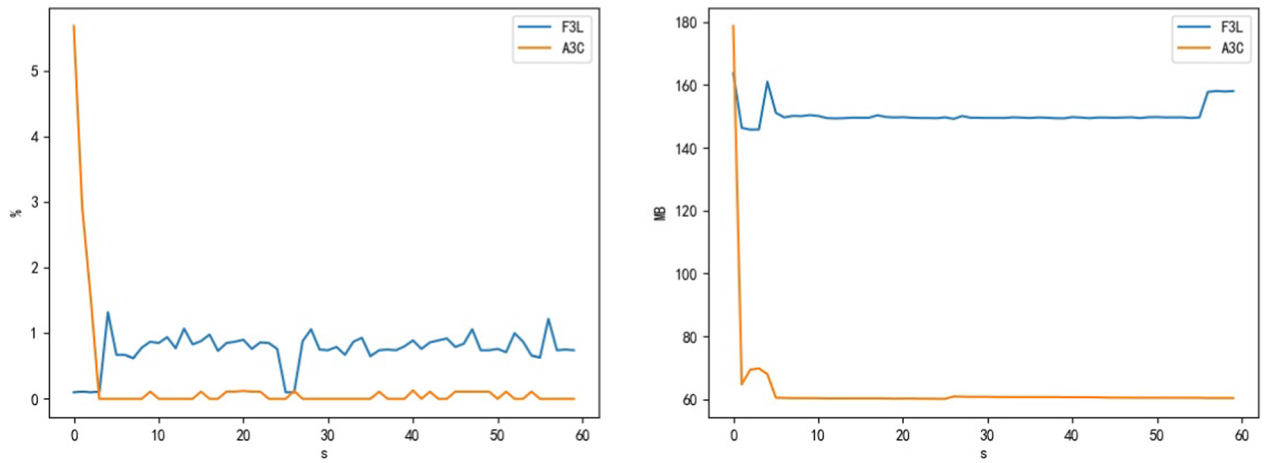

**Fig. 8** Performance and cost of IM in the foreground

## Limitation and discussion

The call of in-app functions is triggered by user behavior. Ignoring user behavior and focusing solely on the in-app functions will result in losing the real user behavior features in daily usage scenarios. F3L considers the feedback and display of in-app functions in UI controls and layouts, replacing simulation with monitoring. By constructing an in-app function label matching library, it identifies and records calls of in-app functions. This method suits in-app functions where UI controls and layouts may change.

However, F3L still has limitations. For example, in gaming and video applications, screen graphics have changed, but the attributes of UI controls and layouts have remained the same, resulting in F3L's inability to capture the in-app functions in these applications. For this type of problem, it may be possible to improve the universality of F3L by introducing a computer vision recognition model.

In addition, only IM experiments have shown that overly complex and sluggish UI animations can also affect the results of F3L. With the improvement of the integration of various IM applications and their increasing demand for performance, higher requirements will be placed on the speed and simplicity of F3L algorithms. Furthermore, there are numerous and complex instant messaging applications on the market, and the in-app functions covered in this paper are only a part of the popular functions. Further experiments on the performance of F3L are needed on other in-app functions.




(a) CPU usage (b) Memory usage

**Fig. 9** Performance and cost of A3C and F3L

## Conclusion

Machine learning and deep learning methods have become the main methods of encrypted traffic classification research, and constructing labeled datasets as their basis is still an urgent problem to be solved. In the current situation of increasingly complex and integrated mobile applications, this paper proposes a function-level low-overhead labeled encrypted traffic datasets construction method for IM in Android, F3L, using the method of monitoring the UI controls and layouts in the foreground, and building an in-app function label matching library, aims for different applications and in-app functions to label and map network traffic. F3L retains the real user behavior features and conforms to the general scenarios of the users' daily use of the applications.

This paper verifies the accuracy, performance, and cost of F3L, measures the calibration threshold of function-level label timestamp, and proves that it is a safe, accurate, efficient, and low-overhead method for constructing function-level encrypted traffic labeled datasets, which will provide data support for fine-grained encrypted traffic classification research. Since F3L has scalability and portability, it can be popularized for other mobile applications and in-app functions with UI controls and layouts changes as feedback in the foreground.

**Acknowledgements**
This work was supported by the General Program of the National Natural Science Foundation of China under Grant No. 62172093.

**Author contributions**
GC proposes the methodology and KX writes the manuscript. All authors read and approved the finnal manuscript.

**Availability of data and materials**
Not applicable.

## Declarations

**Competing interests**
The authors declare that they have no competing interests.

Received: 10 May 2023 Accepted: 22 August 2023
Published online: 01 January 2024

**References**

Aceto G, Ciuonzo D, Montieri A, Nascita A, Pescapé A (2021) Encrypted multi-task traffic classification via multimodal deep learning. In: ICC 2021-IEEE international conference on communications, IEEE, pp 1–6
Afzal A, Hussain M, Saleem S, Shahzad MK, Ho AT, Jung K-H (2021) Encrypted network traffic analysis of secure instant messaging application: a case study of signal messenger app. Appl Sci 11(17):7789
Bahuguna A, Agrawal A, Bhatia A, Tiwari K, Vishwakarma D (2021) User profiling using smartphone network traffic analysis. In: 2021 international conference on communication systems and NETworkS (COMSNETS), IEEE, pp 69–73
Banihashemi SB, Aktharkavan E (2022) Encrypted network traffic classification using deep learning method. In: 2022 8th international conference on web research (ICWR), IEEE, pp 1–8
Bartolec I, Orsolic I, Skorin-Kapov L (2020) Inclusion of end user playback-related interactions in youtube video data collection and ml-based performance model training. In: 2020 twelfth international conference on quality of multimedia experience (QoMEX), IEEE, pp 1–6
Cai W, Gou G, Jiang M, Liu C, Xiong G, Li Z (2021) Memg: mobile encrypted traffic classification with Markov chains and graph neural network. In: 2021 IEEE 23rd int conf on high performance computing and communications; 7th Int Conf on data science and systems; 19th Int Conf on Smart City; 7th Int Conf on dependability in sensor, cloud and big data systems and Application (HPCC/DSS/SmartCity/DependSys), IEEE, pp 478–486
Chen Z, Cheng G, Xu Z, Xu K, Shan Y, Zhang J (2022) A3c system: one-stop automated encrypted traffic labeled sample collection, construction and correlation in multi-systems. Appl Sci 12(22):11731
Chen Z, Cheng G, Jiang B, Tang S, Guo S, Zhou Y (2020) Length matters: fast internet encrypted traffic service classification based on multi-pdu lengths. In: 2020 16th international conference on mobility, sensing and networking (MSN), IEEE, pp 531–538
CNNIC: The 50th Statistical Report on China's Internet Development. China Internet Network Information Center (2022)

Conti M, Mancini LV, Spolaor R, Verde NV (2015) Analyzing Android encrypted network traffic to identify user actions. IEEE Trans Inf Forens Secur 11(1):114–125
Conti M, Mancini LV, Spolaor R, Verde NV (2015) Can't you hear me knocking: identification of user actions on Android apps via traffic analysis. In: Proceedings of the 5th ACM conference on data and application security and privacy, pp 297–304
Dai S, Tongaonkar A, Wang X, Nucci A, Song D (2013) Networkprofiler: towards automatic fingerprinting of Android apps. In: 2013 proceedings IEEE INFOCOM, IEEE, pp 809–817
Deng Q, Li Z, Wu Q, Xu C, Xie G (2017) An empirical study of the wechat mobile instant messaging service. In: 2017 IEEE conference on computer communications workshops (INFOCOM WKSHPS), IEEE, pp 390–395
Developers: create your own accessibility service, (2022). https://developer.android.com/guide/topics/ui/accessibility/service?hl=en. Accessed 12 Dec 2022
Draper-Gil G, Lashkari AH, Mamun MSI, Ghorbani AA (2016) Characterization of encrypted and vpn traffic using time-related. In: Proceedings of the 2nd international conference on information systems security and privacy (ICISSP), pp 407–414
Fu Y, Xiong H, Lu X, Yang J, Chen C (2016) Service usage classification with encrypted internet traffic in mobile messaging apps. IEEE Trans Mob Comput 15(11):2851–2864
Hou C, Shi J, Kang C, Cao Z, Gang X (2018) Classifying user activities in the encrypted wechat traffic. In: 2018 IEEE 37th international performance computing and communications conference (IPCCC), IEEE, pp 1–8
Huabing Z, Sisi Y, Xiaoming C, Zhida L (2021) Real-time detection method for mobile network traffic anomalies considering user behavior security monitoring. In: 2021 international conference on computer, blockchain and financial development (CBFD), IEEE, pp 11–16
IETF: TLS Encrypted Client Hello. https://www.ietf.org/archive/id/draft-ietf-tls-esni-15.txt. 2022-12-12 (2022)
Lin CY, Chen B, Lan W (2022) An efficient approach for encrypted traffic classification using cnn and bidirectional gru. In: 2022 2nd international conference on consumer electronics and computer engineering (ICCECE), IEEE, pp 368–373
Loh F, Wamser F, Poignée F, Geißler S, Hoßfeld T (2022) Youtube dataset on mobile streaming for internet traffic modeling and streaming analysis. Sci Data 9(1):1–12
Maonan W, Kangfeng Z, Ning X, Yanqing Y, Xiujuan W (2021) Centime: a direct comprehensive traffic features extraction for encrypted traffic classification. In: 2021 IEEE 6th international conference on computer and communication systems (ICCCS), IEEE, pp 490–498
Microsoft: About Think Time, (2012). https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms184790(v=vs.90)?redirectedfrom=MSDN. Accessed 12 Dec 2022
Saltaformaggio B, Choi H, Johnson K, Kwon Y, Zhang Q, Zhang X, Xu D, Qian J (2016) Eavesdropping on {Fine-Grained} user activities within smartphone apps over encrypted network traffic. In: 10th USENIX workshop on offensive technologies (WOOT 16)
Satrabhandhu W, Tritilanunt S (2021) Encrypted traffic characterization using none zero payload and payload ratio characteristics. In: 2021 25th international computer science and engineering conference (ICSEC), IEEE, pp 63–69
Shafiq M, Yu X, Laghari AA, Yao L, Karn NK, Abdesssamia F, et al (2016) Wechat text and picture messages service flow traffic classification using machine learning technique. In: 2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS), IEEE, pp 58–62
Sharif MS, Moein M (2021) An effective cost-sensitive convolutional neural network for network traffic classification. In: 2021 International conference on innovation and intelligence for informatics, computing, and technologies (3ICT), IEEE, pp 40–45
Shen M, Liu Y, Zhu L, Du X, Hu J (2020) Fine-grained webpage fingerprinting using only packet length information of encrypted traffic. IEEE Trans Inf Forens Secur 16:2046–2059
Statista: Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022, (2022). https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009. Accessed 12 Dec 2022
Sun B, Yang W, Yan M, Wu D, Zhu Y, Bai Z (2020) An encrypted traffic classification method combining graph convolutional network and autoencoder. In: 2020 IEEE 39th international performance computing and communications conference (IPCCC), IEEE, pp 1–8
Vasudevan S, Jain K, Su C-J (2021) Machine learning based encrypted traffic classification using estimated application layer statistics. In: 38th international communications satellite systems conference (ICSSC 2021), IET, vol. 2021, pp 189–194
Velan P, Čermák M, Čeleda P, Drašar M (2015) A survey of methods for encrypted traffic classification and analysis. Int J Netw Manag 25(5):355–374
Wang X, Chen S, Su J (2020) Automatic mobile app identification from encrypted traffic with hybrid neural networks. IEEE Access 8:182065–182077
Wang P, Li S, Ye F, Wang Z, Zhang M (2020) Packetcgan: exploratory study of class imbalance for encrypted traffic classification using cgan. In: ICC 2020-2020 IEEE international conference on communications (ICC), IEEE, pp 1–7
Wei X, Gomez L, Neamtiu I, Faloutsos M (2012) Profiledroid: multi-layer profiling of Android applications. In: Proceedings of the 18th annual international conference on mobile computing and networking, pp 137–148
Wu H, Wu Q, Cheng G, Guo S, Hu X, Yan S (2021) Sfim: identify user behavior based on stable features. Peer-to-Peer Network Appl 14(6):3674–3687
Yang B, Liu D (2019) Research on network traffic identification based on machine learning and deep packet inspection. In: 2019 IEEE 3rd information technology, networking, electronic and automation control conference (ITNEC), IEEE, pp 1887–1891
Yan F, Xu M, Qiao T, Wu T, Yang X, Zheng N, Choo K-KR (2018) Identifying wechat red packets and fund transfers via analyzing encrypted network traffic. In: 2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE), IEEE, pp 1426–1432
Yao H, Liu C, Zhang P, Wu S, Jiang C, Yu S (2019) Identification of encrypted traffic through attention mechanism based long short term memory. IEEE Trans Big Data 8(1):241–252
Zhang C, An C, Wang JH, Zhao Z, Yu T, Wang J (2021) Safsn: a self-attention based neural network for encrypted mobile traffic classification. In: 2021 IEEE international conferences on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData) and IEEE congress on cybermatics (Cybermatics), IEEE, pp 330–337
Zhao Y, Yang Y, Niu Y, Wu K, Hao Y, Su H, Zhao Q (2021) A classification and identification technology of tls encrypted traffic applications. In: 2021 IEEE 4th international conference on big data and artificial intelligence (BDAI), IEEE, pp 160–164
Zhou Y, Shi H, Zhao Y, Gao W, Zhang W (2021) Encrypted network traffic identification based on 2d-cnn model. In: 2021 22nd Asia-Pacific network operations and management symposium (APNOMS), IEEE, pp 238–241

## Publisher's Note