Hindawi
Security and Communication Networks
Volume 2020, Article ID 8891664, 15 pages
https://doi.org/10.1155/2020/8891664

Research Article

A Novel IM Sync Message-Based Cross-Device Tracking

Naixuan Guo, Junzhou Luo, Zhen Ling, Ming Yang, Wenjia Wu, and Xiaodan Gu

School of Computer Science and Engineering, Southeast University, Nanjing, China

Correspondence should be addressed to Naixuan Guo; guonaixuan@seu.edu.cn

Received 23 June 2020; Revised 17 August 2020; Accepted 31 August 2020; Published 22 September 2020

Academic Editor: Xiaolong Xu

Cybercrime is significantly growing as the development of internet technology. To mitigate this issue, the law enforcement adopts network surveillance technology to track a suspect and derive the online profile. However, the traditional network surveillance using the single-device tracking method can only acquire part of a suspect's online activities. With the emergence of different types of devices (e.g., personal computers, mobile phones, and smart wearable devices) in the mobile edge computing (MEC) environment, one suspect can employ multiple devices to launch a cybercrime. In this paper, we investigate a novel cross-device tracking approach which is able to correlate one suspect's different devices so as to help the law enforcement monitor a suspect's online activities more comprehensively. Our approach is based on the network traffic analysis of instant messaging (IM) applications, which are typical commercial service providers (CSPs) in the MEC environment. We notice a new habit of using IM applications, that is, one individual logs in the same account on multiple devices. This habit brings about devices' receiving sync messages, which can be utilized to correlate devices. We choose five popular apps (i.e., WhatsApp, Facebook Messenger, WeChat, QQ, and Skype) to prove our approach's effectiveness. The experimental results show that our approach can identify IM messages with high F_1-scores (e.g., QQ's PC message is 0.966, and QQ's phone message is 0.924) and achieve an average correlating accuracy of 89.58% of five apps in an 8-people experiment, with the fastest correlation speed achieved in 100 s.

1. Introduction

According to a report of CyberEdge Group, 80.7% of surveyed organizations were affected by a successful cyberattack in 2019 (https://cyber-edge.com/wp-content/uploads/2020/03/CyberEdge-2020-CDR-Report-v1.0.pdf). To defend against the cyberattacks, the law enforcement usually adopts network surveillance technology to track a suspect and analyzes the network traffic of his device to derive his online profile. However, the traditional network surveillance using the single-device tracking method can only obtain the online activities of a suspect's single device. As different devices play different roles in the MEC environment, one suspect can employ multiple devices to carry out a cybercrime. For example, a suspect may launch a cyberattack on a personal computer and communicate with his accomplices on a mobile phone. If only tracking his personal computer, the law enforcement cannot capture the suspect's accomplices. In this paper, we propose a novel cross-device tracking approach which is able to correlate one suspect's different devices (e.g., personal computer and mobile phone), which can help the law enforcement monitor a suspect's online activities more comprehensively.

Cross-device tracking approaches are mainly divided into two categories, i.e., deterministic tracking and probabilistic tracking [1]. The former approach relies on deterministic identifiers to correlate one user's different devices. For example, since one user logs in the same YouTube account on two devices, the YouTube website can achieve his login information directly to realize cross-device tracking. However, this approach can only be utilized by those companies which need users to log in. At present, most researchers focus on probabilistic tracking. This kind of approach is based on the similarity of user preference and behavior when one user operates different devices. For instance, Kane et al. [2] found that there is a certain overlap between the sites that users visit on their personal computer and mobile phone. Therefore, it is possible to perform cross-device tracking based on users' web logs on distinct devices. By providing users' web logs of different devices, ICDM2015

[3] and CIKM2016 [4] held two cross-device tracking competitions. These datasets provided were collected by commercial companies within about one month, and these two competitions mainly focused on designing linking algorithms based on the datasets. This approach has two drawbacks. First, this kind of cross-device approach often requires users' active participation, i.e., software installing and log records, which is difficult in practical application. Second, it is not suitable for fine-grained correlation with a specific range of users who have similar behaviors, while our approach in this paper is better to deal with this situation.

IM applications are typical CSPs in MEC, and they are widely used in daily life, e.g., chatting online with friends, transmitting files, and conducting video conferences. Google initially proposed an IM app called "Hangouts" that enables one user account to be logged in on multiple devices at the same time and sync messages automatically across devices. Therefore, if one user starts a chat on his computer, he can continue his chat on his phone (https://support.google.com/hangouts/answer/2944865?hl=en&ref_topic=6386410). In this way, instant messages are exchanged from one-to-one to one-to-multiple. For example, when you receive a message from a friend, your IM apps simultaneously used on two devices can receive this message almost at the same time. Then, many other IM apps such as WhatsApp, Facebook Messenger, WeChat, QQ, and Skype have added this functionality that is referred to as the cross-device message sync mechanism in this paper. The cross-device sync message can be leveraged to correlate two devices belonging to the same user so as to achieve the goal of cross-device tracking.

In this paper, we put forward a new cross-device tracking approach based on the IM sync message detection. We assume the law enforcement can capture target devices' network traffic from the gateway. Then, we classify the network traffic by IPs and identify the device type according to the user-agent field of HTTP packets. After that, we filter out the retransmitted packets which may reduce our correlation accuracy. We analyze a set of ground-truth IM application network traffic and find that sync messages are contained in the gate server's flow. Thus, we employ domain names matching to identify those flows of gate servers, which have specific domain names and summarize several filtering rules to identify the other flows. In order to identify sync messages in the gate server's flow, we extract the features of sync messages in advance and employ machine learning and rule matching as classification methods. According to the identification result, we extract sync messages' timestamps to form each device's message-receiving time list. Then, we calculate the time interval between each timestamp in two devices' time lists. It is regarded that when the interval is less than a threshold, it is one successful sync message match. According to the matching result, we employ the SPRT (sequential probability ratio testing [5]) algorithm to determine whether two devices are correlated or not. SPRT algorithm fits our scenario very well and gives theoretical support to our approach. It can make a decision as fast as possible when we observe the sequence of sync messages matching results.

To evaluate our cross-device tracking approach, we choose five popular IM applications (https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/) in our experiments: WhatsApp, Facebook Messenger, WeChat, QQ, and Skype. In our experiments, we first test the ability of our approach to identify messages. The best performance is PC's message of QQ, whose F_1-score reaches 0.966, while the worst is PC's message of Skype which also has 0.820. Then, we carry out device correlation experiments by designing a scenario that includes 8 people with 16 devices. The results show that our approach achieves a 97.9% matching accuracy of WeChat and an 83.3% accuracy of Skype. When the number of participants increases to 16, the matching accuracy of WeChat also exceeds 86%. Besides, we also analyze the factors, i.e., different users, message frequency, and user amount. We find that different users have little effect on correlating results, and the increase of message frequency or user amount reduces the correlation accuracy. Last but not the least, we count our matching time of each app, finding that most of the matching time length of QQ and WeChat is less than 400 seconds. This result indicates that our approach is faster than those approaches [6,7] which are based on long-time records of weblogs.

The major contributions of our paper are summarized as follows:

(i) First, we propose a novel approach based on network traffic analysis to realize cross-device tracking. Our approach just sniffs the network traffic silently which does not require users' active participation. Also, we do not analyze the deterministic identifiers such as IM numbers, usernames, and network IDs.

(ii) Second, we extract the features of five popular IM apps' messages to identify sync messages and employ rule matching and machine learning to identify different apps' messages. We deal with real users' network traffic which includes other apps' network traffic and background network traffic interference.

(iii) Third, we utilize the SPRT algorithm to accelerate the speed of judging whether the devices are correlated or not, and we perform experiments to prove that our approach is effective and fast, which can solve the problem of cross-device tracking in a fine-grained scenario.

The rest of this paper is organized as follows. We introduce the architecture of an IM system and the cross-device message sync mechanism in Section 2. In Section 3, we present our cross-device tracking approach, including the tracking scenario, basic idea, and the detailed workflow of our approach. Section 4 shows the experimental results of our approach, which is followed by discussing future research directions and corresponding countermeasures in Section 5. The existing related work is introduced in Section 6, and a conclusion is drawn in Section 7.

2. Background

In this section, we introduce the architecture of an IM system and the cross-device message sync mechanism. Then, we present the observation of IM application network traffic.

2.1. Architecture of an IM System. Figure 1 shows a typical architecture of an IM system. An IM system consists of IM clients and different types of servers, e.g., authentication servers, file servers, gate servers, and route servers [8]. The IM clients are installed on devices (e.g., mobile phones and personal computers) by users. The authentication servers are used to verify the user identity. The route servers act as a message relay center to concatenate the connection between users and relay their messages. The gate servers are edge servers that maintain a persistent chatting connection with the IM clients and mainly relay messages on behalf of the IM client. To communicate with each other, an original IM client sends a request to the route server, requiring the latter to establish a connection to the target IM client. Since the route server knows the gate servers of these two IM clients, it can concatenate connections of the gate servers of the two IM clients and relay their messages. In addition, the file servers are used to store and relay the files shared between users as most of the IM systems support file sharing functionality.

2.2. Cross-Device Message Sync Mechanism. The modern IM system supports the cross-device message sync mechanism so that a user can keep an IM conversation after he switches from one device to another in a hurry. We take the scenario in Figure 1 as an example to illustrate how an IM system works. As shown in this figure, user A logs in the same IM account on two distinct devices (i.e., a mobile phone C_1 and a personal computer C_2), and user B employs a device C_3 to communicate with user A. To keep an IM client online after a user logs in the device, a gate server is responsible for maintaining a persistent connection between the gate server and the IM client and relaying IM messages. As a result, C_1, C_2, and C_3 establish persistent connections to the gate servers S_1, S_2, and S_3, respectively. Once user B intends to communicate with user A, a route server S_4 is used to establish connections to the corresponding gate servers so as to concatenate the connections among C_1, C_2, and C_3 and then forward messages on behalf of user A and user B. To achieve the cross-device message sync functionality, the route server is used to create and send cross-device sync messages to ensure that C_1 and C_2 can receive the same message almost at the same time. In particular, if a message m_1 is sent from C_3 and arrives at the route server S_4, the message is copied and forwarded to gate servers (i.e., S_1 and S_2) by S_4. In this way, user A can receive the message, referred to as the sync message, on both C_1 and C_2.

2.3. Observation of IM Application Network Traffic. We collect the network traffic of IM applications to observe the pattern of sync messages. In this paper, we mainly focus on 5 popular IM applications around the world, including WhatsApp, Facebook Messenger, WeChat, QQ, and Skype. We install these IM clients on two types of devices (i.e., mobile phones and personal computers) and require users to send and receive chatting messages so that we can capture the IM chatting traffic. We find that the messages are transmitted in the gate servers' flow one by one as the user sends them sequentially. As mentioned in [9, 10], one message consists of a series of successive packets with short time interval and similar packet number in one application. In addition, since network traffic is encrypted between clients and gate servers, we cannot derive the plaintext of messages from the network traffic.

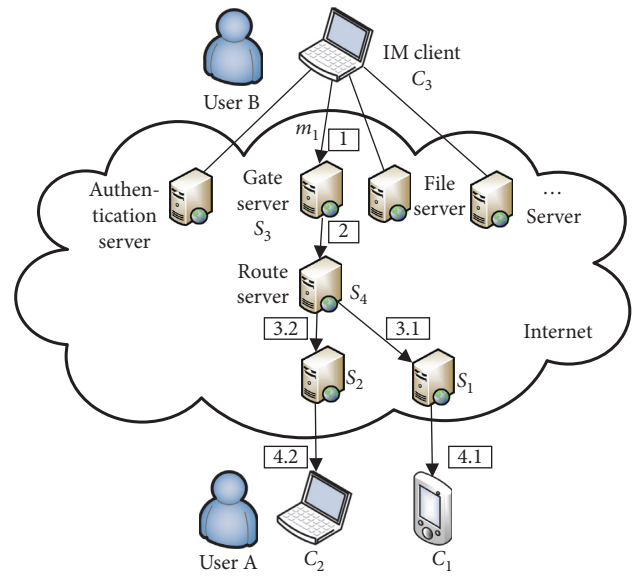

Figure 1: Architecture of an IM system.

3. Methodology

In this section, we first describe the assumption and basic idea of our cross-device tracking approach and then introduce the detailed approach, step by step. Table 1 summarizes the notations used throughout this paper.

3.1. Overview of the Cross-Device Tracking Approach. The law enforcement agency aims to correlate two devices used by a suspect so as to monitor his online behaviors. It is assumed that the suspect employs one personal computer (e.g., a desktop or a laptop) and one mobile phone to access the internet. It should be noted that we mainly focus on two kinds of devices in this paper, i.e., personal computers running Windows systems and mobile phones running Android systems. As Figure 2 shows, the suspect can operate his devices to access the internet via wired or wireless networks. The law enforcement controls the gateways of the network used by the suspect in order to passively record incoming and outgoing network traffic of the suspect. We also assume that the IP addresses of these devices do not change during a short period of time. In addition, we assume the suspect logs in the same IM application with one account

Table 1: Summary of notations.

Symbol	Meaning
$\uparrow, \downarrow$	An outgoing packet, an incoming packet
D, U	The set of devices, the set of users
X, Y	The set of personal computers, the set of mobile phones
X^k	The set of sync message timestamps of the kth PC
Y^ℓ	The set of sync message timestamps of the ℓth mobile phone
$Z^{k\ell}$	The set of sync message timestamps of the kth PC and the ℓth mobile phone successfully matched
$M_i^{k\ell}$	The result of matching sync messages in the ℓth mobile phone with the ith message of the kth PC
$\wedge_n$	The likelihood ratio
θ_0	The probability of sync message matching when two devices are not correlated
θ_1	The probability of sync message matching when two devices are correlated
η_u, η_l	The upper boundary, the lower boundary
α, β	The false-positive rate, the false-negative rate
T_m	The average time interval between user receiving sync messages
T_{interval}	The packet interval threshold
T_{offset}	The time offset of the network environment

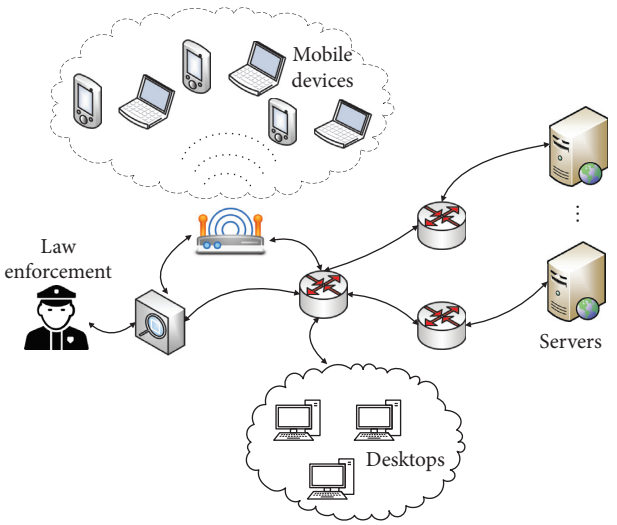

Figure 2: Cross-device tracking scenario.

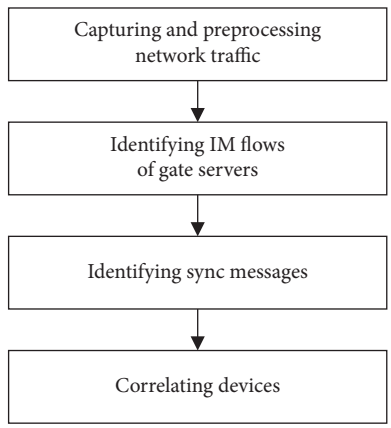

Figure 3: Workflow of the cross-device tracking approach.

on both the mobile device and the personal computer. As a result, when a user sends to the suspect a message, the IM applications on both distinct devices of the suspect will receive the message nearly at the same time due to the cross-device message sync mechanism. Since the sync messages are transmitted through the network, the law enforcement can passively identify two sync messages sent to these devices so as to link them. Last but not the least, we do not rely on the explicit identifiers, such as user IDs, in the IM messages to achieve cross-device tracking.

Figure 3 shows the workflow of our cross-device tracking approach. First, we capture and preprocess users' network traffic of mobile devices and personal computers. The network traffic is classified by IP addresses, and the device type (i.e., Windows personal computers and Android mobile phones) is identified according to the user-agent field of HTTP packets. Second, we collect ground-truth IM application (i.e., WhatsApp, Facebook Messenger, WeChat, QQ, and Skype) network traffic and analyze the features of gate server's flows. After that, we propose several filtering rules to identify the gate server's flows that contain sync messages. Third, we design experiments to analyze the attributes of sync messages and employ the rule matching and machine learning techniques to identify different applications' sync messages. After the identification of sync messages, we record their receiving timestamps to form the device sync message time list. At last, we compare the timestamps of sync messages between two devices and define the time gap that is less than 0.3 seconds as a successful match. Based on the matching results of sync messages of two devices, the sequential probability ratio testing (SPRT) algorithm is leveraged to make a quick decision of whether two devices are correlated.

3.2. Capturing and Preprocessing Network Traffic. As shown in Figure 2, we can capture the network traffic from the gateway and wireless routers, respectively. After capturing network traffic, we first classify them into several groups in terms of the IP addresses of the devices. Then, we identify the devices as either an Android mobile phone or a Windows personal computer based on the HTTP traffic transmitted from these devices. Since HTTP packets are common in network traffic from both mobile phones and personal

computers, we can use the value of the user-agent field in the HTTP packets as keywords to identify the device types. The user-agent field of HTTP packets from a Windows system often contains "Windows NT," while that from an Android system often includes "Android" (http://useragentstring.com/pages/useragentstring.php/). In addition, to reduce the noise, we remove the retransmitted packets from the captured traffic.

3.3. Identifying IM Flows of Gate Servers. We intend to identify the flows of IM gate servers that include the sync messages transmitted between gate servers and IM clients. To this end, we first collect a set of ground-truth IM network flows and analyze the communication mechanism of IM gate servers. We require a group of senders to use different IM clients to transmit messages to the other group of receivers who employ two devices to receive the sync messages. In particular, the senders use each IM client to transmit 20 messages with a time interval (e.g., 10 seconds). The receivers record the message timestamp upon obtaining a sync message. According to the timestamps, we can locate packets carrying the sync messages in network flows and treat these flows as the IM flows of gate servers. Then, we use the ground-truth flows to analyze the IP addresses and DNS request packets of the flows of gate servers. We find that the sync messages produced by the Windows version of QQ are transmitted by OICQ protocol, which is a kind of UDP packet that can be identified by using the DPI (deep packet inspection) technique. Meanwhile, the Android version of QQ and other IM applications employ TCP protocols to transmit sync messages between IM clients and gate servers. We find that gate servers of some IM applications own domain names, and others do not own. Therefore, there are two types of IM gate server flow recognition methods.

For the IM gate servers that have domain names, we collect a set of domain names of gate servers using the ground-truth flows as shown in Table 2. In the identification phase, we derive the DNS traffic of IM clients and use the domain name set to identify the flows of the gate server. It should be noted that the domain names in Table 2 are collected in our network within around 7 months. However, they may be changed due to the IM application update. In practice, the law enforcement can collect sufficient domain names in different network environments in advance.

For the IM gate servers that do not have domain names, we leverage reverse DNS lookup tools (e.g., Whois (https://www.whois.com/whois/)) to choose the candidate flow set and propose traffic filtering rules to identify the IM gate servers based on the IP addresses in the network flows. We discover that only gate servers of QQ and WeChat do not use domain servers, and both IM applications are developed by the Tencent company. Therefore, if the IP addresses of the servers belong to the Tencent company, we can save these flows. In addition, since gate servers of Tencent may use the network from different internet service providers (ISPs), we also save the traffic from the servers that belong to ISPs. Then, we summarize some features of the flows of gate servers to perform further traffic filtering:

(1) Since the persistent flows between IM clients and gate servers last longer than the others, we can discover the persistent flows in terms of the time period of the flows. Commonly, a message flow can last a few minutes or even longer, compared with ordinary flows that only last a few seconds. In practical, we use an empirical threshold (i.e., 1 minute) to identify potential IM persistent flows.

(2) According to the ground-truth traffic, we do not find any HTTP flows between IM clients and gate servers. Therefore, we can exclude the HTTP traffic that is the major traffic generated by user devices.

After employing these two filtering rules, there may be some candidate gate server's flows that we cannot uniquely identify. In this case, we can move to the next step to identify the sync messages for all candidate flows.

3.4. Identifying Sync Messages. After identifying IM flows of gate servers, we further analyze the ground-truth dataset of collected IM flows and extract the features of sync messages in order to identify them. According to our observation, each sync message transmitted from gate servers to the IM client corresponds to a series of successive packets with short packet time interval, while the interval between two messages is longer. Then, we can use an empirical packet interval threshold (i.e., T_{interval}, we conduct statistical analysis to derive the value of T_{interval} in Section 4) to automatically segment the IM flows to derive groups of packet sequences; these groups contain sync message packet group and background network traffic group. As mentioned before, in order to locate the sync message packet group, we record the timestamps when receiving the sync messages. We find that the packet pattern of sync messages from QQ, WeChat, WhatsApp, and Facebook Messenger can be extracted for sync message identification. However, the packet pattern of a sync message of Skype is different from the other four IM applications. Therefore, we try to employ a machine learning method to identify this kind of message (i.e., Skype).

We can use the packet direction pattern and packet length to identify the sync messages from the four IM gate servers (i.e., QQ, WeChat, WhatsApp, and Facebook Messenger). After the flow segmentation, the segmented packets from the PC client of QQ as well as PC and mobile clients of WeChat can be directly used to extract the packet direction pattern as there is no noise packet. However, we need to preprocess the segmented packets from the rest of the clients to remove the noise packets and derive the packet direction pattern. For the QQ mobile clients, we remove the packets, whose length is less than 100 bytes. For the traffic from clients of WhatsApp and Facebook Messenger, we apply the longest common subsequence (LCS) algorithm to sequences of packets so as to extract the common subsequence of the packet direction pattern for their sync messages. Then, the longest common subsequence can be used to detect the sync messages.

After deriving the packet direction patterns, we perform statistical analysis of the packet length and derive a specific

Table 2: Domain names of gate servers.

Application	Device type	Domain names
QQ	PC	*
	Mobile phone	*
WeChat	PC	long.weixin.qq.com
	Mobile phone	*
Skype	PC	(i) ip.wusmw1-client-s.msnmessenger.msn.com.akadns.net
	Mobile phone	(ii) ip.ea2hk2-client-s.msnmessenger.msn.com.akadns.net
WhatsApp	PC	mmx-ds.cdn.whatsapp.net
	Mobile phone	(i) chat.cdn.whatsapp.net (ii) Whatsapp-chatd-edge-shv-02-hkg3.facebook.com
Facebook Messenger	PC (web)	star.c10r.facebook.com
	Mobile phone	(i) mqtt.c10r.facebook.com (ii) edge-mqtt-mini-shv-02-hkg3.facebook.com

packet length or a length range for each packet in the packet direction pattern to further reduce the false-positive rate of sync message identification. It is important to note that we discover that the packet direction patterns extracted from the PC client of WeChat as well as the mobile clients of WeChat and WhatsApp are useful enough to identify the sync message. Therefore, we do not use the packet length to identify the sync messages from these clients. For the WhatsApp PC clients, the range of the first packet length is used to decrease the false detection. Moreover, since the OICQ protocol used by the QQ PC clients is not encrypted, the deep packet inspection technique can be applied to further improve sync message identification. In particular, the fourth and fifth bytes of the two packets in the sequence are 0x17 and 0xce in hex.

Table 3 shows the features used to identify the sync messages. We denote "↑" and "↓" as an outgoing packet and an incoming packet, respectively. Then, packets of a sync message can be represented as a sequence of "↑" and "↓." The packet length range is denoted as (x, y). Then, we use "–" to concatenate the length range of each packet in order. For example, $(200, \text{MTU}) - 97$ indicates that the packet length of the first packet in the sequence is between 200 and MTU (maximum transmission unit), and the second packet length is 97. In addition, we denote "FPL" as the length of the first packet in the packet sequence.

As the packet pattern of the sync message of Skype is different from the other four applications, we employ machine learning methods to identify it. After segmenting the flows of the gate server into different bunches of the packet sequence, we process each bunch of the packet sequence to obtain statistical features which are shown in Table 4. There are four categories which contain 25 features in this table. We count the number of continuous packet subsequences whose directions are "↑↑," "↓↓," and so on. We also derive the statistical data (i.e., mean, standard deviation (STD), and maximum) of the packet length and packet time interval. In addition, we perform statistical analysis of the mean of the length and time interval of the first 1/3 packets, the second 1/3 packets, and the rest 1/3 packets, respectively. Finally, we choose to employ these features and the machine learning methods (i.e., XGBoost [11] and random forests [12]) to identify the packet sequence of the Skype sync message.

Once the sync message is identified, we choose the timestamp of the first packet in the preprocessed packet sequence as the sync message timestamp. After identifying all sync messages, we can derive a sequence of the timestamps of the sync messages for each device.

3.5. Correlating Devices. We formalize the cross-device tracking problem in this section and elaborate on the theory of correlating devices. We denote two different types of devices (i.e., the mobile device and the personal computer) as D_1 and D_2 and denote the hypothesis "two devices are correlated" as H_1 and "two devices are not correlated" as H_0. Then, we can formalize H_1 and H_0 as

$$\begin{aligned} H_1&: D_1 \in U_a \text{ and } D_2 \in U_a, \\ H_0&: D_1 \in U_a \text{ and } D_2 \in U_b, \end{aligned} \tag{1}$$

where U_a and U_b represent different users. If two devices belong to the same user, we accept H_1; otherwise, we accept H_0.

We correlate two devices (i.e., a personal computer and a mobile phone) by matching their sequences of the timestamps of the sync messages. We assume that there are several users and they each have a personal computer and a mobile phone, respectively. We denote sequences of the sync message timestamps from the kth personal computer and the ℓth mobile phone as $X^k = \{t_1^k, \ldots, t_i^k, \ldots\}$ and $Y^\ell = \{t_1^\ell, \ldots, t_j^\ell, \ldots\}$, respectively. To match the sync message transmitted to two different types of devices, we compare the time difference between each identified sync message from a personal computer and all identified sync messages from all of the mobile phones. The ith sync message of the kth personal computer and the jth sync message of the ℓth mobile phone are matched if $|t_i^k - t_j^\ell| \le \Delta t$, where Δt is an empirical value (i.e., 0.3 second) derived by our statistical analysis in Section 4. If we

Table 3: Features of four applications' sync messages.

Application	Device type	Packet direction	Packet length
QQ	PC	↓↑	(200, MTU)-97
	Mobile phone	↓↑↓	(200, 1000)-(200, 1000)-(100, MTU)
WeChat	PC	↓↑↓↑ or ↓↑↓↓↑	
	Mobile phone	↓↑↑↓	
WhatsApp	PC	↓↑ or ↓↑↓↓↑↓↑	FPL > 200
	Mobile phone	↓↑↑↓ or ↓↑↑↑	
Facebook Messenger	PC (web)	↓↑↑↓ or ↓↑↓	(100,MTU)-(0,100)-(0,100)-(0,100) or (100,MTU)-(87 or 94)-54
	Mobile phone	↓↑↑↓ or ↓↑↓	(200,MTU)-66-(54,MTU)-66 or (200,MTU)-(101 or 99)-66

Table 4: Features of Skype's sync messages.

Category	Features
Packet number	Total packet number, ratio of outgoing to incoming packet number, number of ↑, ↓, ↑↑, ↓↓, ↑↑↑, ↓↓↓, ↓↑↓, ↑↑↑↑, ↓↓↓↓, ↓↑↑↓
Packet direction	Direction of the first packet
Packet length	Mean, STD, max, mean (low 1/3), mean (mid 1/3), mean (high 1/3)
Packet time interval	Mean, STD, max, mean (low 1/3), mean (mid 1/3), mean (high 1/3)

have enough samples that match the formula above, we can correlate the two devices.

Next, we use the sequential probability ratio testing (SPRT) algorithm to determine how many sync messages are matched so that we can accept H_1 and reject H_0. We use a binary decision variable $M_i^{k\ell}$ to represent whether the ith message of the kth PC is matched with a sync message of the ℓth mobile phone. If matched, $M_i^{k\ell}$ is equal to 1. Otherwise, $M_i^{k\ell}$ is set as 0. After each sync message from the kth PC and the ℓth mobile phone is compared, we can derive a message-matching sequence, i.e., $\{M_1^{k\ell}, \ldots, M_n^{k\ell}\}$. Then, we define $P(M_i^{k\ell} = 1 \mid H_1) = \theta_1$ and $P(M_i^{k\ell} = 1 \mid H_0) = \theta_0$, i.e., the probability of the ith message of the kth PC matched with a sync message of the ℓth mobile phone when the hypothesis of two devices correlated is true and false, respectively. Intuitively, the probability of two messages matched (i.e., θ_1) is high if the two devices are correlated. On the contrary, if the two devices are not correlated, the probability (i.e., θ_0) is low. Then, we assume $M_i^{k\ell}$ are independent and identically distributed (i.i.d.). We can calculate the likelihood ratio $\wedge_n$:

$$\begin{aligned} \wedge_n &= \ln \frac{\Pr\left[M_1^{k\ell}, \ldots, M_n^{k\ell} \,\middle|\, H_1\right]}{\Pr\left[M_1^{k\ell}, \ldots, M_n^{k\ell} \,\middle|\, H_0\right]} \\ &= \ln \frac{\prod_{i=1}^{n} \Pr\left[M_i^{k\ell} \,\middle|\, H_1\right]}{\prod_{i=1}^{n} \Pr\left[M_i^{k\ell} \,\middle|\, H_0\right]} = \sum_{i=1}^{n} \ln \frac{\Pr\left[M_i^{k\ell} \,\middle|\, H_1\right]}{\Pr\left[M_i^{k\ell} \,\middle|\, H_0\right]}. \end{aligned} \tag{2}$$

According to the SPRT algorithm, we calculate the likelihood $\wedge_i$ sequentially according to the values of $M_i^{k\ell}$ one by one until it reaches stopping boundaries [13, 14]. Then, we have

$$\wedge_i = \begin{cases} \wedge_{i-1} + \ln \dfrac{\theta_1}{\theta_0}, & M_i^{k\ell} = 1, \\ \wedge_{i-1} + \ln \dfrac{1-\theta_1}{1-\theta_0}, & M_i^{k\ell} = 0, \end{cases} \tag{3}$$

where $1 \le i \le n$ and $\wedge_0 = 0$. It means that when $M_i^{k\ell} = 1$, we add $\ln(\theta_1/\theta_0)$; otherwise, we add $\ln((1-\theta_1)/(1-\theta_0))$. The stopping rule is

$$\begin{cases} \wedge_i \ge \eta_u, & \text{stop and accept } H_1, \\ \wedge_i \le \eta_l, & \text{stop and accept } H_0, \\ \eta_l < \wedge_i < \eta_u, & \text{calculate } \wedge_{i+1}, \end{cases} \tag{4}$$

where η_u and η_l represent an upper boundary and a lower boundary, respectively. This rule means that we need to calculate $\wedge_i$ until it reaches η_u or η_l. According to the theory of SPRT, η_u and η_l can be defined by

$$\begin{aligned} \eta_u &= \ln \frac{1-\beta}{\alpha}, \\ \eta_l &= \ln \frac{\beta}{1-\alpha}, \end{aligned} \tag{5}$$

where α and β are the user-chosen false-positive rate and false-negative rate, respectively. We use $\alpha \le 0.01$ and $\beta = 0.01$ [14].

We need to calculate θ_0 and θ_1, respectively, to derive the likelihood $\wedge_n$. To compute θ_1, we collect the ground-truth dataset and conduct statistical analysis to derive the value of θ_1 in Section 4. We collect the ground-truth data of sync messages from several pairs of correlated devices and calculate the ratio of successful matching sync messages to all sync messages. Then, we set $\theta_1 = 0.9$ in our paper.

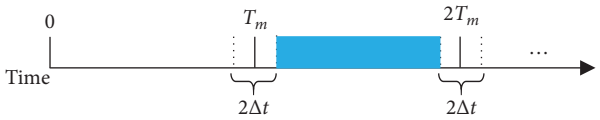

Figure 4: Sketch map of message matching.

We build a model to analyze the factors which impact θ_0 and then compute it. We can take time as a coordinate axis and the sync messages' timestamps of the devices as the coordinates. According to our statistical analysis, the average time interval between the user receiving sync messages equals T_m (i.e., $T_m \geq 10$), and the time gap between two matched sync messages is less than Δt. As shown in Figure 4, if one sync message timestamp of the mobile phone is located near the sync message timestamps of the PC (e.g., $[T_m - \Delta t, T_m + \Delta t]$), these two sync messages are matched. Otherwise, if it is located within the blue area, it does not match with a sync message of the PC. Then, we can calculate the probability of one mobile phone's message not matching with the personal computer's message, the probability being equal to the ratio of the blue area's length to the average time interval (i.e., $((T_m - 2\Delta t)/T_m)$). We denote the number of mobile phones as $|Y|$, and there are $|Y| - 1$ mobile phones which are uncorrelated with the PC. The probability of $|Y| - 1$ phones' message not matching with the personal computer's message is

$$\left(\frac{T_m - 2\Delta t}{T_m}\right)^{|Y|-1}. \tag{6}$$

Furthermore, we can calculate the maximum value of θ_0 which means the probability of at least one mobile phone's message matching with the personal computer's message:

$$\theta_{0\_\max} = 1 - \left(\frac{T_m - 2\Delta t}{T_m}\right)^{|Y|-1}. \tag{7}$$

According to this equation, if T_m is the minimum value (i.e., 10), $\theta_{0\_\max}$ gets the maximum value. And we have $\theta_0 \leq \theta_{0\_\max}$ normally.

In addition, as different networks have different latencies, we also find that most of time gaps between two sync messages may be bigger than Δt in some network environments. In order to deal with this situation, we define a time offset T_{offset}, which equals the average time delay of the sync messages between two devices. We require to collect the ground-truth dataset in the native network to measure T_{offset} in advance. Then, the sync message-matching interval converts to $[T_{\text{offset}} - \Delta t, T_{\text{offset}} + \Delta t]$. Specifically, T_{offset} has no effect on the calculation of $\theta_{0\_\max}$.

To show the matching rounds in one correlation, we can compute the expected number of sync message-matching with reference to Gu et al. [14] and Wald [5]. When we want to make a decision that two devices are correlated, the expected number of sync message-matching rounds we need to observe is

$$E[N \mid H_1] = \frac{\beta \ln(\beta/(1-\alpha)) + (1-\beta)\ln((1-\beta)/\alpha)}{\theta_1 \ln(\theta_1/\theta_0) + (1-\theta_1)\ln((1-\theta_1)/(1-\theta_0))}. \tag{8}$$

If we want to make a decision that two devices are not correlated, the expected number of matching rounds is

$$E[N \mid H_0] = \frac{(1-\alpha)\ln(\beta/(1-\alpha)) + \alpha \ln((1-\beta)/\alpha)}{\theta_0 \ln(\theta_1/\theta_0) + (1-\theta_0)\ln((1-\theta_1)/(1-\theta_0))}. \tag{9}$$

The detailed calculation process of these two formulas is in [5]. We analyze the possible values of the four parameters and substitute them into the formula and then we plot Figure 5 to show how $E[N \mid H_1]$ changes with the parameter change. When we fix $\beta = 0.01$ and vary α, θ_0, and θ_1, we can see that if θ_0 increases or θ_1 decreases, it demands more messages to make a decision of whether two devices are correlated. We assume that there are 8 people in one correlation (i.e., $|Y| = 8$), and we have $\theta_{0\_\max} \approx 0.35$. Therefore, when we set $\theta_1 = 0.9$, $\theta_0 = 0.35$, and $\alpha = 0.01$, we can get $E[N \mid H_1] \approx 7$ and $E[N \mid H_0] \approx 5$. It means that, after observing an average of 7 rounds of sync message-matching, we can make a decision H_1 (correlated); otherwise, we can accept H_0 (uncorrelated) after observing 5 rounds of sync message-matching.

Although the SPRT algorithm can help us to make a quick decision, there are still two cases that cannot be solved by employing it. One case is that if we detect more than one mobile phone correlated with a computer, then how to distinguish the right mobile phone? The other case is that the number of the sync messages of one device is not sufficient enough to make a decision (i.e., correlated or not). In order to deal with these cases, we employ the Jaccard index [15]. We denote the matched sync message set of the kth PC and the ℓth mobile phone as $Z^{k\ell}$ and the size of this set as $|Z^{k\ell}|$. As mentioned before, the sync message set of the kth personal computer and the ℓth mobile phone is denoted as X^k and Y^ℓ, respectively, and their size is denoted as $|X^k|$ and $|Y^\ell|$, respectively. According to the definition of the Jaccard index, we have

$$J\left(X^k, Y^\ell\right) = \frac{\left|Z^{k\ell}\right|}{\left|X^k\right| + \left|Y^\ell\right| - \left|Z^{k\ell}\right|}. \tag{10}$$

In order to ensure that the two devices have sufficient correlation, we only calculate the value of the Jaccard index if $Z^{k\ell} \geq 10$ and choose the biggest one as the final matching result.

4. Evaluation

In this section, we introduce the settings of our experiments and evaluate the performance of our cross-device tracking approach. We also perform the experiments to discuss the factors which can affect the performance of our approach.

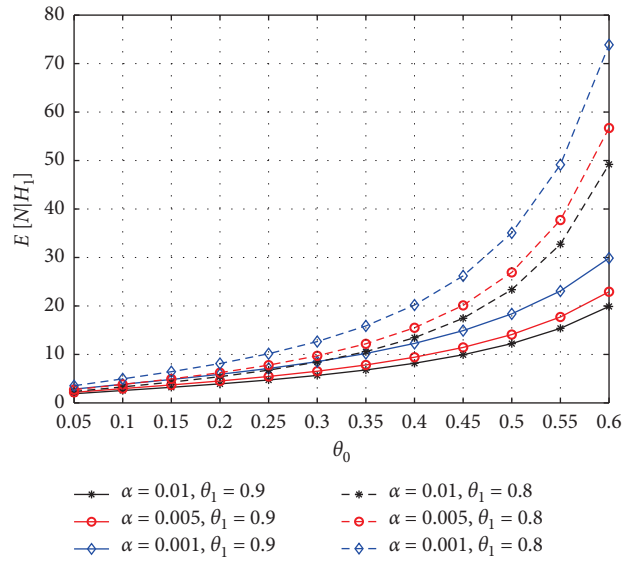

Figure 5: Values of $E[N \mid H_1]$ and $\beta = 0.01$.

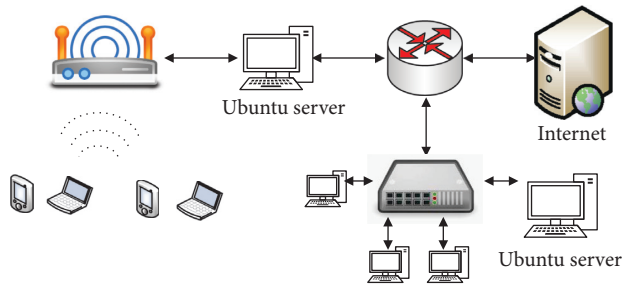

Figure 6: Experimental setup.

4.1. Experimental Setup. We recruit 23 participants and set up an experimental platform to connect their devices to the internet and capture the traffic. The experimental setup is shown in Figure 6. Desktops used by the participants are connected to a switch that connects to the internet via our campus network. We configure a mirror port of the switch which is connected by the first Ubuntu server used to capture traffic of the desktops. The mobile phones or laptops of the participants are connected to a wireless route that is set as a bridge mode. Then, the wireless route is connected to another Ubuntu server which has two network cards configured with a bridge mode. Since the traffic of the wireless devices passes through the second Ubuntu server, the server can capture all traffic of the wireless devices. The IP addresses of all the devices are assigned from a Dynamic Host Configuration Protocol (DHCP) server in the campus network. Therefore, we can use the IP addresses to identify each device in advance so as to collect the ground-truth traffic. We also install a Network Time Protocol (NTP) server [16] in the first Ubuntu server and synchronize the second Ubuntu server's time with that of the first one.

We require the 23 participants to use the five IM applications on their own devices so as to derive the ground-truth and testing traffic. All of the mobile phone OSes used by the participants are Android OSes, and all of the PC OSes are Windows OSes. The participants are required to chat with each other by logging in the same IM application both on a mobile phone and a computer. The IM applications include WhatsApp, Facebook Messenger, WeChat, QQ, and Skype. Four of the applications, except Facebook Messenger, have both Android and Windows versions. The participants use the Facebook Messenger chat widget on the Facebook website through a browser on their PCs. We capture the network traffic and save it in the pcap format.

4.2. Experimental Results. We first collect ground-truth data to derive the parameters which are mentioned in our approach. Then, we perform experiments to evaluate the effectiveness of different apps' sync message identification. After that, we implement device correlation experiments to verify our approach's ability in cross-device tracking. We also discuss the factors which can affect the results.

4.2.1. Parameter Settings. We first collect sync message ground-truth data and conduct experiments to derive the optimally empirical packet interval time threshold T_{interval} so as to accurately segment the sync message from the traffic. We invite 10 participants who are divided into five groups, and each participant has the same IM account logged in on two devices (e.g., a mobile phone and a personal computer). We let each participant employ each app to send 10 messages to their partner and let the receiver record the timestamps of the messages. At last, we obtain a ground-truth dataset of 100 pairs of IM sync messages with receiving time for each IM application. We locate the IM sync message packet sequences based on the recording timestamps and employ the time threshold to segment the traffic flow. We denote the correct segmentation as the sync message packet sequence is not separated, and the background traffic is separated from the sync message packet sequence. Figure 7 illustrates the relationship between different time intervals and the accuracy of the sync message segmentation. The accuracy is calculated by the number of correctly segmented sync messages divided by the number of all sync messages. As we can see from the figure, the accuracy approaches around 100% when T_{interval} is equal to 1.2 s. Therefore, we choose $T_{\text{interval}} = 1.2$.

We perform statistical analysis on the ground-truth data and derive an appropriate threshold Δt which is used to determine whether a sync message of a mobile phone is matched with a specific sync message of a PC and then derive $\theta_1 = 0.9$ in terms of Δt. To evaluate the value of Δt, we calculate the time gap between the timestamps of a pair of two sync messages from two devices according to the messages' timestamp in the network traffic. Figure 8 shows the cumulative distribution function (CDF) of 500 pairs of sync messages' time gaps collected from five IM applications. Since large Δt can lead to a number of false matches, we choose $\Delta t = 0.3$. Then, about 90% of the sync message pairs can be matched using this threshold, and we, therefore, can derive $\theta_1 = 0.9$.

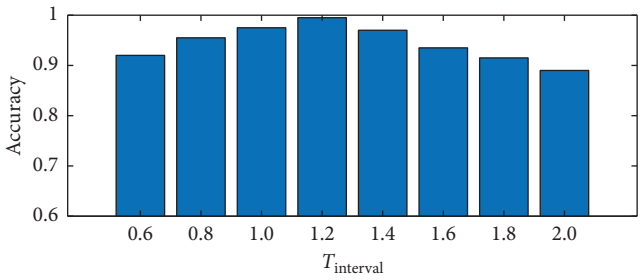

Figure 7: The result of segmenting sync messages at different time intervals.

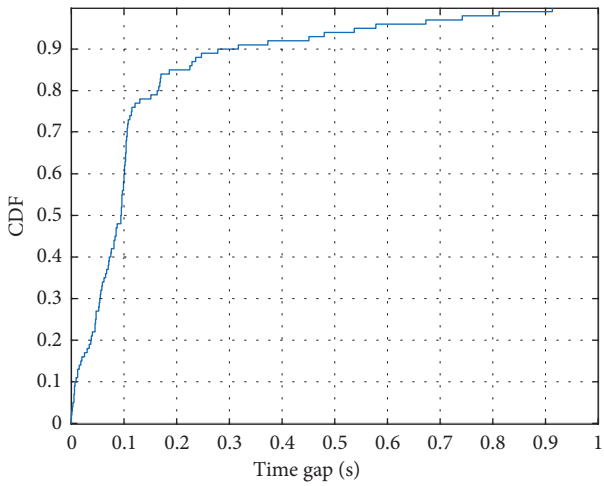

Figure 8: The CDF of the time gap between sync messages.

4.2.2. Sync Message Identification. In the next step, we collect testing data and evaluate the performance of sync message identification. In this set of experiments, we divide 10 participants into 5 groups and let them chat with his partner in the same group. We ask them to send 10 messages to their partner with a time interval (e.g., 10 seconds), which ensures that the network traffic of two sequential messages does not interfere with each other. The receivers record the timestamps of the received messages as the ground truth. Then, we obtain a dataset of 100 pairs of IM sync messages with receiving time for each IM application. At last, we evaluate our sync message identification method mentioned in Section 3.

To evaluate the performance of our sync message identification method, we define some evaluation indicators (i.e., precision, recall, and F_1-score). Precision is equal to the number of correctly identified sync messages divided by the number of all identified sync messages. Recall is equal to the number of correctly identified sync messages divided by the actual number of sync messages. F_1-score is

$$F_1 - \text{score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}. \quad (11)$$

Table 5 illustrates the F_1-score of the sync message identification method of QQ, WeChat, WhatsApp, and Facebook Messenger. As we can see from this table, QQ and WhatsApp have better performance on the PC version, while WeChat and Facebook Messenger have better performance on the phone version. From the perspective of the application type, the sync message identification performance of QQ is the best due to its special protocol and stable features. On the contrary, we find that the sync messages of WhatsApp are more difficult to be distinguished from background traffic which leads to a lower F_1-score. The other two apps have similar identification performance.

Table 5: Sync message identification result of four applications.

Device type	Indicator	QQ	WeChat	WhatsApp	Facebook Messenger
PC	Precision	94.3%	81.4%	95.3%	86.8%
	Recall	99.0%	96.0%	82.0%	92.0%
	F_1-score	0.966	0.881	0.882	0.893
Mobile phone	Precision	88.2%	90.5%	80.3%	88.3%
	Recall	97.0%	95.0%	94.0%	98.0%
	F_1-score	0.924	0.927	0.866	0.929

Table 6: Sync message identification result of Skype.

F_1-score	PC	Mobile phone
XGBoost	0.820	0.892
Random forest	0.786	0.796

Table 6 depicts the F_1-score of the Skype sync message identification method. In order to identify Skype sync messages, we collect 400 sync messages of the PC and 400 sync messages of the mobile phone to train models. In addition, we preprocess our data used for XGBoost by a data standardization function "scale" in Python scikit-learn library. Then, we employ XGBoost and random forest as the classification methods. The performance of XGBoost is better than that of random forest, and we therefore choose XGBoost as the Skype sync message identification method in device correlation experiments.

4.2.3. Correlation Results. We evaluate our cross-device tracking approach and compare several scenarios to analyze different factors. Considering the situation of wireless router capacity and network speed, we set one experiment consisting of 8 people and 16 devices. When doing experiments, all participants connect their devices to our network and log in the designated apps. After they chat with each other for a while (e.g., 30 minutes), we analyze their network traffic to correlate their devices. In order to make a decision of whether two devices are correlated or not correlated (i.e., make a decision of accepting H_1 or H_0), we require to set four parameters (i.e., α, β, θ_0, and θ_1). Here, 8 people (i.e., $|Y| = 8$) lead to $\theta_{0\_\max} = 0.352$. Then, we set $\theta_1 = 0.9$, $\theta_0 = 0.35$, $\alpha = 0.01$, and $\beta = 0.01$, and after the calculation, we have $\ln(\theta_1/\theta_0) \approx 0.944$, $\ln((1-\theta_1)/(1-\theta_0)) \approx -1.872$, $\eta_u \approx 4.595$, and $\eta_l \approx -4.595$. In addition, we count the time length of correlating devices in our experiment to evaluate the speed of our approach. We also discuss some factors such

Table 7: Correlation result of five applications.

Correlation accuracy	QQ	WeChat	WhatsApp	Facebook Messenger	Skype
8 people	91.7%	97.9%	89.6%	85.4%	83.3%
16 people	80.2%	86.5%	80.2%	77.1%	72.9%

as different users, message frequency, and user amount which can make an influence on the result.

At last, we get 30 sets of data, with each app having 6 datasets. For each app, we have 8 device pairs in one experiment, and totally, we have 240 device pairs in these experiments. Every set of the experiment lasts dozens of minutes so that we can have enough time to correlate each device pair. The first row of Table 7 shows the correlation result of each application, noting a 12.5% baseline with 8 participants. The correlation accuracy is equal to the number of correctly correlated device pairs divided by the number of all device pairs. We can find that WeChat achieves the best performance with only one device pair matching failure in 48 pairs. QQ and WhatsApp achieve about 90% correlation accuracy. We also notice that the lowest correlation accuracy of Skype is more than 80%. Moreover, in these experiments, we let participants chat with other participants freely. It means the message-receiving frequency is not under control, and the traffic of sequential messages may mix with each other. This phenomenon results in the decrease of the sync message identification accuracy which further leads to the decrease of the correlation accuracy. From the result, we can conclude that WhatsApp, Facebook Messenger, and Skype are impacted by the factors mentioned above.

(1) Time Length. In order to study the distribution of matching time, we draw a boxplot of different device pairs' matching time of different applications in Figure 9. In this diagram, we can see that QQ and WeChat require less matching time compared with the other three applications and also have a relatively average time distribution, which is caused by two factors from our point of view. One is sync message matching accuracy; high matching accuracy leads to less matching time. The other is receiving message frequency, which is related to the number of user's friends. In our experiments, participants have more friends in QQ and WeChat, so they can receive more messages within the same length of time. The average matching time of QQ is about 290 seconds and WeChat is around 340 seconds, which means our cross-device tracking approach is fast enough. In addition, according to the theoretical expected number of matching rounds, the actual matching time is longer. We think there are two reasons: one is the actual message-receiving time interval is longer, and the other is misrecognition of sync messages leading to a lower message synchronization probability.

(2) Different Users. To make a comparison, when doing experiments, we group participants randomly, and each group consists of 8 people with 16 devices. The results of different groups are shown in Figure 10, with different colors meaning different groups; in a word, the differences between

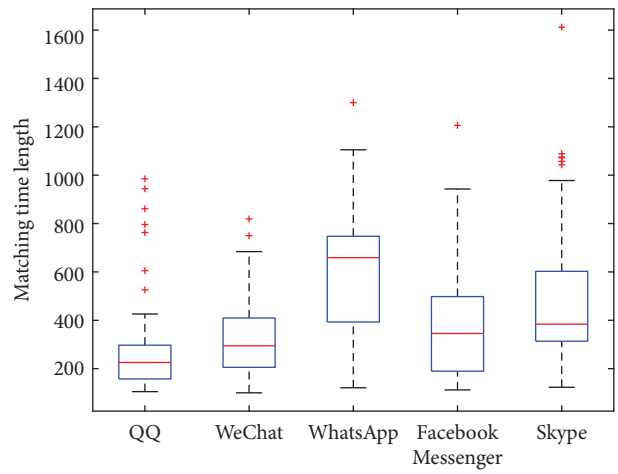

Figure 9: Matching time length of different applications.

the groups are not significant. Therefore, our approach is generally applicable to different users.

(3) Message Frequency. In this experiment, we try to identify the impact of different message-receiving frequencies. Here, we choose WeChat and Skype as the apps to be tested. We do an 8-people experiment and let participants send messages to their partner in 5 minutes with different message frequencies (i.e., 15, 20, 25, and 30). Altogether, we have four groups of statistics from both WeChat and Skype. The correlation result is shown in Table 8. For WeChat, when the sync message frequency is relatively low (i.e., 15 and 20), we can correlate all 8 device pairs successfully, and while the sync message frequency is comparatively high (i.e., 25 and 30), there is one device pair linking failure. For Skype, the increasing message frequency decreases the correlation accuracy too. As far as we know, an increase in the message frequency leads to a rise in the probability of mismatching messages, which further results in the decrease of the correlation accuracy.

(4) User Amount. In this set of experiments, we simulate a scenario with more people to make a comparison with the 8-people experiment. In order to simulate the tracking scenario of more device pairs, we subtract each message list's initial time from the total message time and then put data from different batches together to link. To make a comparison with the 8-people experiment, we do new 8-people experiments and combine two of them to form a 16-people experiment; then, we also derive 30 datasets of the 16-people experiment and totally 480 device pairs in this experiment. In this set of experiments, 16 people (i.e., $|Y| = 16$) lead to $\theta_{0\_\max} \approx 0.606$. Therefore, we set $\theta_0 = 0.6$, $\theta_1 = 0.9$, $\alpha = 0.01$,

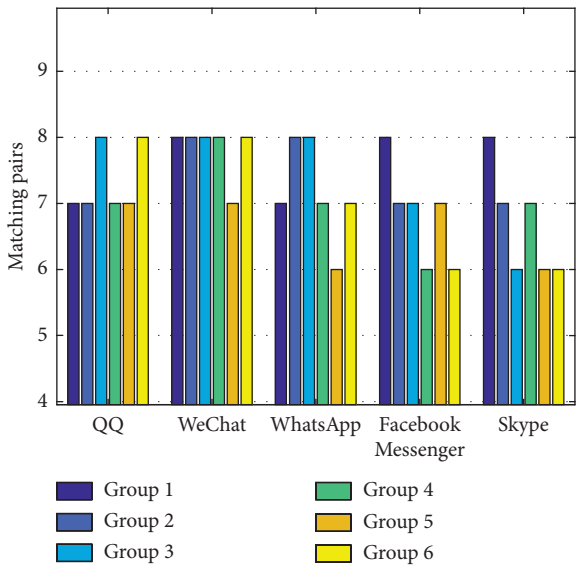

Figure 10: Matching result of different groups.

Table 8: Correlation result of different frequencies.

Correlation accuracy	15	20	25	30
WeChat	100%	100%	87.5%	87.5%
Skype	100%	87.5%	87.5%	75%

and $\beta = 0.01$, and thus, we have $\ln(\theta_1/\theta_0) \approx 0.405$, $\ln((1-\theta_1)/(1-\theta_0)) \approx -1.386$, $\eta_u \approx 4.595$, and $\eta_l \approx -4.595$ after the calculation. The correlation results are shown in Table 7, the statistics of the second row indicating that the accuracy of each app decreases about 10 percent compared with the first row. It is due to that, with the increase of the user amount, the possibility of sync message mismatching will increase, which results in the decrease of the linking accuracy.

5. Discussion and Future Work

In this section, we put forward some measures to promote our approach in the future and give some countermeasures for users to avoid devices being correlated.

5.1. Ways of Promoting Our Approach. In our experiments, we ask participants employing the same app to send messages over the same period of time, the purpose of which is to ensure the sufficient quantity of equipment so as to evaluate the performance of our approach. However, people are not likely to employ the same app to send instant messages at the same time in the real world. Therefore, we can separate different devices which do not have time overlap of network traffic and classify the devices which run different IM apps during the preprocessing period. After that, devices will be divided into different groups, which reduce the device number of one matching group, thus improving the correlation accuracy. It is important to note that our approach is universal for apps which have message synchronization function. Therefore, as long as the other IM app synchronizes messages between devices, our approach can correlate them. In particular, if the suspect employs a niche app which is out of our chosen apps and unpopular, it may cause the suspect's devices, generating the rare network traffic of the niche app in a local area network. In that case, we can correlate his different devices by identifying the network traffic of the niche app.

During our study, studying different message types is also helpful to promote our approach. As we know, the instant messaging app can send messages in various forms such as text, voice, picture, and file. If we can utilize the message type as an attribute in the period of sync message matching, it can reduce message mismatching. However, after analyzing the features of different messages, we find that there are some difficulties to employ them. One difficulty is that delivering a file or a picture message requires more than one server to work, i.e., one server sends a message notification first, and then another server sends the message content. Apart from that, different message-receiving strategies of devices also make them difficult to utilize these types of messages. For example, when the mobile phone version of QQ receives a voice message, it starts to receive the moment when the sender starts to record the voice. However, the PC version of QQ starts to receive when the sender stops the recording process. These difficulties will result in more complex identification of various messages and poorer message synchronization. In the future, we will figure out how to utilize different message types to correlate devices.

5.2. Countermeasures. In this paragraph, we give some advices to prevent the devices from being correlated. First, IM application manufacturers can modify message delivering packets to interfere sync message identification. They can add useless messages to the gate server's flow, add random packets to the message packet sequence, or add random padding to the message packets. Second, users can avoid using two devices to log in the same account, which is the simplest method from the aspect of users. However, users sometimes have to log in two devices at the same time. In that circumstance, they can choose not to keep online for a long period of time so that there will not be enough messages to correlate devices. Moreover, another way is choosing a suitable network connection mode, which means mobile phones can use 3G/4G to connect the internet to avoid their network traffic being captured.

6. Related Work

Cross-device tracking is developed from device tracking. In this section, we will introduce related research development.

According to the techniques employed in device tracking, there are two main categories, which are network traffic analysis-based tracking and web-based tracking. In the field of network traffic analysis, researchers devote to extracting attributes from network traffic to identify the device or users even though the device IP changes. In the

physical layer, Polcak et al. [17] proposed a kind of attribute to identify computers by their clock skew computed from TCP timestamps. In the operating system layer, Franklin and McCoy [18] found that the 802.11 probe request time interval can reveal the active scanning algorithms employed by wireless derivers. In the network layer, the traffic bursts of mobile devices are counted to analyze the running applications [19]. In the application layer, the fields of unencrypted network traffic such as user-agent, IP address, cookies, and user IDs [20] are abstracted to identify devices. Furthermore, Gu et al. [21] found that the search history of the shopping website can also be used to track users. In addition, the DNS traffic can be used to analyze the device user's activities which can be used to track users [22].

Web tracking is based on acquiring attributes from the browsers to identify devices. It is widely used to serve the websites for statistics, tracking, and advertisement recommendation [23, 24]. Eckersley [25] first proposed this method in 2010 PETS. The author extracted the user-agent, HTTP header, screen size, fonts, and plugins from the browsers. Some researchers are interested in analyzing the tracking mechanism of commercial websites; Acar et al. [26] found that the canvas fingerprint is the most widely used web-tracking technique. Besides, Diaz et al. [27] proposed that the Battery Status API of HTML5 can be used to identify the browser. Compared to desktop browsers, the mobile device browsers, which lack plugins and some functions, therefore, achieve less information. It is not until in 2016 S&P conference that Laperdrix et al. [28] applied fingerprint identification technology to mobile terminal browser identification on a large scale. The results show that although plugin list and font information are lacking, the recognition rate still reaches 81%. With the further improvement of browser functions, Bojinov et al. [29] presented a method of achieving accelerometer data from the mobile device browser to realize identification. Then, Das et al. [30] employed the gyroscope and microphone to evaluate the effect of utilizing sensors to identify mobile devices.

Based on device tracking, cross-device tracking aims to correlate different types of devices to the same person. A recent study showed that most people operate multiple devices in daily use, and they often accomplish a task by switching devices (https://www.thinkwithgoogle.com/research-studies/the-new-multi-screen-world-study.html). The advertising companies need to collect the browsing history from different devices, which can help to recommend more comprehensive advertising. Cross-device tracking can be divided into two categories, which are deterministic tracking and probabilistic tracking [1]. Deterministic tracking requires the user to log in the same account on different devices. Then, the service provider is able to correlate different devices by the explicit identifiers such as account number and cookies. However, many apps can be used without logging in a user account, and the cookies probably cleared by users cannot be obtained each time.

Most researchers focus on probabilistic cross-device tracking. The feasibility of these methods is based on the similarity of user interests and activities when they operate different devices. For example, Kane et al. [2] found that 97.1% domains browsed on the mobile devices are also visited on the desktop. Also, 13.1% domains visited on the desktop are browsed on the mobile devices. During ICDM 2015 [3] and CIKM 2016 [4], the participants were asked to propose machine learning methods to do cross-device tracking. The datasets included many users' devices, cookies, IP addresses, and browsing history. These two competitions only focused on the algorithm design, such as using pairwise ranking [6, 7]. In the 2017 USENIX Security International Conference, Zimmeck et al. [1] indicated that online tracking is evolving from browser- and device-tracking to people-tracking. As one user tends to operate multiple devices, a person-centric view is established for cross-device tracking. In this paper, 126 users' browsing history of mobile devices and desktops was collected. The results showed that the IP address plays an important role in cross-device tracking. Choo [31] proposed a method of utilizing users' social media feeds to realize cross-device tracking. This method is based on the idea that each person has a distinctive social network, and thus, the links appearing in one's social media feeds are unique. However, this approach only works for a small number of social networks, which makes it impractical to be widely analyzed. In addition, there are some cross-device tracking approaches that do not analyze web communication, such as detecting inaudible ultrasonic sound embedded in websites [32]. Considering that using mobile phone's microphone demands user permission, Matyunin et al. [33] found that the gyroscope has a reaction to resonance frequencies in the frequency domain which means it is zero-permission tracking. Nevertheless, these approaches can only detect different devices which are physically close, and the devices are not necessarily owned by the same user. Moreover, some researchers paid attention to measure the cross-device tracking activity of commercial websites [34, 35] and found that cross-device tracking is widely used in many websites. Our work, which does not need long-time historical data, is a novel cross-device approach that is based on the network traffic analysis and easier to implement.

7. Conclusion

In this paper, we propose a novel cross-device tracking approach based on network traffic analysis. The premise of our approach is that users log in two devices with one IM account, and the two devices will receive messages simultaneously. We analyze the mechanism of devices' receiving sync messages and find that we can identify sync messages to correlate devices. In our scenario, we assume the law enforcement can sniff users' network traffic without users' active participation or long online time. Then, we extract features of five popular IM apps' received messages. In network traffic processing, we filter out useless flow and identify gate server's flow according to the server IP. After that, we employ rule matching and machine learning to identify sync messages. At last, we employ the SPRT algorithm to determine whether two devices are correlated according to the sync message-matching results. To evaluate our approach, we design contrast experiments with an 8-participant experiment and a 16-participant experiment. We find that the increasing participant amount

will decrease the matching accuracy, and different apps achieve different matching accuracies, with WeChat getting the highest matching accuracy, which is 97.9% (8 participants) and 86.5% (16 participants) and Skype receiving the lowest matching accuracy, which is 83.3% (8 participants) and 72.9% (16 participants). We also study different users who make little impact on the results, while the increase of the message frequency will reduce the correlation accuracy. At the end of this paper, we discuss how to promote our approach in the future, and we give some advice to overcome this cross-device tracking problem.

Data Availability

The network traffic data used to support the findings of this study have not been made available because they contain a lot of privacy information.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the National Key R&D Program of China (nos. 2018YFB0803400, 2017YFB1003000, and 2018YFB2100300); the National Natural Science Foundation of China (nos. 61972088, 61632008, 61572130, 61532013, and 61702097); the Jiangsu Provincial Natural Science Foundation for Excellent Young Scholars (no. BK20190060); Jiangsu Provincial Key Laboratory of Network and Information Security (no. BM2003201); and the Key Laboratory of Computer Network and Information Integration of the Ministry of Education of China (no. 93K-9).

References

[1] S. Zimmeck, J. S. Li, H. Kim, S. M. Bellovin, and T. Jebara, "A privacy analysis of cross-device tracking," in *Proceedings of the 26th USENIX Security Symposium*, pp. 1391–1408, Vancouver, Canada, 2017.

[2] S. K. Kane, A. K. Karlson, B. R. Meyers, P. Johns, A. Jacobs, and G. Smith, "Exploring cross-device web use on PCS and mobile devices," in *Proceedings of the International Conference on Human Computer Interaction*, pp. 722–735, Uppsala, Sweden, 2009.

[3] ICDM, "Drawbridge cross-device connections," 2015, https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections/overview.

[4] C. Cup, "Track 1: cross-device entity linking challenge," 2016, http://cikm2016.cs.iupui.edu/cikm-cup/.

[5] A. Wald, *Sequential Analysis*, Dover Publications, Mineola, TX, USA, 2004.

[6] J. Walthers, "Learning to rank for cross-device identification," in *Proceedings of 2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pp. 1710–1712, Atlantic City, NJ, USA, 2015.

[7] M. C. Phan, A. Sun, and Y. Tay, "Cross-device user linking: URL, session, visiting time, and device-log embedding," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 933–936, Tokyo, Japan, 2017.

[8] R. B. Jennings, E. M. Nahum, D. P. Olshefski, D. Saha, Z.-Y. Zon-Yin Shae, and C. Waters, "A study of internet instant messaging and chat protocols," *IEEE Network*, vol. 20, no. 4, pp. 16–21, 2006.

[9] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 439–454, Saarbrücken, Germany, 2016.

[10] N. Guo, J. Luo, Z. Ling, M. Yang, W. Wu, and X. Fu, "Your clicks reveal your secrets: a novel user-device linking method through network and visual data," *Multimedia Tools and Applications*, vol. 78, no. 7, pp. 8337–8362, 2019.

[11] T. Chen and C. Guestrin, "Xgboost: a scalable tree boosting system," in *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 785–794, San Francisco, CA, USA, 2016.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pp. 211–225, Berkeley, CA, USA, 2004.

[14] G. Gu, J. Zhang, and W. Lee, "Botsniffer: detecting botnet command and control channels in network traffic," in *Proceedings of the 2008 Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2008.

[15] P. Jaccard, "Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines," *Bulletin de la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 241–272, 1901.

[16] D. Mills, *Computer Network Time Synchronization-The Network Time Protocol*, CRC Press, Boca Raton, FL, USA, 2006.

[17] L. Polcak, J. Jirasek, and P. Matousek, "Comment on "remote physical device fingerprinting"" *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 494–496, 2014.

[18] J. Franklin and D. McCoy, "Passive data link layer 802.11 wireless device driver fingerprinting," in *Proceedings of the 15th USENIX Security Symposium*, Vancouver, Canada, 2006.

[19] T. Stöber, M. Frank, J. B. Schmitt, and I. Martinovic, "Who do you sync you are?: smartphone fingerprinting via application behaviour," in *Proceedings of the 6th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*, pp. 7–12, Budapest, Hungary, 2013.

[20] T. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, "Host fingerprinting and tracking on the web: privacy and security implications," in *Proceedings of the 2012 Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2012.

[21] X. Gu, M. Yang, C. Shi, Z. Ling, and J. Luo, "A novel attack to track users based on the behavior patterns," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 6, 2017.

[22] D. Herrmann, C. Banse, and H. Federrath, "Behavior-based tracking: exploiting characteristic patterns in DNS traffic," *Computers & Security*, vol. 39, pp. 17–33, 2013.

[23] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: policy and technology," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pp. 413–427, San Francisco, CA, USA, 2012.

[24] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: exploring the

ecosystem of web-based device fingerprinting," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pp. 541–555, Berkeley, CA, USA, 2013.

[25] P. Eckersley, "How unique is your web browser?" in *Proceedings of the 10th International Symposium on Privacy Enhancing Technologies (PETS)*, pp. 1–18, Berlin, Germany, 2010.

[26] G. Acar, C. Eubank, S. Englehardt, M. Juárez, A. Narayanan, and C. Díaz, "The web never forgets: persistent tracking mechanisms in the wild," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 674–689, Scottsdale, AZ, USA, 2014.

[27] C. Diaz, L. Olejnik, G. Acar, and C. Casteluccia, "The leaking battery: a privacy analysis of the html5 battery status api," *Lecture Notes in Computer Science*, vol. 9481, pp. 254–263, Springer, Berlin, Germany, 2015.

[28] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: diverting modern web browsers to build unique browser fingerprints," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P)*, pp. 878–894, San Jose, CA, USA, 2016.

[29] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile device identification via sensor fingerprinting," 2014, https://arxiv.org/abs/1408.1416.

[30] A. Das, N. Borisov, and M. Caesar, "Tracking mobile web users through motion sensors: attacks and defenses," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2016.

[31] Y. S. Choo, *Cross-device tracking of employees with social networks*, Ph.D. dissertation, Imperial College London, London, UK, 2017.

[32] K. Waddell, "Your phone is listeningłliterally listeningłto your TV," 2015, https://www.theatlantic.com/technology/archive/2015/11/yourphone-is-literally-listening-to-your-tv/416712/.

[33] N. Matyunin, J. Szefer, and S. Katzenbeisser, "Zero-permission acoustic cross-device tracking," in *Proceedings of 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 25–32, Washington, DC, USA, 2018.

[34] J. Brookman, P. Rouge, A. Alva, and C. Yeung, "Cross-device tracking: measurement and disclosures," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 133–148, 2017.

[35] K. Solomos, P. Ilia, S. Ioannidis, and N. Kourtellis, "Talon: an automated framework for cross-device tracking detection," 2019, https://arxiv.org/abs/1812.11393.