CFDMA: A Novel Click Fraud Detection Method in Mobile Advertising

1st Yanglin Liu
Department of Computer Science and Technology
Harbin Institute of Technology, Shenzhen
Shenzhen, China
19s151086@stu.hit.edu.cn

2nd Yang Liu
Department of Computer Science and Technology
Harbin Institute of Technology, Shenzhen
Shenzhen, China
liu.yang@hit.edu.cn

3rd Mehdi Gheisari
Department of Computer Science and Technology
Harbin Institute of Technology, Shenzhen
Shenzhen, China
mehdi.gheisari61@gmail.com

***Abstract*—While the Cost-Per-click mechanism ensures the prosperity of the mobile advertising ecosystem, click fraud becomes a significant threat that results in huge losses for advertisers. On the one hand, some existing studies identify click fraud in a predefined test environment on the client side. However, these works neglect the influence of user interaction on fraudulent behaviours. On the other hand, some studies employ binary classifiers to detect click fraud on the server side without considering the particularity of the click fraud scenario. This paper presents the design and implementation of CFDMA, a click fraud detection system that conducts both server side and client side detection. CFDMA makes connections between fraudulent behaviours and explicit user inputs through observing sensitive system classes on the client side. CFDMA constructs the publisher graph on the server side and uses its information to detect invalid traffic. We evaluate the performance of CFDMA on real click datasets. Our evaluation demonstrates that CFDMA is capable of identifying invalid click traffic and fraudulent behaviours.**

***Index Terms*—Click fraud detection, Mobile advertising, Android, App testing, Invalid traffic**

I. Introduction

Mobile advertising is now one of the essential means of product promotion and is also the primary profit method of many app publishers. Global mobile advertising spending hit a record 288 billion USD in 2021, and it is expected to reach nearly 413 billion USD by 2024 [1]. Desktop advertising revenue has not changed much in recent years, mainly because mobile advertising has driven the growth of online advertising revenue [2].

In order to incentivize app developers to display advertisers' ads in apps, the Cost-Per-Click (CPC) mechanism [3], in which an advertiser pay per click on a given ad, is widely applied.

This work is supported by the Shenzhen Basic Research (General Project) (No. JCYJ20190806142601687), Shenzhen Stable Supporting Program (General Project) (No. GXWD20201230155427003-20200821160539001). Correspondence to: Yang Liu.

The profit of mobile advertising attracted the attention of participants who try to inflate incomes by fabricating user clicks [4], [5]. Click fraud, usually performed by malicious code or automatic bots, has threatened the mobile ad ecosystem. In 2020, ad fraud caused economic losses of 35 billion USD globally [6], [7].

The detection of click fraud in mobile advertising has attracted increasing attention due to the significant economic loss inflicted by click fraud. Detecting click fraud in mobile advertising can be characterized by two types: detecting fraud by emulating user interaction with apps on the client side and identifying invalid traffic on the server side.

This paper proposes CFDMA, which is an efficient and deployable mobile ad click fraud detection method, to identify click fraud at both the client side and server side. Click fraud is more likely to occur on the Android system due to its open environment and liberal policy compared to other mobile systems. Therefore, CFDMA focuses on click fraud detection in Android advertising to ensure a healthy advertising ecosystem. The goal of CFDMA is to automatically detect invalid traffic through the model as well as fraud behavior through black-box testing of Android apps. The client side and server side detection are orthogonal in CFDMA; any third parties can use CFDMA to detect click fraud based on their need. For example, advertisers can use CFDMA to identify whether Android apps embedding their modules or libraries have fraudulent behaviours. Researchers can employ CFDMA to check invalid click traffic by re-training our model on the new data set.

We implement CFDMA based on Xposed [8], word2vec [9], as well as Wide&Deep [10], and evaluate its performance using apps and datasets from the real world. The result shows that CFDMA can detect click fraud with high accuracy. The contributions of this paper are listed as follows:

1) We present the design and implementation of CFDMA. As far as we know, it is the first system that enables

efficient and deployable click fraud detection at both the client and server side.
2) We propose a client side detection method which identifies fraudulent behaviours by computing the causality between fraudulent events and direct user inputs.
3) We propose a hybrid model consisting of word2vec and Neural Network to focus on the inherent correlation between fraudulent publishers. Experimental results show that this approach can address the problem of fake clicks to a certain degree.

The rest of this paper is organized as follows. Section II discusses the related work. Section III proposes the design of CFDMA and briefly introduces the components in it. Section IV presents the evaluation of CFDMA, and finally, section V concludes this paper.

II. Background

A. Mobile advertising ecosystem

The mobile advertising ecosystem typically involves *Advertiser*, *Publisher*, *Demand Side Platform (DSP)*, *Supply Side Platform (SSP)*, and *Ad Exchange*. Advertisers design ads that will be distributed to user devices and use DSPs (*e.g.*, Tencent and Bytedance) to bid on ad slots. Advertisers focus on how successfully their ads can attract potential visitors' interest. The Publisher (*e.g.*, app developers) reverses ad slots in their apps to display ads and employs SSPs to sell reversed ad slots. The *Ad Exchange* (*e.g.*, Google and Tencent) serves as stock exchange, which allows *Advertisers* and *Publishers* to trade ad slots.

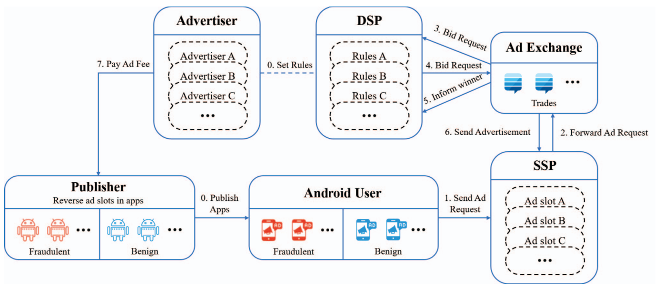

Fig. 1. Overview of the mobile advertising ecosystem.

Fig. 1 illustrates the typical workflow of serving an ad in mobile advertising. When a user uses a mobile app and meets an ad slot, an ad request is sent to the cooperated SSP, which would forward this request to the Ad Change. There is some information in the ad request, such as device ID, IP address, and location. The Ad Exchange will initiate an auction of the ad slot after receiving the ad request and then send the bidding request to registered DSPs. These DSPs bid on ad slots based on the interest of advertisers and then respond optimal prices to Ad Exchange. Upon receiving all expenses, the Ad Exchange will inform the winner and send the advertisement to the app.

B. Click Fraud in mobile advertising

App developers can earn interest according to the number of clicks on advertisements. This increased prominence attracts the attention of tech-savvy fraudulent developers/developers fabricating user clicks to trick advertisers into trusting that interested users have visited their ad [11], [12]. Some researchers point out that around 10% to 15% of clicks are not authentic traffic [13]–[15]. Multiple sources can create fabricated clicks. On the one hand, they can be produced by known invalid data-centre, botnets, spiders, or non-browser proxies; attackers use them to send fabricated click requests to targeted ad networks [15]. On the other hand, some of them can be caused by emulators, automated systems, real devices, and malware that conducts fraudulent actions and generate admissible click requests [5]. Unfortunately, clicker frauds have become a critical concern for users' experience and the investments of advertisers, especially the reputation of ad networks.

C. Click Fraud detection in mobile advertising

Several kinds of research have focused on click fraud in mobile advertising in recent years. MAdFraud [16] take the first step to execute Android apps in emulators to observe fraudulent behaviours to fight ad frauds. However, it uses a testing environment without user interaction that is different from the real world. ClickScanner [17] build data dependency graphs from the byte-code level, extract features from graphs, and train the autoencoder model to detect frauds. Some of the detection approaches analyze the ad requests at the network server. Clicktok [18], [19] proposes that unusual click-stream traffic is likely to be reused by legitimate traffic, so they try to identify patterns in the same click-stream of data. However, it does not consider detecting click fraud based on the relationship between fraudulent publishers. Thejas [20], [21] proposes combing Auto Encoder, Neural Network and Semi-supervised Generative Adversarial Network as a hybrid model to detect click fraud in the imbalanced scenario. Although these models can observe some fraud, they cannot effectively detect group fraud.

III. CFDMA Overview

To address ad frauds in the mobile advertising ecosystem, we design and implement CFDMA, a system that combines dynamic testing on Android apps as well as identifying fraudulent behavior from network traffic. At a high level, our system is implemented as a service in the background of the ad network or ad exchange to detect click fraud and ensure a healthy mobile advertising ecosystem.

In general, CFDMA consists of three components, as shown in Fig. 2. We consider each in turn.

1) In the traffic detection module, CFDMA proposes a hybrid model to discriminate between fraudulent clicks and benign clicks by exploiting the features obtained from the publisher graph.
2) CFDMA proposes identifying malicious invocation from user interaction to fraudulent behavior by Xposed in the dynamic testing module.
3) The management scheduler is employed with an administrator web page. Users of CFDMA can manage

the whole system, such as scheduling dynamic testing in several devices and viewing results classified by the traffic detection module via the web interface.

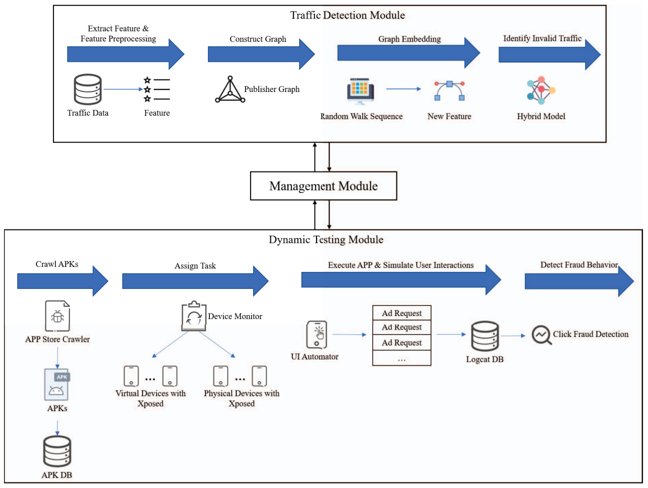

Fig. 2. The workflow of CFDMA

A. Traffic Detection Module

Click Fraud attackers often have a large number of devices and virtual IP addresses to carry out attacks in mobile advertising. Attackers can continuously change devices or IPs to commit click fraud on apps. Abnormal users are related to each other due to click records. In addition, the relationship between fraudulent users is strong. For example, multiple fraudulent users click on the same app, which can be regarded as a connection between fraudulent users within the fraudulent group. These connections can be described through a graph modelled by user click sequences. Therefore, we consider building a graph model of mobile advertisement clicks based on user click sequences. Then, we can use the information in the graph model to detect click fraud.

1) Publisher Graph: The relationship between users and publishers can be described as a graph based on the click sequence. As shown in Fig. 3, Attacker A and attacker B click on advertisements on several publishers, and this kind of behavior form an intersection that contains two publishers. The fraudulent actions point to both Publishers, which means these two publishers are likely to conduct click fraud. Therefore, we consider modelling the user's click sequence on publishers as a graph for more information.

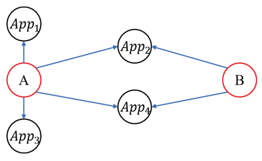

Fig. 3. Click fraud graph.

The process of forming the publisher graph is as follows:

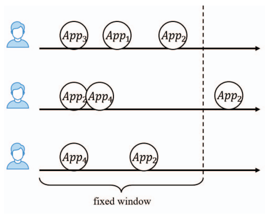

Fig. 4. User click sequence.

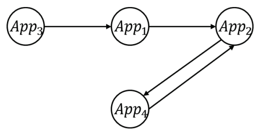

Fig. 5. Publisher graph.

1) Process the data of each user's click record on different publishers from the user dimension so as to extract the user's click sequence.
2) Use the user's click sequence for modelling. Instead of a complete behavior sequence of each user, we use a fixed window to segment behavior sequence into a multi-segment sequence, as shown in Fig. 4. There are two reasons for not using the complete behavior sequence. The first is the limitation of computing resources; the time and space complexity brought by the sequence is unacceptable. The second is to avoid the generation of non-correlated sequences.
3) After obtaining the user's click behavior sequence, the sequence is split into the node pair format of $App_i \rightarrow App_j$,which is used to construct edges of the graph.
4) Build the graph through the set of edges and vertices. As shown in Fig. 5, the user clicks on App_1, App_2 App_3, App_4, and App_3 in order.
5) Calculate the weights for each edge of the graph separately as (1), where $p_{(App_i,App_j)}$ is the count of the node pair.

$$w_{ij} = \frac{p_{(App_i,App_j)}}{P} \tag{1}$$

2) Publisher Graph Embedding: Typically, graph data structures are represented by adjacency matrices, which suffer from computational efficiency issues. The graph embedding method can convert the high-dimensional information of the graph into a low-dimensional vector representation while preserving the topology and node information of the graph. Compared with high-dimensional dense matrix representation, low-dimensional vectors can be used directly in the model.

The random walk is used to embed the weighted directed graph of publishers. The local topology of each node in the digraph is converted into sequence information through the random walk. Then sequences generated by random walk are input in word2vec to generate the embedding feature of each publisher node. The loss function of word2vec is listed as follows. The local context information of graph nodes can be obtained by random walk, and the processed features can also represent the correlation between vertices.

Typically, the probability of jumping to one of the connected nodes in the next step is the same because they embed on unweighted graphs. However, the publisher graph established in this paper is a weighted graph, and the weight information can be used as the basis for random walk jumps. The jump probability is defined as (2):

$$P\left(v_i \mid v_j\right)=\begin{cases}\frac{w_{ij}}{\sum_{v_k \in N_+\left(v_i\right)} w_{ik}}, & v_j \in N_+\left(v_i\right), \\ 0, & e_{ij} \notin E,\end{cases} \tag{2}$$

where w_{ij} means the weight of the edge connecting the node v_i and the node v_j, e_{ij} means The edge connecting the node v_i with the node v_j, $N_+\left(v_i\right)$ means the set of neighbor nodes of v_i.

3) Optimization On Graph Embedding: Click fraud attackers can mix fraudulent clicks with clicks on other publishers to hide the attack target. In addition, they can implement click fraud on the target publisher by replacing a large number of devices or changing the identifier (*e.g.*, IP address, IMEI). The graph embedding focuses on finding the relationship between vertices in mobile advertising. However, the connection between the fraudulent users is also worthy of attention. The graph embedding algorithms mentioned above do not focus well on the distributed click attack method. In addition, although the above graph construction method can model the user behavior of clicking ads multiple times, it does not pay attention to the user's single ad click behavior. An attacker can complete click fraud only once every time a device or IP is changed, then switch devices or IP addresses and repeat the process.

To compute the tightness between fraudulent publishers, we propose *publisher similarity*, which defines the tightness of connections between publishers. The publisher click user set is regarded as the representation vector of the publisher, and then the cosine distance of the vector is calculated. The definition of publisher similarity is defined as (3), where U_i and U_j represent the set of users who have clicked on App_i and App_j respectively. The physical meaning of user similarity lies in the fact that attackers have scattered click behavior on the fraudulent publisher, which leads to less crossover between dishonest publishers and regular publishers when calculating publisher similarity, making the distance between them smaller. The publisher similarity is different from the clustering algorithm. The publisher similarity is used to assist the model in learning the scattered click behavior. There are also normal users under the fraudulent publisher, and the clustering algorithm cannot be directly used to separate the fraudulent publisher from the normal publisher.

$$\operatorname{sim}(i, j)=\frac{\left|U_i \cap U_j\right|}{\left|U_i\right|\left|U_j\right|} \tag{3}$$

In the embedding process, the similarity of the embedding features needs to be close to the publisher similarity after embedding. The cosine similarity is used to measure the similarity of the embedding features of the publisher graph. Combined with the publisher similarity, a new loss function term can be obtained as (4), and the final loss function is defined as (5):

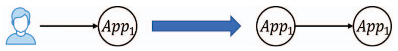

Fig. 6. The graph of attacker A and B.

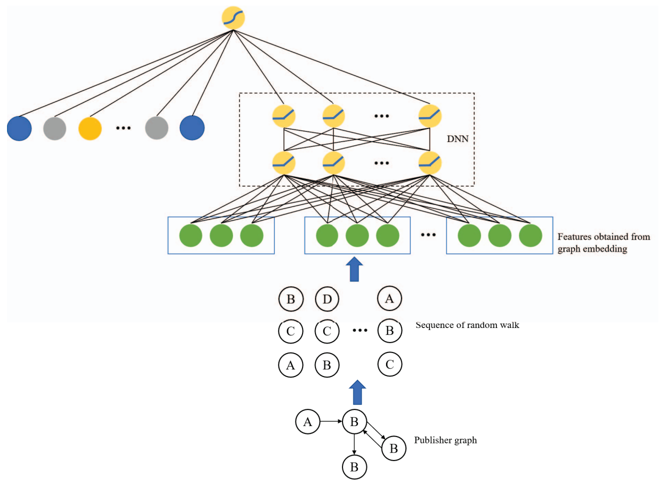

Fig. 7. The hybrid model.

$$\text{loss }=\sqrt{\left(\frac{\Phi\left(v_i\right) \cdot \Phi\left(v_j\right)}{\|\Phi(v_i)\|\|\|\Phi\left(v_j\right)\|}-\operatorname{sim}(i, j)\right)^2} \tag{4}$$

$$\begin{aligned} & \arg\min -\log \sigma\left(\Phi\left(v_j\right)^T \Phi\left(v_i\right)\right)- \\ & \sum_{t \in N\left(v_i\right)^{\prime}} \log \sigma\left(-\Phi\left(v_t\right)^T \Phi\left(v_i\right)\right)+\operatorname{los} s \end{aligned} \tag{5}$$

where $\Phi\left(v_i\right)$ is the mapping function to map node v_i to a vector.

If there are a large number of users clicking ads on a certain publisher, the publisher likely uses an IP proxy to conduct click fraud. For the single click record in the data set, the single click information is supplemented in the publisher graph by adding a symmetrical pair of publisher nodes, as shown in Fig. 6. This symmetric node pair can increase the publisher node's modulus value and reduce the distance and similarity with other normal publisher nodes. In the final feature space, publisher nodes with a large number of single advertisement records will be farther away from other nodes.

4) Hybrid Model Based On Graph Embedding: WideDeep jointly trains wide linear models and deep neural networks to combine the benefits of memorization and generalization. As shown in Fig. 7 We combine WideDeep with the process of graph embedding to get a hybrid model for click fraud detection. The wide component uses a logistic regression model, and the deep component uses DNN with two fully connected layers.

B. Dynamic Testing Module

The dynamic testing module of CFDMA can detect in-app click fraud, which is commonly caused by malicious code. In-app fraud invokes sensitive Android APIs to fabricate user clicks without involving user interaction. For example, invoking java.net.HttpURLConnection() to send a click URL request without user interaction. The existence of user interaction that causes the fraudulent behavior is the key point to detecting click fraud precisely.

The dynamic testing module considers two types of in-app click fraud, each emerging without a user input event. The first type programmatically fabricates a user click and lets the app send the click request. The second type calls request method, such as HttpURLConnection.connect(), to generate a forged ad click request without user interactions. We consider each in turn.

1) Detect Fabricated Clicks: A series of motion events are triggered and delivered to the app after a user clicks on ads in the app. The MotionEvent object contains movement data defined in its motion code, along with a set of axis values, and records a set of other motion properties.

In the application, the Activity.dispatchTouchEvent() method [22] is called to deliver the MotionEvent object [23] from the Android Activity to a View instance. The stack trace of Activity.dispatchTouchEvent should be unforged and contain only internal Android classes without developer-defined classes.

The dynamic testing module uses Xposed to hook the dispatchTouchEvent method to get movement data from the MotionEvent object and the stack trace of this method. The touch events (*e.g.*, drag, press, axis) input to devices is defined by this module. Therefore, the module first matches the input with movement data obtained from the MotionEvent object to detect fraudulent behavior.

In addition, attackers can invoke third-party libraries that precede the invocation of dispatchTouchEvent. For example, Fig. 8 shows the difference in the stack trace from the program entry to the motion event handler. The stack trace in the figure above shows that all the methods preceding dispatchTouchEvent are system classes, which means the stack trace is generated by real user interaction. However, the stack trace below contains several classes belonging to the third party, which forcibly invokes the touch event handler, thus generating forged user clicks. Therefore, the dynamic testing module tries to detect fraudulent behavior by identifying developer-defined classes in stack traces leading to dispatchTouchEvent.

2) Detect Forged Click Request: Click requests are sent to the ad network after users click ads. Therefore, some attackers conduct click fraud by generating fake click requests without user interaction, as shown in Fig. 9. Click requests could be forged by malicious codes written by developers or imported from malicious libraries. The dynamic testing module identifies this type of fraud by checking whether there is a user interaction event handler in stack traces and the action of sending click requests.

```
java.lang.Throwable: stack dump
    at java.lang.Thread.dumpStack
    at android.view.View.dispatchTouchEvent
    ...
    at android.view.ViewRootImpl$WindowInputEventReceiver.onInputEvent
    at android.view.InputEventReceiver.dispatchInputEvent
    ...
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run
    at com.android.internal.os.ZygoteInit.main
```

```
java.lang.Throwable: stack dump
    at java.lang.Thread.dumpStack
    at android.view.View.dispatchTouchEvent
    ...
    at com.xxx.ClickForge.click
    ...
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run
    at com.android.internal.os.ZygoteInit.main
```

Fig. 8. The difference of the stack trace that triggered by genuine user interaction and a fabricated click.

```
package com.xxx.xxx;
Public class AdActivity extends Activity {
     ...
     private void fabricateClickRequest() {
     ...
     URL url = new URL(ClickURL)
     HttpURLConnection connection = url.openconnection()
     connection.connect()
     ...
   }
}
```

Fig. 9. An example of forging click request.

The dynamic testing module needs to identify click requests and the stack trace that sends them. The module first intercepts all traffic by hooking the http-connection-related methods or classes in the system classes. For example, in Fig. 10, we hook the HttpGet class in org.apache.http.client.methods to our class to record the information of interest into logs. Then we take the HTTP classifier [24] to identify the ad click requests. If the classifier identifies a click request made by a connection-related method, the dynamic testing module checks the existence of the user interaction event handler. If not, the module marks the click request as fraudulent behavior.

3) Management Module: The management module employed with a web administrator page is designed to manage and monitor the whole system. Users could upload and process datasets, modify the hybrid model, view the hybrid model's train process, and so on by this module. In addition, the management module conducts the producer-consumer architecture

```
hookhelper.hookMethod(executeRequest, new AbstractBahaviorHookCallBack() {
        @Override
        public void descParam(HookParam param) {
            Logger.log_behavior("Apache Connect to URL ->");
            HttpHost host = (HttpHost) param.args[0];

            HttpRequest request = (HttpRequest) param.args[1];
            if (request instanceof org.apache.http.client.methods.HttpGet) {
                org.apache.http.client.methods.HttpGet httpGet =
(org.apache.http.client.methods.HttpGet) request;
                Logger.log_behavior("HTTP Method : " + httpGet.getMethod());
                Logger.log_behavior("HTTP GET URL : " + httpGet.getURI().toString());
                Header[] headers = request.getAllHeaders();
                if (headers != null) {
                    for (int i = 0; i < headers.length; i++) {
                        Logger.log_behavior(headers[i].getName() + ":" +
headers[i].getName());
                    }
                }
            }
        }
}
```

Fig. 10. An example of hook operation.

in the dynamic testing module. The testing task is distributed by users using the management module to different devices via an internal message queue framework. The management module supports users in analyzing apps and filtering invalid traffic in parallel. It also supports conducting several testing tasks parallelly in the dynamic testing module by controlling heterogeneous Android devices.

IV. Evaluation

In this section, we evaluate the overall performance of CFDMA. We conducted experiments on two machines running 64-bit Ubuntu 20.04 LTS with Intel Xeon CPUs, 128GB of main memory, and NVIDIA TITAN RTX 24G. One host implements the management module that distributes analysis tasks and trains the hybrid model in the traffic detection module. The other one monitor devices used in the dynamic testing module. Virtual devices are Pixel 2 with Android version 4.4 based on Android Virtual Device (AVD). The physical device is realme Q2 pro with Android version 4.4.

A. Experimental Setup

1) Click Dataset: We conducted experiments on the following three datasets to evaluate our detection method.

1) TalkingData dataset[1]. This dataset contains records of 200 million clicks with eight features over four days.
2) Avazu dataset[2]. This dataset consists approximately 40 million click records with 24 features over ten days.
3) Kad dataset[3], which contains 1000 click records with ten features.

2) Crawled Android apps: There is still a lack of benchmarks for mobile click fraud detection, so we first collected 20 Android apps with confirmed click fraud behaviours from the Internet. We combine these apps with 20 known benign apps as the benchmark dataset. Then, we collected 500 apps from HUAWEI App Store and Google Play Store from February 2022 to March 2022 as our test dataset. We randomly sample apps from top-rated free apps, including all categories. These apps account for the vast majority of app downloads. Thus, we can focus on apps that have the greatest influence.

3) Emulate Real Scenario: CFDMA manages each device through an ADB connection and uses Android UI Automator to simulate user interactions on devices. The testing actions as follows are performed randomly on each device.

1) Turn on the screen, wait for random seconds from 1 to 5, and then unlock the mobile.
2) Press the home button, wait for random seconds from 1 to 5, and then go back to the application
3) While using the app, press the volume down/up button several times.
4) Open the notification bar and close it in 3 seconds.

[1]https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection
[2]https://www.kaggle.com/c/avazu-ctr-prediction/data
[3]https://www.kaggle.com/datasets/tbyrnes/advertising

TABLE I
Comparison based on TalkinData dataset

Model	AUC	Precision	Recall	F1-score
ETCF	89.71	84.02	83.50	82.24
HYBRID DL	95.45	95.80	95.80	95.67
Wide&Deep1	95.43	95.21	95.32	95.43
Wide&Deep2	**96.82**	**96.00**	**96.01**	**96.32**

TABLE II
Comparison based on Avazu dataset

Model	AUC	Precision	Recall	F1-score
ETCF	75.50	69.61	69.68	69.83
HYBRID DL	86.11	82.80	82.77	81.70
Wide&Deep1	85.03	83.21	82.10	81.03
Wide&Deep2	**87.57**	**87.25**	**84.64**	**82.35**

B. Experimental results

1) Performance of Traffic Detection Module: Based on the above three click fraud datasets, we evaluate the performance of the traffic detection module and compare it with ETCF [25] and HYBRID DL [26], which are state-of-art click fraud detection researches. In order to evaluate the impact of graph embedding features, we divide the traffic detection module into two types: (1) detection not using graph embedding (Wide&Deep1) and (2) detection using graph embedding (Wide&Deep2).

As shown in Table I-III, compared with other methods, our traffic detection method (Wide&Deep2) has a significant improvement in AUC, Precision, Recall and F1-score on almost all datasets, except that the Precision of HYBRID DL on Kad dataset is slightly higher than that of Wide&Deep2. In addition, considering that the Kad dataset only contains 1000 records, it also shows that the traffic detection module can be applied to datasets with a small number of samples. By comparing Wide&Deep1 and Wide&Deep2, it can be observed that graph embedding features can help improve the detection effect. It also shows that when constructing graph structures or using graph embedding in different scenarios, it is necessary to consider the particularity of the scenarios and optimize the construction or embedding process to obtain better effects.

2) Performance of Dynamic Testing Module: We first use the benchmark dataset to test the effect of the dynamic testing module proposed above. As shown in Table IV, we identify 18 true positive fraudulent apps, two false positives and one false negative from the benchmark dataset.

Then, we conduct fraud detection on 500 collected apps and identify 16 fraudulent apps. By manually checking the decompiled codes of these apps, we confirm that 14 of the 16

TABLE III
Comparison based on Kad dataset

Model	AUC	Precision	Recall	F1-score
ETCF	74.81	75.90	75.91	68.80
HYBRID DL	93.43	**92.41**	91.37	91.12
Wide&Deep1	92.43	90.21	90.77	90.20
Wide&Deep2	**93.87**	**92.00**	**92.24**	**91.70**

TABLE IV
EVALUATION ON THE BENCHMARK DATASET

Precision	Recall	F1-score
90.00	94.70	92.30

apps conduct click fraud. Table V shows the details of mobile apps that commit click fraud.

TABLE V
DETAILS OF APPS COMMITTING CLICK FRAUD.

Number of apps	Type of fraudulent behavior
1	Fabricated clicks by third-party libraries
1	Fabricated click requests by local codes
12	Fabricated click requests by third-party libraries

We observe that third-party SDK injection plays an important role in mobile click fraud. Most of the fraudulent apps commit click fraud by directly sending forged click requests to ad networks based on third-party libraries rather than injecting fraudulent codes into the app's native codes. Attacks based on third-party libraries are the dominant form of click fraud in our dataset. Table VI shows third-party SDKs that commit click fraud in apps and the number of apps embedding them. We then analyze how the third-party libraries performed click fraud by checking decompiled codes. com.moxxx obtains the click URL by accessing a specific URL and parsing the return JSON data, and then instantiates WebView object to generate click requests without user interaction. com.ba*** and com.an*** commit fraud by creating an off-screen ad WebView instance that loads click URLs to send click requests to various ad servers. com.xm*** uses java.net.HttpURLConnection to generate ad click requests. com.dz*** uses the loadURL method to generate and send click requests. com.ku*** commit click fraud by randomly generating horizontal and vertical coordinate values to forge users' click behavior.

TABLE VI
DETAILS OF THIRD-PARTY LIBRARIES THAT COMMIT CLICK FRAUD.

Library name	Number of apps	Type of fraudulent behavior
com.mo***	4	Fabricated click requests
com.ba***	3	Fabricated click requests
com.an***	2	Fabricated click requests
com.xm***	2	Fabricated click requests
com.dz***	1	Fabricated click requests
com.ku***	1	Fabricated clicks

We compare our dynamic testing module with MAdFraud [16], which also detects click fraud on the client side. We conduct MAdFraud on 500 collected apps. MAdFraud only detects two fraudulent apps, far less than the 14 fraudulent apps we detect. Since there is no user interaction with apps when detecting, MAdFraud cannot further detect click fraud when apps need to click the user's permission, the user's consent window, or when the open-screen advertisement needs to be closed manually. The dynamic testing module uses UI Automator to parse the UI information of the current screen so that more views and functions can be explored.

V. CONCLUSIONS

In this paper, we designed and implemented CFDMA for automatically uncovering click fraud on both server side and client side. By applying CFDMA to three real-world click datasets, we proved that our method achieves higher detection accuracy than state-of-art. We also conducted CFDMA on apps from the real world to evaluate our detection method on the client side. The results show that our method can detect fraudulent apps well on the client side.

REFERENCES

[1] Statista, "Mobile advertising spending worldwide from 2007 to 2024," https://www.statista.com/statistics/303817/mobile-internet-advertising-revenue-worldwide/, accessed 22 Jan 2021.

[2] IAB, "Iab internet advertising revenue report," https://www.iab.com/wp-content/uploads/2019/05/Full-Year-2018-IAB-Internet-Advertising-Revenue-Report.pdf, accessed 22 Jan 2021.

[3] Sproutsocial, "Cpc (cost per click)," https://sproutsocial.com/glossary/cost-per-click/, accessed 18 Jan 2021.

[4] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *The Third International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2009, 18-23 June 2009, Athens/Glyfada, Greece*, R. Falk, W. Goudalo, E. Y. Chen, R. Savola, and M. Popescu, Eds. IEEE Computer Society, 2009, pp. 268–273. [Online]. Available: https://doi.org/10.1109/SECURWARE.2009.48

[5] M. Zhang, W. Meng, S. Lee, B. Lee, and X. Xing, "All your clicks belong to me: Investigating click interception on the web," in *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-[T1]16, 2019*, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 941–957. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/zhang

[6] Statista, "Economic losses due to digital ad fraud in selected countries worldwide in 2020," https://www.statista.com/statistics/1117175/digital-ad-fraud-economic-loss-world/, accessed 2 Feb 2021.

[7] B. Behdani, M. Allahbakhshi, A. Asheralieva, and M. Gheisari, "Analytical method for ferroresonance solutions in series compensated power systems due to gics: A graphical approach," *2021 IEEE Madrid PowerTech*, pp. 1–6, 2021.

[8] Xposed, "Xposed module repository," https://repo.xposed.info/, accessed 2 Feb 2021.

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[10] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide amp; deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, ser. DLRS 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 7–10. [Online]. Available: https://doi.org/10.1145/2988450.2988454

[11] H. Haddadi, "Fighting online click-fraud using bluff ads," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, p. 21–25, apr 2010. [Online]. Available: https://doi.org/10.1145/1764873.1764877

[12] K. A. Raza, A. Asheralieva, M. M. Karim, K. Sharif, M. Gheisari, and S. Khan, "A novel forwarding and caching scheme for information-centric software-defined networks," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021, pp. 1–8.

[13] V. Dave, S. Guha, and Y. Zhang, "Measuring and fingerprinting click-spam in ad networks," in *ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012*, L. Eggert, J. Ott, V. N. Padmanabhan, and G. Varghese, Eds. ACM, 2012, pp. 175–186. [Online]. Available: https://doi.org/10.1145/2342356.2342394

[14] X. Pan, X. Wang, Y. Duan, X. Wang, and H. Yin, "Dark hazard: Learning-based, large-scale discovery of hidden sensitive operations in android apps," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[15] P. Pearce, V. Dave, C. Grier, K. Levchenko, S. Guha, D. McCoy, V. Paxson, S. Savage, and G. M. Voelker, "Characterizing large-scale click fraud in zeroaccess," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, G. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 141–152. [Online]. Available: https://doi.org/10.1145/2660267.2660369

[16] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 123–134. [Online]. Available: https://doi.org/10.1145/2594368.2594391

[17] T. Zhu, Y. Meng, H. Hu, X. Zhang, M. Xue, and H. Zhu, "Dissecting click fraud autonomy in the wild," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 271–286. [Online]. Available: https://doi.org/10.1145/3460120.3484546

[18] S. Nagaraja and R. Shah, "Clicktok: Click fraud detection using traffic analysis," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 105–116. [Online]. Available: https://doi.org/10.1145/3317549.3323407

[19] M. Mangla, S. Deokar, R. Akhare, and M. Gheisari, "A proposed framework for autonomic resource management in cloud computing environment," in *Autonomic Computing in Cloud Resource Management in Industry 4.0*. Springer, 2021, pp. 177–193.

[20] G. S. Thejas, K. G. Boroojeni, K. Chandna, I. Bhatia, S. S. Iyengar, and N. R. Sunitha, "Deep learning-based model to fight against ad click fraud," in *Proceedings of the 2019 ACM Southeast Conference*, ser. ACM SE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 176–181. [Online]. Available: https://doi.org/10.1145/3299815.3314453

[21] G. S. Thejas, J. Soni, K. G. Boroojeni, S. Iyengar, K. Srivastava, P. Badrinath, N. Sunitha, N. Prabakar, and H. Upadhyay, "A multi-time-scale time series analysis for click fraud forecasting using binary labeled imbalanced dataset," in *2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*, vol. 4, 2019, pp. 1–8.

[22] Android Documentation, "Input events overview," https://developer.android.com/guide/topics/ui/ui-events.html, accessed 2 Feb 2022.

[23] Android Documentation, "MotionEvent," https://developer.android.com/reference/android/view/MotionEvent.html, accessed 2 Feb 2022.

[24] C. Cao, Y. Gao, Y. Luo, M. Xia, W. Dong, C. Chen, and X. Liu, "Adsherlock: Efficient and deployable click fraud detection for mobile applications," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1285–1297, 2021.

[25] X. Qiu, Y. Zuo, and G. Liu, "Etcf: An ensemble model for ctr prediction," in *2018 15th International Conference on Service Systems and Service Management (ICSSSM)*, 2018, pp. 1–5.

[26] T. G.S., S. Dheeshjith, S. Iyengar, N. Sunitha, and P. Badrinath, "A hybrid and effective learning approach for click fraud detection," *Machine Learning with Applications*, vol. 3, p. 100016, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827020300165