Classification of Touch Spam in Mobile Ad Networks using Bi-Partite Graph

M. Sree Vani
Dept of CSE, MGIT , Hyderabad-500075

Abstract

A touch user interface (TUI) is a computer-pointing technology based upon the sense of touch (haptics). Touch-spam is a type of fraud that occurs over TUI gadgets ex. Smartphones, tablets, phablets, touch laptops etc. It actually happens in TUI applications when a person, automated script, computer program or robotic action imitates a legitimate user of a TUI application touching on an advertisement (ad), for the purpose of generating a charge per touch without having actual interest in the target of the ad's popup. Touch-spam is becoming an issue due to the advertising networks being a key beneficiary of this spam. In present days, smartphone gaming applications (apps) are playing a vital role to attract mobile-advertisements (ads) since their pocket portability and other versatile features. Popular apps are able to read the user personalized data to process user interests helping to generate customized ads. Touch-spam in smart phone apps is a fraudulent or invalid tap or touch on online ads, where the user has no actual interest in the advertiser's site. It requires a user touch on online ads that pop-up dynamically in smartphone gaming apps. It all need the user to tap the screen close to where the ad is displayed .While the ad networks continue taking active measures to block click-spam today, the touch-spam still creeping under the TUI. It is being used by spammers to misappropriate the advertising revenue. The presence of touch-spam is largely unknownThen we propose a node-tag propagation algorithm on click-through logs to identify spam Apps in Smartphone-game Apps. We validate our methodology using click/touch-log data from major ad network. Our findings highlight the extremity of the spam in mobile advertising.

Keywords : *spam, mobile apps, click spam, bi-partite graph.*

I. INTRODUCTION

By vast usage of mobiles, a specialized third-party applications ("apps") are playing big role for growing of Smartphone and tablet markets in leaps and bounds. Whether on the iPhone or

Android platforms, apps often come in two flavors: a free version, with embedded advertising, and a pay version without. Both models have been successful in the marketplace. Mobile advertisements within the apps are only source of revenue for several mobile app publishers. Maximum of the apps in the major mobile app stores show ads. To embed ads in an app, the app developer typically registers with a third-party mobile ad network such as AdMob [2], iAd [3], Microsoft Mobile Advertising [4] etc. The ad net- works supply the developer with an ad control (i.e. library with some visual elements embedded within). The developer includes this ad control in his app, and assigns it some screen real estate. When the app runs, the ad control is loaded, and it fetches ads from the ad network and displays it to the user. Different ad networks use different signals to serve relevant ads. One of the main signals that mobile ad networks use today is the app metadata . As part of the registration process, most ad networks ask the developer to provide metadata information about the app (for e.g. category of the app, link to the app store description etc.). This allows the ad network to serve ads related to the app metadata. Ad networks also receive dynamic signals sent by the ad control every time it fetches a new ad. Depending on the privacy policies and the security architecture of the platform, these signals can include the location, user identity, etc. Note that unlike JavaScript embedded in the browsers, the ad controls are integral parts of the application, and have access to the all the APIs provided by the platform.

A. Background on mobile advertising

A typical mobile advertising system has five participants: mobile clients, advertisers, ad servers, ad exchanges and ad networks as Figure 1 shows. A mobile application includes an ad control module (e.g., AdControl for Windows Phones, AdMob for Android) which notifies the associated ad server any time an ad slot becomes available on the client's device. The ad server decides how to monetize the ad slot by displaying an ad. Ads are collected from an ad exchange. Ad exchanges are neutral parties that aggregate ads from different third party ad networks and hold an auction every time a client's ad slot becomes available. The ad networks participating in the exchange estimate their expected revenue from showing an ad in such an ad slot and place a bid on behalf of their customers (i.e., the advertisers). An ad network attempts to maximize its revenue by choosing ads that are most appropriate given the context of the user, in order to maximize the possibility of the user clicking on the ads. The ad

network receives information about the user such as his profile, context, and device type from the ad server, through the ad exchange. Ad exchange runs the auction and chooses the winner with the highest bid.

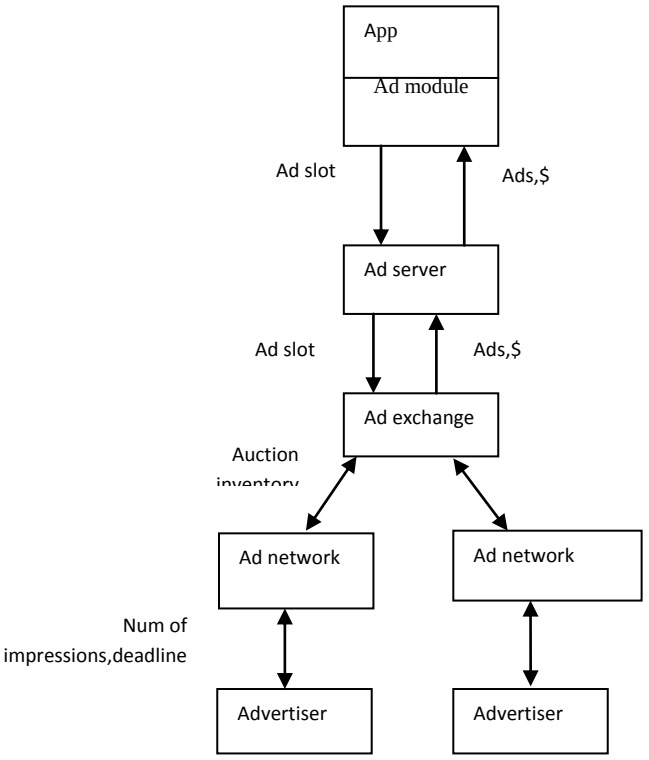


Figure 1: Architecture of a typical mobile ad system.

Advertisers register with their ad networks by submitting an ad campaign. A campaign typically specifies an advertising budget and a target number of impressions/clicks within a certain deadline (e.g., 50,000 impressions delivered in 2 weeks).

They can also specify a maximum cap on how many times a single client can see a specific ad and how to distribute ads over time (e.g., 150 impressions per hour). The ad server is responsible for tracking which ads are displayed and clicked, and thus determining how much money an advertiser owes. The revenue of an ad slot can be measured in several ways, most often by views (Cost Per Impression) or click-through (Cost Per Click), the former being most common in mobile systems. The ad server receives a premium on the sale of each ad slot, part of which is passed to the developer of the app where the ad was displayed.

The rest of this paper is ordered as follows. In Section 2, we review approaches for click spam detection in previous work. In Section 3, we introduce our methodology for prediction of spam in Mobile Apps. Section 4 presents our novel Graph based label propagation algorithm. Section 5 describes our experiment setup and shows experimental results. Finally, In section 6, we presented our conclusions and future work.

II. RELATED WORK

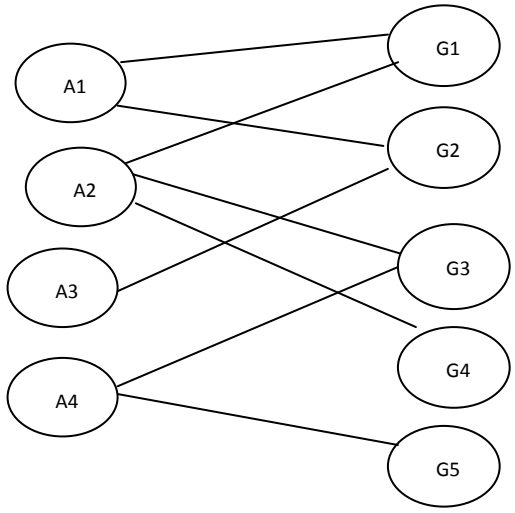
Existing works on ad fraud mainly focus on the click-spam behaviors, characterizing the features of click-spam, either targeting specific attacks [5, 6, 16, 18], or taking a broader view [7]. Some work has examined other elements of the click-spam ecosystem: the quality of purchased traffic [19, 20], and the spam profit model [12, 13]. Very little work exists in exploring clickspam in mobile apps. From the controlled experiment, authors in [7] observed that around one third of the mobile ad clicks may constitute click-spam. A contemporaneous paper [9] claimed that mobile malware is nowhere in the wild which performs spam in advertising. DECAF focuses on detecting violations to ad network terms and conditions, and even before potentially fraudulent clicks have been generated. With regard to detection, most existing works focus on bot-driven click spam, either by analyzing search engine query logs to identify outliers in query distributions characterizing networking traffic to infer coalitions made by a group of bot-driven fraudsters [14, 15], or authenticating normal user clicks to filter out bot-driven clicks [10, 11]. A recent work, Viceroi [8], designed a more general framework that is possible to detect not only bot-driven spam, but also some non-bot driven ones (like search-hijacking).To the best of our knowledge, ours is the start-up work to detect touch spam in mobile apps.

III. METHODOLOGY

By observing the above facts, we propose a node-tag propagation algorithm on click-through logs. Initially, a few game-apps are selected as seed set and tagged as spam or non-spam. Then their tags are propagated on the bipartite graph and other possible spam/non-spam game-apps are identified. Our input data set contains three items. The first item is a set of tagged game-apps (spam or non-spam), second item is a set of untagged game-apps and last item is a set of constraints between game-apps and the ad-controls in the log. The goal is to find spam game-apps which are misplaced ad-controls from the unlabeled data.

A. *Advertiser web server log contains click-through data C and bipartite graph G.*

The click data consists of triples <a, g, f_{ag}>, where a is a ad-control clicked by gamer or player, g is publisher website which serves an ad to user by fetching from ad-network and f_{ag} is the number of times that ad-control ad is clicked when user is playing game g. Define AD = {a | a appears in C} and GA = {g | g appears in C}.



Ad-controls **2 :Bipartite Grap** Game-Apps

Click-through log C has an equal representation– a click-through bipartite graph G = (AD, GA, E). There are two different types of nodes, ad-controls and GAs in G. For every record <a, g, fag> in C, there is an edge (a, g) ∈ E with weight fag. Each a/g is assigned with a probability pa/pg which denotes how likely this a/g is to be a spam game app which is misplaced ad-control in game or in other words, the spamicity of a/g. The construction of click-through bipartite graph can be on page-level or site-level. In the site-level, g is replaced by its site but not the URL of itself. For example, <"Nokia", http://product.pcpop.com/Mobile /00283_1.html, 100> is replaced by <"Nokia", http://product.pcpop.com/, 100>.

B. Tagged Seed GA set T.

T contains all of the game-apps in C (G) that are manually tagged as spam or non-spam. Formally, T = {g | g is tagged as a spam game-app or non-spam game-app}. We will elaborate the construction details of T in Section 5.2.

C. GA Result Set GU

Set GU contain all the <g, pg> and <a, pa> tuples, respectively. After our algorithm ends, each GA g or ad-control a in C (or G) will be assigned with a probability pg/ pa, which denotes the probability that this GA or ad-control is a spam game-app which are misplaced ad-control. More formally, GU = {<g, pg > | pg is the spam score for g-app}. Given G = (AD, GA, E) and T∈ GA, the aim of the spam game apps mining problem is to get the result set GU , which contain all of the possible spam game apps which are misplaced ad-control in G.

IV. NODE-TAG PREDICTION ALGORITHM

A. *Design of Algorithm*

In our work, we propose a node-tagl propagation (N-TP) algorithm to get solution for the problem that is defined in the above section. More pointedly, for each game-app g, we could calculate the probability pg that g-app is a spam game-app by incorporating all of the node information of its neighbors. We elaborate this procedure more formally as follows.

For a/g, we use la/lg to denote its tag, which is S for spam and N for non-spam. Note that P(lg=N) = 1-P(lg=S). Thus every GA g in tagged set T would have P(lg=S)=1 or P(lg=S)=0 initially and every GA g in the set GA- T would have P(lg=S)=0.Then we have

$$p(l_g = S) = \sum_{a:(a,g)\in E} \omega_{ga} P(l_a = S) \quad (1)$$

Where

$$\omega_{ga} = \frac{f_{ag}}{\sum_{a:(a,g)\in E} f_{ag}} \quad (2)$$

is the transition probability from GA g to ad-control a.

Similarly, for each ad-control AD a in GA\T, the probability $P(l_a = S)$ is computed as

$$p(l_a = S) = \sum_{g:(a,g)\in E} \omega_{ag} P(l_g = S) \quad (3)$$

Where

$$\omega_{ag} = \frac{f_{ag}}{\sum_{g:(a,g)\in E} f_{ag}} \quad (4)$$

is the transition probability from ad-control a to GA g. Note that both ω_{ag} and ω_{ga} are not limited to the above form but arbitrary. The only requirement for them is they should have a probability interpretation, which means $\sum_{a:(a,g)\in E} \omega_{ag} = 1$ and $\sum_{a:(a,g)\in E} \omega_{ga} = 1$. Using Equation (1) and (3), we can get P(lg=S) and P(la=S) recursively for all of the GAs in the bipartite graph. We can have a brief representation of this iterative process. Suppose that there are |AD| ad-controls: a1, a2…a|AD| and |GA| Game-apps: g1, g2,…g|GA|. Define vectors: PAD=(P(la1=S), P(la2=S)…P(l a|AD|=S))T, PGA=(P(lg1=S), P(lg2=S)…P(lg|GA| =S))T, and the transition probability matrixes: Mag= (ω_{ag}

) |AD|x|GA|, and Mga= (ω_{ga}) |GA|x|AD .Then in the ith iteration, we have PiAD=Mag Pi-1 GA , Pi GA= MgaPi AD

We notice that in every round of iteration, all of the GAs in seed set T should be re-assigned their initial tags. In this way, the algorithm converges. We will demonstrate the convergence in section 4.4. The outline of the Graph based Node-tag Propagation algorithm is shown in Figure 3.

Input：tagged seed set T, click-through log data C(G)

Output：P(lg=S) and P(la=S) for all GAs and ad-controls in G

Begin
Do

for (g $\in$ L, set P(lg=S)=1 or 0 according to their tags by human assertors.)

for (all a $\in$ AD) do

$$p(l_a = S) = \sum_{g:(a,g)\in E} \omega_{ag} P(l_g = S)$$

end for

for (all g $\in$ GA\T)do

$$p(l_g = S) = \sum_{a:(a,g)\in E} \omega_{ga} P(l_a = S)$$

end for

until convergence

Output P(lg=S) for every GameApp g in GA and P(la=S) for every ad-control a in AD

End

Figure 3:The Graph based node-tag propagation algorithm

B. *N-TP Algorithm Convergence*

As we know that $\mathbf{M_{ag}}$ and $\mathbf{M_{ga}}$ are right stochastic matrixes means each of whose rows consists of positive real numbers, with each row summing to 1. Then consider $\mathbf{M_{gg}}=\mathbf{M_{ga}}\mathbf{M_{ag}}$. For each element m_{ij} in $\mathbf{M_{gg}}$, we have $m_{ij} = \sum_k \omega_{ik}\omega'_{kj}$ in $\mathbf{M_{gg}}$, where ω_{ik} and ω'_{kj} are elements $\mathbf{M_{ga}}$ and $\mathbf{M_{ag}}$, respectively. Thus we have

$$\sum_j m_{ij} = \sum_j \sum_k \omega_{ik}\omega'_{kj}$$
$$= \sum_j \sum_k \omega_{ik}\omega'_{kj}$$
$$= \sum_k \omega_{ik} \sum_j \omega'_{kj}$$
$$= \sum_k \omega_{ik}$$
$$= 1$$

That means Mgg is also a right stochastic matrix. Now, we are only interested in PGA, the iteration process can be rewritten as PiGA= MggPi-1GA= MgaMag P i-1GA, where i denotes the iterations. Suppose that there are |T| seed GAs in T, |C| 1-degree nodes and thus r = |GA|-|T|-|C| remaining GAs in C. More pointedly, let the probability vector PGA= (PT PL) where PT are the top |T|+|C| rows of PGA(the labeled data and the pseudo labeled data) and PT are the remaining r rows of PGA(the unlabeled data). We split Mgg after the (|T|+|C|)th row and the (|T|+|C|)th column into 4sub-matrixes

$$M_{gg} = \begin{bmatrix} M_{(|T|+|C|)(|T|+|C|)} & M_{(|T|+|C|)r} \\ M_{r(|T|+|C|)} & M_{rr} \end{bmatrix}$$

.Note that PT never really changes. It can be shown that in our algorithm,

$\mathbf{P_T} = \boldsymbol{M_{rr}}\mathbf{P_T} + M_{r(|T|+|C|)}\,\mathbf{P_T}$ which lead to

$$\mathbf{P_T} = \lim_{n\to\infty} (\boldsymbol{M_{rr}})^n \mathbf{P_T^0} + \left[\sum_{i=1}^{n} \boldsymbol{M_{rr}^{i-1}}\right] M_{r(|T|+|C|)}\,\mathbf{P_T}$$

Zhu and Ghahramani [27] proved that PL converges to (I-Mrr)-1 $M_{r(|T|+|C|)}\,\mathbf{P_T}$.□ if Mgg is a right stochastic matrix. Thus the initial value of PT is inconsequential. Using the same approach, we could prove that PAD also converges.

V. EXPERIMENTS

The aim of our experiments is to examine how effective our algorithm is in predicting spam Game Apps. Given a seed set T, the N-TP algorithm returns a list of game apps that are arranged according to their probability of being spam. Seed game apps are not included in the list. We also have a list of ad-controls that are arranged according to their probability of being used as a spam-oriented ad-control.

A. *Datasets*

We collected our data sets from few top most ad networks by sign-up as an advertiser. We gathered all web requests made to our server and formed a log. The logs used in this study are standard Apache web server logs that contains the date and time of access ,user's IP address, , URL accessed (of a page on our webserver) along with any GET parameters, User-Agent value sent for that request,the HTTP Referer value and a cookie value

we set the first time we see a user to identify repeat visits from the same user. Our datasets also consist of all selected game apps crawled from the Apple iOS App Store in 2012. We collected 13,267 top free game apps from App Store.

B. Bipartite Graph Construction

We pruned the entire ad -game apps tuples with just one click/touch on any day in the log because they may contain possible privacy information and noise. Then, this click-through log consisted of 24,435 unique ad-controls, with 34,708 unique game apps in 1,055 sites. To construct bipartite graph , we used Altogether, 50,660 ad-game apps tuples. The maximal component of the graph contains 25.0% unique ad-controls, 29.0% game apps and 44.2% ad-game apps pairs.

From our datasets, we computed metadata for apps, developers, and users who post reviews .We divide the whole data sets into two parts as training dataset and test dataset. We then went for manual labeling of training datasets. To label apps as spam or non spam, we invited some volunteers who had experience with mobile game playing to participate. After labeling process, we have 81 spam posts (17%) and 401 non spam posts (83 %.).

C. Results

We would like to detect as many spam apps as possible while avoiding misclassifying non-spam ones. We conduct various experiments with our dataset using our Label propagation algorithm. We compared our algorithm against with pagerank and Trust algorithm. The experimental results shown in the Figure 4.

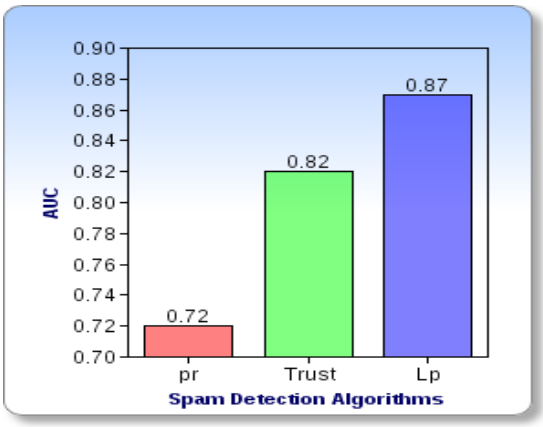


Figure 4 :AUC Values for Different Spam Detection Algorithms.

From the above Figures, we can observe that AUC values of the five algorithms are greater than 0.78, which suggests that they are effective in detecting Web spam. page rank performs the worst because spam sites can boost their page rank scores using tricks such as the link-farm. TRUST works better than PR, which is consistent with previous research (10). The AUC of N-TP is 0.870, which is much better than both PR. And TRUST. This illustrates our algorithm gives much better performance in detecting web spam sites. To evaluate our algorithm, we conducted experiment on seed selection .We randomly split our spam sites into 21 subsets (each with 100 seed sites and then add them gradually into seed set. The experiment results are summarized in Figure 5. It can be seen that all of our algorithms are very good performers. They can achieve a relatively high AUC value after only 400 sites are added into the seed sets. We also notice that N-TP performs consistently better than PR and TRUST.

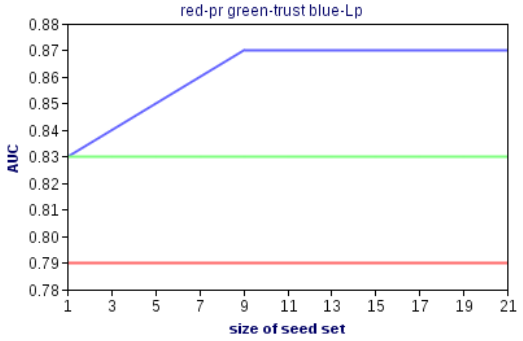


Figure 5 :Algorithm Performance with Different Seed Sets

VI. CONCLUSION

In our work, we have proposed a Graph based Node-Tagged propagation algorithm to predict spam in Mobile game apps. This Algorithm constructed bipartite graph G from Advertiser web server log data and generates Labeled Seed set T. Then finally extracts the resultant set GU, which contains all of the possible spam game apps which are misplaced ad-control in G. Experiment results indicates that our algorithm is effective and efficient for predicting spam in mobile game Apps. In future, we try to combine our algorithm with some current anti-spam techniques results in a much better performance.

REFERENCE

[1] M.Najork. Web spam detection. In L. Liu and M. T. □Ozsu, editors, Encyclopedia of Database Systems, pages 3520{3523. Springer US, 2009.
[2] Googleadmob. ttp://www.google.com/ads/admob/.
[3] iadappnetwork. http://developer.apple.com/support/appstore/iad-app-network/.
[4] Microsoft advertising. http://advertising.microsoft.com/en-us/splitter.
[5] S.Alrwais, A. Gerber, O. Spatscheck,M. Gupta, and E. Osterweil. Dissecting ghost clicks: Ad fraud via misdirected human clicks. ACSAC, 2012.
[6] T.Blizard and N. Livic. Click-fraud monetizing malware: A survey and case study. 2012.
[7] P.Chia, Y. Yamamoto, and N. Asokan. Is this app safe? a large scale study on application permissions and risk signals. In WWW, 2012.

[8] V.Dave, S. Guha, and Y. Zhang. Measuring and fingerprinting click-spam in ad networks. In ACM SIGCOMM, 2012.
[9] C.Cadar D. Dunbar and D. Engler. Klee: In USENIX OSDI, 2008.
[10] P.Gilbert, B. Chun, J. Jung. Vision:automated security validation of mobile apps at app markets. In MCS, 2011.
[11] H.Haddadi. Fighting online click-fraud using bluff ads. ACM Computer Communication Review, 40(2):21–25, 2010.14
[12] C.Hu and I. Neamtiu. Automating gui testing for android applications. In AST, 2011.
[13] A.MacHiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for android apps. In FSE, 2013.
[14] A.Mesbah and A. van Deursen. Invariant-based automatic testing of ajax user interfaces. In ICSE, 2009.
[15] Ali Mesbah, Arie van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web, 6(1):1–30, 2012.
[16] A.Metwally, D. Agrawal, and A. El Abbadi. Detectives:Detecting coalition hit inflation attacks in advertising networks streams. In WWW, 2007.
[17] A.Metwally, F. Emekci, D. Agrawal, and A. El Abbadi.Sleuth: Single-publisher attack detectIon using correlation hunting. In PVLDB, 2008.
[18] B.Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson. What's clicking what? techniques and innovations of today's clickbots. In DIMVA, 2011.
[19] L.Ravindranath, J. Padhye, S. Agarwal, R. Mahajan,I. Appinsight: mobileapp performance monitoring inthe wild. In USENIX OSDI, 2012.
[20] W.Yang, M. Prasad, and T. Xie. A grey-box approach for automated gui-model generation of mobile applications. In FASE, 2013.