Highly Accurate Marketing Source Attribution for Android Out-of-Store Apps

Kaushik Wavhal, Abhishek Bharti, Divyesh Topiya, Vivek Shah, Monish Shah, Jignesh Karelia
Play Games24x7 Private Limited,
Mumbai - 400 064, India
(kaushik.wavhal, abhishek.manit, divyesh.topiya, phantomvivek, monish85, jhkarelia)@gmail.com

Abstract

Tracking installation sources of an app is vital in any mobile app's online advertising. It helps to identify the efficiency of marketing campaigns at driving app installations and thus optimize the marketing spends to generate a higher return on investment (ROI). However, in case of Android apps, the Android operating system only has provisions to track the installation source of apps which are hosted on Google's Play Store. Hence, it is extremely difficult for out-of-store android apps to accurately track its installation sources.

This paper presents a novel method to achieve hundred-percent accuracy in tracking the installation source of out-of-store android apps. In addition, the method involves a technique to effectively scale and deploy this feature on production environment. Based on the evaluation of the performance of installation source attribution, this technique outshines the accuracy of previously used techniques by a large margin.

keywords: Android, Attribution, Install, Referrer

1 Introduction

As mobile internet usage is growing day by day, more marketers are relying on mobile advertisements for promoting mobile apps. These advertisements typically involves advertising through multiple online marketing sources. For mobile advertisements promoting app installations, the number of app installs that a particular advertisement generates is a crucial metric. For quantifying these app installs, it is important to attribute every app installation to its appropriate source. Google's referral tracking system enables marketers identify the marketing sources that drive visitors to Google Play Store and download the app [8]. However, this technique does not work for apps that are not hosted on Google Play Store.

Return on Investment (ROI) of app-install campaigns indicates the effectiveness of that campaign at driving installations. The key is to utilize this attribution information from advertising campaigns to optimize them for maximum user engagement with smart targeting. The marketing spends can be adjusted accordingly based on the ROI of campaigns. The marketing budget for a campaign with a lower ROI can be reduced while increasing the budget for the campaigns with higher ROI. This can help companies save a few million dollars from their marketing budget by fine tuning the spends across marketing campaigns.

Motivation: Google Play Store is banned in China[2][3]. Moreover, more than 50 countries across the globe does not have access to paid apps and apps with in-app products on Google Play Store[4]. Fantasy sports, betting and gambling apps are banned from Google Play Store while being legalized in several parts of the world. Apps falling in the above categories have to rely on third-party app stores for its distribution. Clearly, there is a pressing need for a solution to enable highly accurate Install Source attribution for such apps.

Accurate tracking of the installation source for Android apps is hard because web pages run in browsers which maintain a sand-boxed environment relative to the Android system. Hence, it is difficult to identify the marketing source driving the app's installation from within the app. In Google's referral tracking system, Google Play Store has made a provision to pass an identifier to the Play Store via it's URL [8]. This identifier gets passed on to the app after installation in the form of a broadcast event from Google's Cloud Messaging Service. An installation source identifier can thus be passed to the apps installed via Google Play Store. This identifier is linked with the origin URL on the server. Thus the app can fetch the origin URL of the marketing campaign by sending the identifier to the server. Since, the referral tracking system works in close conjunction with Google Play Store, this method cannot be used for out-of-store apps.

Device Fingerprinting is a widely used technique for the purpose of installation source attribution of out-of-store apps. In this technique, user data is collected at

978-1-943436-09-5 / copyright ISCA, SEDE 2017
October 2-4, 2017, San Diego, California, USA

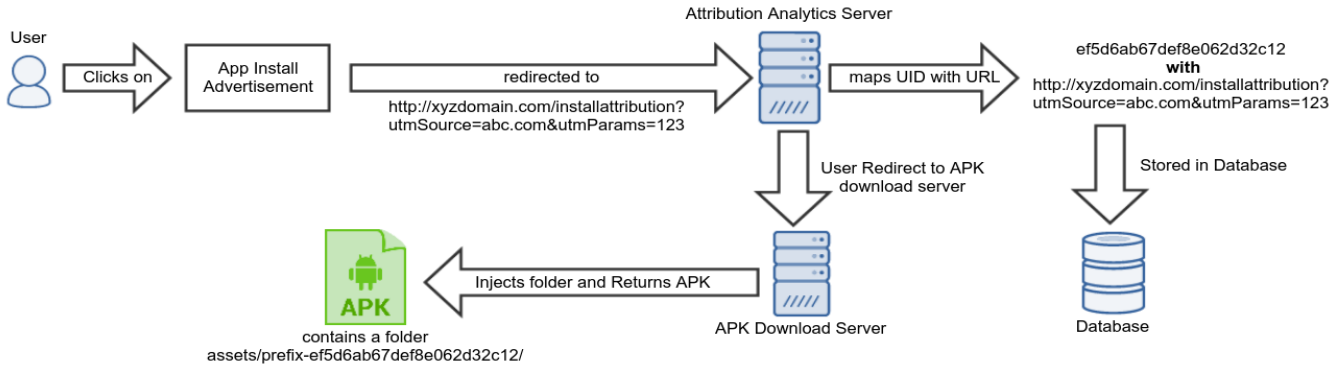

Figure 1: Overview of the proposed solution.

two events. Once when the download is initiated from the server and once when the app starts for the first time. When the user initiates the app download, the values of few device parameters like IP address, screen size, operating system version, device model number and more are collected and stored on the server. When the app runs for the first time, this same parameters are collected and sent to the server for matching. If a match is found for a download event, the app is assumed to have originated from the source tied with the matched fingerprint combination. Various attribution analytics platforms use this technique for installation source attribution. The accuracy of this method is highly probabilistic as it assumes that no two users will have the same fingerprint which is incorrect. Also, as the time difference between the app download and install fingerprint match increases, the probability of a false match increases. Some sources claim this method's accuracy to be close to 95% while some also claim it to be 75% [1][5][6]. We at RummyCircle.com, while using branch.io for the purpose of attribution analytics found the accuracy to be close to 60%.

We have devised a novel approach to achieve highly accurate attribution of installation sources for android apps. This approach involves generating a unique identifier (UID) based on the download URL of the app. The information regarding the marketing campaign driving the app download is present in the URL. A mapping is generated of the UID along with this URL. This UID is then injected in the app's APK file when the user downloads it. This way the user receives an APK file which has the exact UID in it corresponding to the originating URL. On installation, the app sends this UID to the server to mark an installation. The server fetches the stored UID and its corresponding URL and attributes it to the proper marketing campaign to which the URL belongs. In this technique, since a mapping between the generated UID and marketing campaign URL is maintained and the generated UID is always available to the app, highly accurate installation source attribution can be guaranteed.

2 The Proposed Method

In order to achieve complete accuracy for attribution of installation sources of android apps, this method utilizes a small and insignificant loop hole in the Android system's signing mechanism for injecting the UID into the APK file. It also includes a scalable and efficient way to repeat this process and serve these UID injected APK files on the fly over http protocol, thereby generating a unique APK build for each request with a unique identifier. We depict the architecture of the whole approach in Figure 1.

2.1 Preparing the APK file for injection

An APK file is essentially a ZIP file in nature. For modifying a ZIP file, the entire file has to be copied with the modification and compressed again. This means that a signed APK file would be restructured. The ZIP utility includes a 'grow' function with which a new file or folder can be added inside a ZIP file without having to re-compress the entire file [7].

Android's signing mechanism involves calculating an SHA1 check-sum of each file in the APK. The check-sums of all files are store in MANIFEST.MF file along with its path. When an app is installed on an Android System, integrity check of all the files inside the APK is done using the entries in MANIFEST.MF. Any file with a different check-sum would immediately fail the integrity check. Similarly, any new file whose entry is not found in MANIFEST.MF would also fail the integrity check. However, the integrity check does not factor folders in an APK. Hence, if an empty folder is encountered, it would simply be ignored. Thus, an empty folder inserted inside the APK would not break Android's JAR-signed APK verification.

The prime idea of this approach is that we insert an empty function inside the 'asset' folder of the APK file with a default UID as its folder name. The folder name has a string prefix followed by a '-' and the UID string. An example is stated below.

assets/prefix-ef5d6ab67def8e062d32c12/

Inserting an empty folder inside an existing and signed APK does not require any re-compression and the APK does not need to be signed again.

2.2 UID generation and source linking

When a user clicks on an advertisement for a marketing campaign targeting app installs, the URL to download the app is hit. This URL points to the Attribution Analytics Server and also contains the information of the marketing campaign responsible for the advertisement. The Attribution analytics Server is depicted in Figure 1. It processes the request and generates a unique identifier (UID) by hashing the URL with an algorithm like SHA1. The HTTP request is then forwarded to the APK download server. The UID is passed as a query parameter with this redirection request. Simultaneously, a mapping of the URL and its UID is stored in the database.

2.3 Serving UID Injected APK files

The APK download server is a custom HTTP server written by us to serve the APK files while injecting the UID inside it on the fly. The APK file's binary is loaded in to the memory when the program starts. When the APK file is loaded in the memory, the location of the default UID is searched in the APK's binary code. This location is stored in memory along with the binary code. When a forwarded request is received from Attribution Analytics Server to this server with an UID as a query parameter, the APK file is served from the memory while replacing the default UID in the APK with the one that was passed as a query parameter. As a result, the user downloads an APK containing the UID corresponding to marketing campaign from where the user has requested to download the APK.

The APK file data is streamed into the response of the HTTP request. In the process of streaming the APK to the request's response, chunks of the binary code of the APK are loaded into the output buffer. Once the data in the output buffer have been pushed onto the TCP socket connection, the output buffer is emptied and filled with the next chunk of data. This process continues until the whole file binary is served. During streaming, a position pointer is implemented to keep track of the part of the file being streamed into the output buffer. The APK download server continuously tracks the output buffer using the position pointer and the location of the default UID in the file binary. If the output buffer is loaded with a chunk from the binary which consists of the default UID, the server replaces the default UID with the dynamic UID obtained from the request query parameters.

2.4 Attribution on First App Launch

When the App is launched for the first time, it checks for the UID folder inside the app's 'asset' folder. A pattern made up of a predefined prefix followed by a '-' and the UID string is used to search for the folder. Once a match is found, the UID is extracted from it. This UID is then sent to the server using an API call to mark to attribute the installation source of the app. The response to this API call includes the marketing campaign URL that was stored when the user clicked on the campaign advertisement. This way both the server and the app are able to accurately attribute the marketing campaign with the installation of the app.

3 Performance Experiments

We evaluate the following different methods for achieving maximum performance for the proposed solution. All of the below tests are conducted on a machine with a quad code processor of 3.2GHz clock speed and 8GB of memory. The response time comparison of these methods excluding network latency time is presented in the Table 1

Table 1: Response time and CPU usage of different approaches while serving a 20MB file.

	Response (ms)	CPU
Dynamic android build	4000	90%
Folder injection	30	20%
String UID replacement	0	4%

3.1 Dynamic Android Build

This method involves compiling a new android app build for each new download request received with the dynamic UID as a variable constant in the code. On receiving the download request, a fresh build is triggered and the APK file is served subsequently. Before the build is triggered, the UID is inserted as a variable constant inside the JAVA code of the app. As observed, the response time for this method is 4 seconds. Also, the CPU usage climbs above 90% for this period of 4 seconds. Such a high response time and high CPU footprint for a single request is not practical from a scaling point of view.

3.2 Folder injection at run-time

By making use of ZIP utility's 'grow' function, this method injects the folder in the 'assets' folder of the APK for each new request. This injected folder consists of the UID string in its name with a prefix and a delimiter. As observed from the performance test, the command injecting the folder in the APK takes about 30ms to execute. But as this process involves creating a new copy of the APK on the disk for each request, the response time increases drastically when the number of requests increase. This method has the CPU usage around 20%. Also, as new copies of APK are created on each new requests, the memory usage increases in proportion to the number of concurrent requests being served. Thus, the number of requests a server can handle using this method is very low.

3.3 Replacing UID string in the output buffer

In this method, we eliminate the need to make a fresh copy of the APK for each new request. An empty folder is injected in the pre-compiled APK file. The name of the folder includes a default UID with a predefined prefix and a delimiter. This pre-compiled APK is then loaded in memory. For each request, this APK is streamed to the HTTP request's response while the default UID is replaced in the output buffer. We have created a custom HTTP server for this purpose. This method has the smallest memory and CPU footprint as no copies of the APK are made for each request. Observations from the performance test indicate that the response time for this method is close to 0ms as the server starts streaming the APK file as soon as the request is received. As the only CPU bound task for this method consists of keeping track of the chunks of data and the default UID's position in the output buffer, it is possible to serve hundreds of request at a time. This method can approximately serve 1000 concurrent requests for a file of size 20MB using the aforementioned server configuration.

4 Future Work

If the APK generated by this approach is shared by a user to another, a repeat attribution will take place attributing the app-install to a marketing campaign. This should not happen as the source for this app's installation is Word of Mouth (WOM). We are currently extending this method to eliminate the repeat attribution using the same UID.

5 Conclusion

For Android apps hosted out of Google Play Store, this paper presents a novel approach that guarantees hundred-percent installation source attribution. Accurate attribution is critical to optimize the marketing spends on mobile app marketing campaigns. Results from the performance experiments show that this approach provides a way to have a scalable solution which provisions for high user traffic. We have deployed this solution for distribution of RummyCircle app at RummyCircle.com with great success.

References

[1] Wenjia Wu, Jianan Wu and Yanhao Wang. Efficient Fingerprinting-Based Android Device Identification with Zero-Permission Identifiers. *IEEE Access*,4:8073-8083, 2016.

[2] Google Won't Play in China. `https://www.bloomberg.com/gadfly/articles/2017-02-06/google-china-app-entry-won-t-be-something-to-shout-about`, February 2017

[3] China wants to control what apps citizens use. But will Google play ball? `http://mashable.com/2017/02/08/google-play-store-china/`, February 2017

[4] Supported locations for distribution to Google Play users. `https://support.google.com/googleplay/android-developer/table/3541286?hl=en`

[5] Device Fingerprinting Methodology. `https://help.tune.com/marketing-console/device-fingerprinting-methodology/` June 2013

[6] How to Build Deferred Deep Linking with Device Snapshotting `https://blog.branch.io/deferred-deep-linking-with-device-snapshotting/` September 2015

[7] ZIP `http://www.info-zip.org/mans/zip.html`

[8] Campaign Measurement. `https://developers.google.com/analytics/devguides/collection/android/v4/campaigns`