App Coins
Distributed and Trusted App-based Transactions Platform

Paulo Trezentos
ISCTE / Aptoide

Diogo Pires
Aptoide

Aptoide Team

September 12, 2017

Abstract

App Coins is an open and distributed protocol for App Stores. Platform agnostic. By redesigning the transactions inside an App Store - such as Advertising, In-App Billing and App approval, creates efficiencies by disintermediation and redistributes the value released in a way that create incentives for App Store dissemination. The protocol is being supported by Aptoide, an App Store wih 200 million unique users.

1 Introduction and Problem Statement

App stores are a distribution channel between the developer and the end user. Although software distribution exists since there is software development, the current model of smartphone became popular with the launch of Apple App Store in July 2008, and with its pre-load in iPhone 3G.

In the same year, but later in August 2008, Google announced the launch of Android Market[**?**], the App Store for Android.

These initial app stores followed a centralized model where one entity is responsible for assuring the core features of software distribution: file delivery, app discovery, financial transactions, and app approval. As the smartphone userbase grew, the centralized model start to show severe flaws. The flaws and problems identified are strongly related with the model: lack of trust and economical efficiciency.

By not being transparent, app stores don't create trust among the different stkeholders: developers, advertisers, users, and OEM manufacturers. Being centralized, they cannot benefit of the shared and crowdsourcing economy. Being closed source and hidden data, they don't promote competition and innovation.

The App Coins protocol cover three critical flows:

- **Advertising inside the app store**: the transactional flow where a developer pays for a user to install their app or game. There are different advertising models depending on the action that triggers the actual payment of the Ad: CPI (Cost per Installation), CPA (Cost per Action), CPM (Cost per mil impressions),... There are different technology and platforms to support it: Ad networks, Exchanges and RTB (Real Time Bidding).

- **In-App Purchase**: when there is something that the user wants to buy inside the app or the game, like gems or unlock levels, the purchase mechanism is done through the app store. To enable those transactions the developer has to integrate the SDK from the App Store or to use the app store API.

- **App approval**: in order the app to be available, the developer has to go through an approval process where the App Store manually tests the application and then screen it through automatic tools like anti-virus, anti-malware tools, and static and dynamic code analysis platforms.

In the next sub-sections, we'll analyse each of the above flows and the main problems faced today.

1.1 Advertising

1.2 Paper organization

2 Design of the Solution

2.1 Elementary Components

2.2 Protocol Overview

3 Blockchain Overview

(maybe we should call this section as something like **Protocol Construction**)

Our solution solves the use cases of Advertising, In-App Billing (IAB) and Developer Reputation within App Stores by leveraging the blockchain construction to create value for the different participants.

As it works today, advertising in App Stores is a flow composed by a campaign created by the developer, which is submitted to the several networks of campaigns. App Stores then connect to these networks and get the active campaigns, trying to have their users converting them. As it is now, the value is diluted through all the networks and the most relevant players, which are the developers and the App Stores, are the ones losing the most value, since the latter receive a small amount per conversion compared to the networks, and the former have to pay more than necessary in order to make it interesting for the App Stores to get their campaigns.

The developers and the App Stores should be the players getting the most for their effort, since they are the creators of most of the value in the flow by either creating the apps or by distributing and making them available for the users.

In IAB, developers provide content inside their apps, which can be exchanged for money. They integrate third-party SDKs that handle the payments and those third-parties (e.g. Google Play) receive a large amount of the payments, which is between 15% to 30% for the larger players in the market (Google Play [1], Apple [2]).

Once again, the players creating the most value, i.e. the developers, should be the ones actively rewarded for their effort and the players supporting them should, i.e. App Stores, not ask for so much in return.

Moreover, in both the above scenarios, the OEMs that preload the App Stores in their devices are not included in the value chain of all these transactions. Our solution comprises incentives for OEMs to preload App Stores that add the most value to users as well as to them.

The Developer Reputation problem is the one from the mentioned three that, when not addressed properly, can harm users the most. Malignant apps can harm users in several ways, ranging from the most extreme as theft of personal information and device inoperability, to the less harmful as apps containing (almost) only ads or non-working apps. For App Stores with already considerable reach, i.e. already with several millions of users, malignant apps can find their way to a significant amount of devices and thus need to be taken care of before reaching the users.

Today this assessment of apps and developers is done in a centralised way, i.e. each App Store uses its own system, and there is no shared knowledge between App Stores. Our

[1] https://support.google.com/googleplay/android-developer/answer/1153481

[2] https://developer.apple.com/app-store/subscriptions/

solution includes an approach for the categorisation of developers based on the usage of their apps by the users, and the ability to share knowledge between App Stores.

In this section, we present the data structures and algorithms used to solve each of the aforementioned use cases. It will be organised as follows: a section for each use case and subsections explaining the data structures and the algorithms.

3.1 Advertising

Advertising campaigns in the context of a blockchain construction must be constructed in a way to overcome the following problems:

- *Double attribution problem*: For all instances of the same bid (explained in 3.1.1) in different App Stores, a user can only be attributed once.

- *Bid refutation*: A developer must not be able to state that he did not create a bid if it is not true.

- *Bid validation*: A developer must be able to confirm that the attribution rules match the ones submitted at campaign (explained in 3.1.1) creation time.

3.1.1 Data Structures

Campaign. A *campaign* is a statement of intent to pay the attention of users. Developers create *campaigns* and submit them to App Stores, which in turn match and propagate them to users. Campaigns are composed by an amount of funds, a duration, the amount of tokens per attribution, and the filters (app name, app version, geolocation,...).

Bid. A *bid* is a mapping between developers and the users in an App Store that match specific campaigns created by developers. It also contains the funds available for the *bid* and the amount of tokens per attribution, and its duration.

Inverted bid. An *inverted bid* is a mapping between users and the bids they are participating in for a specific app store. See Table 1 for more details.

Ledger. The *ledger* is the record of attributions, i.e. users that did the required action (having an app open for at least 2 minutes) for a bid. The ledger is populated in a way that avoids the *double attribution problem*. Bids in different App Stores for the same apps in similar time intervals need to be identifiable across all the stores, as well as the users.

Data Structures	
Campaign	**Bid**
campaign $C_{ij} := \langle F_{ij}, \Delta t_{ij}, T_{ij}, filters \rangle_{D_i, AP_j}$ • funds F_{ij}, the amount of tokens the developer D_i is willing to spend for C_{ij} in an app store AP_j • duration Δt_{ij}, the duration of C_{ij} in AP_j • tokens per attribution T_{ij}, the amount of tokens to be sent from D_i and distributed to the other parties per attribution • filters, the specifics of C_{ij}, as the app name, app version, geolocation of C_{ij}, and others available to D_i	bid $B_i : \{D_1 \rightarrow (U_1..U_n)\}_{AP_j}$ • developer D_i, developer that submitted a bid B_i to the app store AP_j • user U_i, user matching the filters of campaign C_{ij} associated with bid B_i in app store AP_j
Inverted Bid	**Ledger**
inverted bid $IB_i : \{U_1 \rightarrow (B_1..B_n), U_2..\}_{AP_j}$ • user U_i, user matching one or more bids $B_{i..n}$ in app store AP_j • bid B_i, bids submitted to app store AP_j	ledger $L : (A_t^1..A_t^n)$ • attribution A_t^i, i-th attribution in the ledger L, which is composed as a mapping $A_t^i : \{B_N^j \rightarrow (U_N^1..UN^n)\}$, where B_N^j is the standardised bid B_j across all the app stores where it is defined and U_N^i is the normalised user U_i across all the app stores

Table 1: Data Structures

3.1.2 Algorithm construction

Create bid. When a campaign is submitted to the App Store, the associated bid B is created and inverted bids $IB_{i...n}$ are created/updated.

Ad.CreateTarget

- INPUTS:
  - Campaign parameters:
    - Funds F
    - Duration Δt
    - Tokens per attribution T
    - Filters (geolocation, app name, app version,...)
  - Developer D

  - App Store AP

- OUTPUTS: Bid B and inverted bid IB

Validate bid. When a bid is to be shown to the user, that bid is validated in the blockchain to validate if it is a valid bid and if the user matches its filters.

Ad.ValidateBid
- INPUTS:
  - User U
  - Bid B
  - App Store AP
- OUTPUTS: Result R (0 or 1)

Set attribution. When a user has been attributed to a bid, i.e. the user performed the required action (e.g. had the app open for at least 2 minutes), the ledger is checked to make sure the user has not yet been attributed to the bid and, if so the attribution is written in the ledger.

Ad.SetAttribution
- INPUTS:
  - User U
  - Bid B
- OUTPUTS: Result R (0 or 1)

4 Limitations

5 Related Work

6 Future Work

7 Acknowledgements