AppCoins
Distributed and Trusted App-based Transactions Protocol

Paulo Trezentos
ISCTE / Aptoide

Diogo Pires
Aptoide

Aptoide Team

October 16, 2017
Version DRAFT 0.40d

Abstract

Today there are 2.1 bn smartphone users in the world generating more than USD77 bn in annual gross revenue. Those numbers are projected to double by 2020. However, app stores are still riddled with inefficiencies and malware. In-app purchases (IAP) are not accessible to the low-end market, in-app advertising is plagued by too many intermediaries, malware is still prevalent and innovation is slowing down.

The reasons are diverse: payment models are not suited for emerging countries and younger generations; there is no trust between the actors of the ecosystem; and there is a lack of standardisation defining clear interfaces and enabling market free entry for new players.

The AppCoins network is an open and distributed protocol built on the Ethereum blockchain. It aims to mitigate the current inherent deficiencies of app stores. By marrying blockchain technology with app store technology, app advertising, in-app billing and app-approval can be drastically improved and sped up through disintermediation and redistributing the unlocked value to end-users and developers. "AppCoin" ERC20 token will be used by the developers to advertise their Apps to the users. From every advertising investment inside the app store, 85% goes to the user. The user has to use those coins to buy items (in-app purchase) inside the apps and games, generating the return of the investment to the developers. In parallel, the Advertising and IAB transactions are used to establish the reputation of the developer.

The design of the AppCoins protocol rests on three main pillars: 1) transparency, 2) equitability and 3) community-focused. Firstly, open and transparent standards facilitate trust and privacy. Secondly, revenue shares are redistributed away from unnecessary intermediaries to end-users and developers. Thirdly, through open-source code, knowledge is accessible to the community.

The current problems in the app stores flows are further described in this document, as well as the risks contained in each of them. Concepts like "proof-of-attribution", use Ethereum network smart contracts and state storage, allowing cryptographically to reach an acceptable digital agreement between users and developers.

This new AppCoins power app economy network will be launched within the next 12 months, leveraging on existing 200 million annual unique active users of the Aptoide app store. Token sale proceeds will be used to incentivise developers, OEMs and end-users. By 2022, Aptoide aims that 1.3 bn people use AppCoins powered app stores.

Contents

1 Introduction and Problem Statement **3**
1.1 Motivation . . . 3
1.2 Historical perspective . . . 3
1.3 Advertising . . . 4
1.4 In-App Billing . . . 7
1.5 App Approval . . . 8
1.6 Paper organisation . . . 10

2 Design of the Solution **11**
2.1 Assumptions . . . 11
2.2 Client side support . . . 11
2.3 Fraud . . . 13
2.4 Protocol Overview and sketch . . . 14

3 AppCoins: Protocol Definition **18**
3.1 Advertising . . . 18
3.1.1 Data Structures . . . 19
3.1.2 Algorithms' Pseudo-code . . . 20
3.1.3 Wallet Transactions . . . 22
3.2 In-App Billing . . . 23
3.2.1 Data Structures . . . 23
3.2.2 Algorithms' Pseudo-code . . . 24
3.2.3 Wallet Transactions . . . 25
3.3 Developer Rank . . . 26
3.3.1 Developer Reputation . . . 27
3.3.2 Data Structures . . . 27
3.3.3 Algorithms' Pseudo-code . . . 28
3.3.4 Wallet Transactions . . . 32

4 Blockchain Limitations and Proposed Approach **32**
4.1 Blockchain limits . . . 33
4.2 Existing technology . . . 34
4.3 Ethereum and Bitcoin based . . . 34
4.4 Lightning Network . . . 34
4.4.1 Raiden . . . 35
4.4.2 Plasma . . . 35
4.4.3 OmiseGO . . . 36
4.5 Independent blockchains . . . 36
4.5.1 Nxt . . . 36
4.5.2 Tezos . . . 37
4.5.3 IOTA . . . 37
4.6 Proposed Approach . . . 37

5 Related Work **39**
5.1 Related Projects 39
5.1.1 Basic Attention Token 39
5.1.2 Kin 40
5.1.3 Monetha 41
5.2 Projects Affinity 42
5.2.1 Advertising 42
5.2.2 IAB 42
5.2.3 Developer Reputation 43

6 Acknowledgements **43**

References **43**

1 Introduction and Problem Statement

1.1 Motivation

In 2016, the Aptoide app store was used to install 1.1 bn apps in 200 million unique devices[1]. Although the Aptoide brand may - or not - be familiar to the reader before reading this white paper, one out of five young people between the ages of 16 and 25 worldwide uses Aptoide. In certain countries - like Brazil or Mexico[2] - that number increases to one out of three.

Over this incredible journey to earn the trust of Aptoide users, without any paid acquisition, we discovered that app stores can be much more. The app discovery can be much better. The financial transactions - like advertising and in-app purchases - can be much more efficient. The sharing can be much more powerful. The current state does not benefit the developers or the user. It only benefits Google and Apple. In a closed market, they can impose margins and their own distribution rules.

The proposed protocol in this document is a call to the community. It is a call to developers, to other app stores, and to users. It is a call to work together towards a free-entry app store market. A market that unlocks the world of in-app purchases to billions of users. A market that benefits the talented developers providing them significant revenue and transparent ways to reach their users. AppCoins envisions a world where app stores compete by innovation and service level.

The same way open source and community helped Aptoide to reach 100 million users in 4 years, we are certain that *blockchain* is the technology that will enable this revolution, providing trust and transparency. If you share this vision, join us on this journey [26], collaborating in the protocol, developing open source software and spreading the word. Changing the app store ecosystem and breaking monopolies.

1.2 Historical perspective

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's 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 [42], the app store for Android.

These initial app stores followed a centralised model where one entity is responsible for assuring the core features of software distribution: file delivery, app discovery, financial transactions and app approval. This centralisation comes with its inherent drawbacks such as little to none transparency and being closed source. As the smartphone user base grew, the centralised model started to reveal additional flaws: a lack of trust and economic inefficiencies.

Another weakness of the centralised app economy is the intransparency behind rule-making and enforcement. For instance, apps are commonly censored if they compete with the app

[1] Aptoide web site is considered the #692 most visited site in the world by SimilarWeb (Sept 2017).

[2] Aptoide Top 5 countries: Brazil, Mexico, US, India, Italy

store owner's economic interests. In addition, collected personal user data (e.g. user age, preferences, apps installed, etc) can be exploited for other purposes. A lack of transparency erodes trust among the participating stakeholders: developers, advertisers, users and Original Equipment Manufacturers (OEM). Moreover, the central authority behind the app store reaps a disproportional high share of the created revenue in the smartphone value chain. Further, centralisation also stifles competition and innovation.

The AppCoins network redefines the following three app store core processes:

- **Advertising inside the app store**: Developers advertising to users to install their app or game. There are different advertising models depending on the intended action: CPI (Cost per Installation), CPA (Cost per Action), CPM (Cost per Thousand Impressions) and others. There are different technologies and platforms to support it: Ad networks, Exchanges and RTB (Real Time Bidding).
- **In-App Purchase (IAP)**: When users want to unlock premium features inside the app or game, the purchase mechanism is tied to the respective app store. To enable payment transactions, the developer has to either integrate the SDK from the app store or use its API.
- **App approval**: To offer the app in the store, developers have to go through a stringent approval process in which the submitted app is screened by anti-virus and anti-malware tools, as well as static and dynamic code analysis platforms. Some app stores also rely on manual app testing.

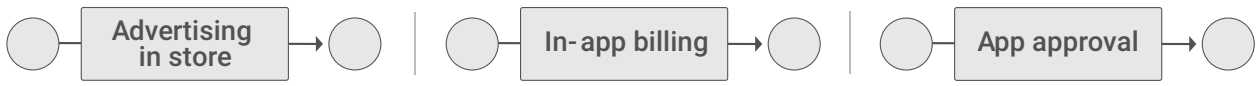

Figure 1: Individual existent core flows in app stores.

In the next subsections, each of the above flows and problems are analysed.

1.3 Advertising

The three flows presented in Figure 1 are not interacting with each other. Resources and information generated by each flow are siloed from each other. The numerous intermediaries mushroomed to address the lack of trust and integrate the various players in the fragmented market. In the following subsections, we will analyse each of the flows individually and the main problems faced today.

For a developer or a publisher, the most natural place to advertise an application or game is where the users are looking for that kind of content: the app store.

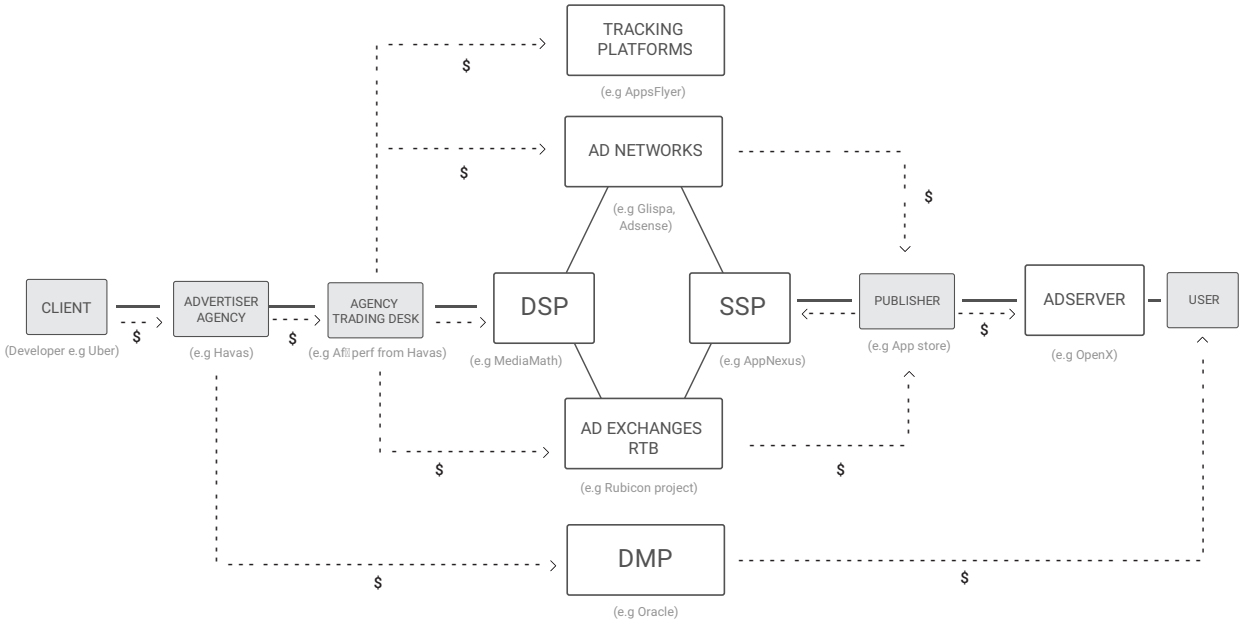

Figure 2: Cost per Installation (CPI) ecosystem.

For simplicity, we will focus on the Cost per Install (CPI) model shown in Figure 2, since the difference to the other models, like Cost per Action (CPA) or Cost per Thousand Impressions (CPM), is a matter of who shares the risk and captures the value.

In an advertising model where the advertiser (developer) bids for an installation, we have three different moments:

- *Campaign creation*: The developer (advertiser) defines the conditions for the ad to run in the store. Typically, he establishes a value for the bid representing the value that he is willing to pay for an install. There are other types of conditions called "filters", which represent target requirements. For example, requirements stating that the campaign must run in a specific country, a specific smartphone, a specific operating system version, and others.

- *Impression*: When the campaign conditions are met and the bid is competitive, the ad is shown. The user may click on the ad to see the complete description of the app.

- *Install*: If the user installs the app, thus converting the impression, an attribution is due and the corresponding money is transferred.

At each of the above moments, the lack of trust between the developer and the user carries different risks. Table 1 summarises the different risks at each stage.

The risks presented above are today managed in different ways by the advertising ecosystem and have different impacts:

Role	Campaign	Impression	Install
User			Is not a real user (*R1: Risk of fake person*) Double conversion (*R2: Risk of double attribution*) Don't use the app (*R3: Risk of no attention*)
Publisher / App store			Selling the data to third-parties (*R4: Risk of data leak*)
Developer	Not enough funds to start campaign (*R5: Risk of default*)	Run out of budget (*R5: Risk of default*)	Don't pay the conversion (*R6: Risk of repudiation*)

Table 1: Risks in advertising industry classified by action and by role.

The ***R1.1: Risk of fake person*** consists of the impression of the ad and later installation being presented to a non-real person (bot,...) with the purpose of deluding the advertiser.
The ***R1.2: Risk of double attribution*** happens with the possibility of the same user to count twice as a conversion, leading the developer to pay two times what was due.
The ***R1.3: Risk of no attention*** consists in the user installing the app that is being advertised but paying no attention to it. Even if they open the app, there is the possibility of no interaction with the app, i.e. the user opens and immediately closes the app. This leads to a zero return-of-investment.
The ***R1.4: Risk of data leak*** consists in the information regarding the user being leaked to third-parties for advertising purposes. Information about the user's preferences are aggregated in Data Management Platforms (DMP) and later used by advertisers in programmatic / RTB targeting.
The ***R1.5: Risk of default*** consists in the developer creating a campaign but not having enough funds to pay the conversions that are generated in that campaign, leading to him not paying the due amount.
The ***R1.6: Risk of repudiation*** happens when the developer does not recognise the installation, failing to attribute the conversion to the publisher. The attribution is generally monitored by tracking platforms like AppsFlyer, Adjust or Kochava that have multiple variables that can be changed by the developer to define what it considers a real attribution. These variables can take in account the time window period between the click URL and the conversion, the network fingerprint, among others. Attribution, or the lack of it, is harming the industry with only 15% to 25% of the real installations being considered conversions[a].

[a]Values based on Aptoide experience.

These risks in the Advertising flow will be considered in a section ahead in the design of the AppCoins blockchain.

1.4 In-App Billing

In-App Billing (IAB), also called In-App Purchase, consists in the possibility for the user to buy digital items inside an app or a game. Although those items are perceived to be bought inside the App, the items are bought through the app store.

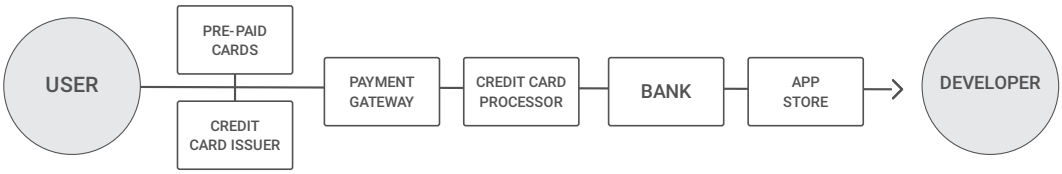

Figure 3: Current IAB flow and intermediaries.

The need for the transactions to go through the app store were introduced as mandatory by Apple App Store and then by Google Play. The conditions for the developer's app to be distributed is that all the financial transactions have to be managed by the app store.

The app store adds some value to this flow: 1) it may know the customer already and have their payment data, thus easing the entrance hurdles for the user and providing a better user experience, 2) it has the trust of the user when the developer may not have it yet and 3) it develops the necessary technology, allowing the developer to focus on the app development.

Although IAB represents a market with a huge volume of transactions processed by Google Play and Apple, there are still two big challenges.

The number of users with a credit card loaded in the store is still a minority. Only small part of the world population has access to credit card. Alternative methods like pre-paid cards are an approach but they are physical and depend on points of sale, therefore do not scale well.

On the other hand, some of other payment methods like carrier billing have prohibitive margins that compromise the revenue share of 70% for the developer. In some markets, the margin of the telecom operator varies between from 35% to 60% of the cost of the transaction. The reasons given by the telecom operators are: 1) high risk of fraud that has to be compensated and 2) the users may cannibalise the telecommunications balance so the margin has to pay that possibility.

Providing the user has a proper payment method, there are still some risks that have to be mitigated:

The ***R2.1: Risk of user data leak*** consists in the information regarding the user being leaked to third-parties for advertising purposes. Information about the user preferences are aggregated in DMP platforms and later used by advertisers in programmatic / RTB targeting.
The ***R2.2: Risk of digital goods lost*** may happen when a user buys a digital good inside the game or app but it is not delivered. Often, the user does not have a way to recover the payment or claim the digital good.
The ***R2.3: Risk of double payment*** occurs when the user pays twice for the same in-app item purchase. Also in this case, the user may not have a proof that they paid twice.
The ***R2.4: Risk of digital items cloning*** when the user is able to duplicate and transfer the digital good to another user, leading to losses for the developer that charge once for a digital good that is used twice.

A platform that handles the IAB transactions has to deal with those risks.

1.5 App Approval

The app approval is one of the more critical challenges of an app store. By definition, the app store is a channel between the developer and the user.

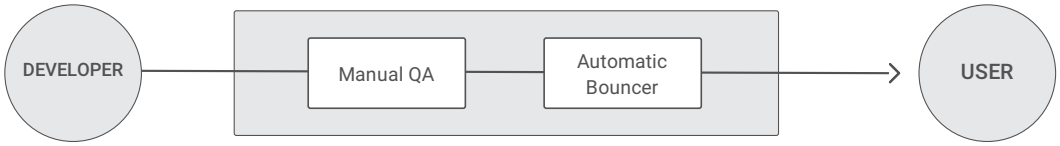

Figure 4: App approval in centralised App Stores.

In order to enforce security, legal and business requirements, app stores define limits in terms of acceptable app behaviour and/or content. These policies also mirror the store's philosophy (e.g. defining the acceptable content) and protect both users and developers against unwanted or potentially dangerous behaviour, thus promoting trust. Policies can include general categories such as safety - protecting against malware behaviour, offensive content or physical harm - or legal - protecting privacy and intellectual property. More restrictive stores such as the Apple App Store also imposes strict rules regarding the user interface design, minimum functionality and quality. [10, 17]

The risk of infringement occurs when new apps are added to the store. Therefore, stores which are open to public upload of apps (e.g. Google Play Store or Apple Play Store allow submission by developers) need to ensure that uploaded apps abide by their rules by putting them through a reviewing process.

The app screening may be performed through manual and/or automatic processes and differ between stores as they are defined by their own policies. The manual process involves a group of

people (typically belonging to the Quality Assurance and/or the Security Team) who manually install and test apps on real devices. They examine the apps' behaviour and content in order to decide whether each app respects the store's policy. The automatic process consists of a computer program which automatically analyses the submitted apps and compares features to a given dataset of rules, signatures, unwanted apps, content or behaviour. Multiple techniques may be used by the program to automatically classify given apps into unwanted, accepted or unknown states [3].

Google's Play Store and Apple's App Store, the current largest and most well-known app stores, use a combination of both processes. When a new app is submitted to their store, they first go through an automatic process which will automatically discard identified unwanted apps and then proceed to the manual process. However, the two stores differ in the techniques they use in their automatic processes and the amount of apps that go through manual reviewing [15, 11].

Apple App Store approval flow is simple. All submitted apps go through an automatic static analysis process, a method which examines the app code without running it. In this process, the apps are analysed for traces of calls to Apple's private API as the company's policy only allows calls to their public API. The identified apps are discarded while all the remaining apps are passed on for manual review [15, 23]. Apple states the following most common reasons for failing their strict manual reviewing process: crashes and bugs, broken links, placeholder content, incomplete information, inaccurate description, misleading users, substandard user interface, advertisements, web clipping, similar apps and not enough lasting value [18]. According to Apple, the complete review process takes on average between 24 (50%) to 48 hours (90% of submitted apps) [16].

Google's Play Store has a more evolved reviewing system where the automatic process involves a complex machine learning engine. This engine relies on multiple technologies including static and dynamic analysis (where both code and runtime behaviour is analysed), heuristic and similarity analysis (for finding new trends of unwanted apps) and signatures (identifying known unwanted apps). The engine also includes features from external independent security research as well as the developer's behaviour (history with other apps and billing profile) as well as metadata such as ratings and downloads. The automatic process assigns to each application a risk level ranging from safe to harmful. Low risk applications are automatically accepted and high risk applications are automatically rejected. Apps with medium risk level are submitted for manual review [11].

Although these app approval systems are capable of detecting a large number of unwanted apps, they also pose problems to developers. Apple's automatic process has shown problems with false positives and rejecting legitimate apps [15] and their strict policy is known to frequently change the categories of rejected apps, posing problems to developers of such apps. Also, Apple's reviewing process strongly based on human analysis is known to have flaws, namely not being able to detect apps which hide their malware behaviour by being inactive for a given amount of time and showing a regular behaviour in order to escape the human test [40]. Other reports [25] have also shown a big presence of scamware in Apple's store where apps are able to scam users into paying for unneeded services. Google's more automated system has also been shown to have flaws [23] due to its more permissive system with apps being accepted

without manual evaluation. Frequent security reports show breaches in the security control of Play Store reporting the existence of multiple malware (ransomware, backdoor and trojans) infected apps compromising several millions of devices and thus posing serious threats to users [6, 22, 30].

Both Apple's App Store and Google's Play Store have a history of refusing and banning apps. Examples of recent complaints include the rejection of the social network GAB's app by both Apple and Google [24], the refusal of music streaming service Spotify's app update [21] or the Anti-Spam App [34] by Apple and the rejection of Popcorn Time, TubeMate, Adguard or Fildo by Google [1]. However, these rejections are often considered unfair by developers who claim an abusive and anticompetitive behaviour as the apps conflict with other services provided by the app stores and have motivated a number of complaints to legal authorities [33].

As described above, several risks can be found from the current centralised app approval processes:

The ***R3.1: Risk of Malware*** is the possibility of an app or game submitted to the app store being infected with a virus or malware that steals data or damages the user's device.
The ***R3.2: Risk of user data leak*** consists in the information regarding the user being leaked to third-parties for advertising purposes. Information about the user's preferences are aggregated in DMP platforms and later used by advertisers in programmatic / RTB targeting.
The ***R3.3: Risk of censorship*** when an app store blocks the publishing of an app or game based on political, religious or social factors that are subjective and totally unrelated with technical aspects. The censorship can be self-inflicted when the company running the app store follows orders or guidelines of national governments or can be result of technical external restrictions that limits the access to the app store (The Great Firewall of China, for example).
The ***R3.4: Risk of arbitrary decisions*** happens when the app store denies the distribution of an app based on "anti-competition clauses" [9] or other reasons only related to its business interest, even if the interest is in other markets or industries.

1.6 Paper organisation

This paper is organised in the following chapters. In this chapter we started to introduce the flows, the current challenges and the flaws they carry.

Chapter 2 will propose the overall design of the solution for the core app store flows supported by blockchain technology.

Chapter 3 will dive deep in the blockchain technology, presenting the main data structures and algorithms that are proposed.

The current limitations of the blockchain technology when applied to app stores are introduced in Chapter 4.

In Chapter 5, related work that shares common approaches with the AppCoins protocol are introduced, as well as projects that inspired parts of the AppCoins protocol.

The future protocol developments will be included in Chapter 6 and this document will end with acknowledging the contributions of the several community members that contributed to this document with their suggestions and opinion.

2 Design of the Solution

2.1 Assumptions

Building a model always depends on making certain assumptions. The generalisations taken in the assumptions allows one to focus on the global mechanics and do not take extreme/corner cases into account. As exceptions, they are part of the reality but they do not have material importance to change the result of the model.

The design of the AppCoins platform was made having the following eight assumptions:

- ***A1 Crowdsourcing***: Community wisdom works for big numbers better than individual wisdom [39];
- ***A2 Incentives***: If there are enough incentives, the community contributes;
- ***A3 New***: A new developer is always an unknown developer accurately;
- ***A4 Trusted:*** The Apps from a Trusted Developer are Trusted Apps;
- ***A5 Dispute***: - In a dispute, if 50% + 1 of the community is honest, the right side wins;
- ***A6 Reputation***: - Transactions registered in the blockchain ledger (IAB, Ads) reflects well the trustworthiness and reputation of a developer;
- ***A7 Unknown downloads***: - If the app downloads made by the users come from more than 5% of unknown developers, users will start to have trust in unknown developers.
- ***A8 Zero-day***: - A community dispute may take 30 days. While the dispute is handled, the app stores have the option to hide the app to avoid zero-day attacks.

2.2 Client side support

Besides the blockchain technology, the environment where the user is running the app store should also support the AppCoins protocol to be able to monitor if the advertised app is really used during two minutes.

As Android represents 86% of the smartphones market, we have focused our implementation analysis in this platform. However, many of the constraints and solutions are applicable to other smartphone operating systems.

This subsection aims to describe a client-side method, running in the smartphone's untrusted environment, to register that a user is paying attention to an app (installed from an app store)

for a certain amount of time. Once those requirements are met, a *proof-of-attention* (PoA) is issued using through a server-side platform and stored in the blockchain.

When designing a solution, two main factors should be taken into account: reliability and availability. Reliability consists of avoiding fraud. Once a PoA is generated, the solution needs to have a high level of confidence that a user paid attention to an app installed from an app store. Availability consists of making sure that whenever a user pays attention to an installed app, the app store will recognise the event and will request the PoA once the requirements are met.

The app store process has to be running when the user is paying attention to an app in order to request the issue of PoA. Latest releases of the Android operating system (Android OS) - from Lollipop (API level 21) onwards - have been limiting the ability of app processes to run while the app is in the background. In our scenario the app store process could eventually be killed by the Android OS while not being in the foreground. In order to overcome that issue, we will take advantage of the Binder framework to bind the app and app store's processes while the app is in the foreground, which will ensure that the app store process is not killed by the Android OS.

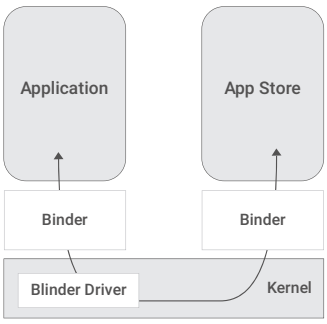

Figure 5: Operating system binder.

Binder framework is a core component in Android architecture and its main goal is to simplify Inter- Process Communication (IPC). Binder is implicitly used whenever an app communicates with OS services or with other apps through the Android Java API Framework. The Binder framework will also provide information regarding the app process, which will positively contribute to the app store PoA reliability.

Once the app and the app store's processes are bound, the app store will periodically verify whether the app is in the foreground and the user is actively interacting with the device. In order to certify that the user is paying attention to the app, the following conditions should be met: the app's process must be bound to the app store's process, the app must be in foreground, the device screen must be on, the device must not be locked, and the signature of the app must be verified on the app store's servers.

To assure that the app's process is bound to app store, Binder and PackageManager APIs can be used. To verify whether an app process is in the foreground, both ActivityManager,

UserStatsManager and PackageManager APIs can be used. To check whether the device screen is on, the PowerManager and Display APIs can be used. Regarding the state of the lock screen, the KeyguardManager API can be used.

Every application has to be signed by the developer before being installed on an Android device. App stores have access to the apps' signatures and can validate by confirming whether they match with the signature on their servers. If the signature does not match, the app may have been tampered with. To obtain the app's signature, the Binder and PackageManager APIs can be used.

The proposed solution has some limitations regarding reliability - imposed by Android's inherently insecure environment - and Android API availability - due to Android's version fragmentation and app store permission level. The use of several different Android APIs can help harden the solution against an attacker but can not assure full protection against fraud on the client side. The AppCoins protocol aims to find a balance between the effort of breaking and the incentives of doing it.

2.3 Fraud

One important concern is naturally the possibility of fraud by a (fake) user to earn more tokens. The only certainty is that it is not possible to totally avoid fraud. The mobile phone is an untrusted environment that can be tampered. Today, fraud in installation attribution exists, like seen on videos of farms of smartphones in a rack where clicks in banners are done automatically.

AppCoins protocol deals with fraud in CPI advertising with the following design:

- **Thresholds**: The app store sets a lower limit, e.g. one install a day and three per week for unknown users. As the user has more transactions in the blockchain, the level of trust increases and the threshold is slightly raised. The user would never be able to do more than two installations per day in any case.

- **Cash out**: As mentioned before, the user is not able to cash out tokens. He has to use the AppCoin tokens for in-app purchases. A fraudulent user could try to register as a developer and buy items inside his own app, but 15% of the transaction volume would be lost (app store and OEM margin) and it would be easier to detect a fraudulent pattern for that user.

- **User Fingerprint**: The fingerprint of the user (a unique identification to detect double attribution) is not done only at the smartphone level but, more importantly, at networking level. The *IP* of the user smartphone, as well as routing information, is used in the fingerprint.

- **Shared information**: The existent transactions with the fingerprint information will persist in the blockchain, allowing the different app stores to access and avoid cross-store fraud. The partial anonymity of the user is accomplished by using hash functions (like SHA256) in the fingerprint.

- **Targeting**: It is the store that sends the Ads for the user, trying to do an intelligent matching between the user and the running campaigns. The client cannot try to install an advertised app that was not targeted to him.

As mentioned before, it is impossible to totally eradicate fraud since the smartphone is an untrusted environment. The goal is to create enough difficulties for the attacker beyond the point that the effort and resources needed to explore the attack vector do not compensate the benefits extracted from that attack.

2.4 Protocol Overview and sketch

The AppCoins protocol is depicted in Figure 6. It consists in 3 main blocks: Advertising, Developer's Reputation (Rank) and IAB.

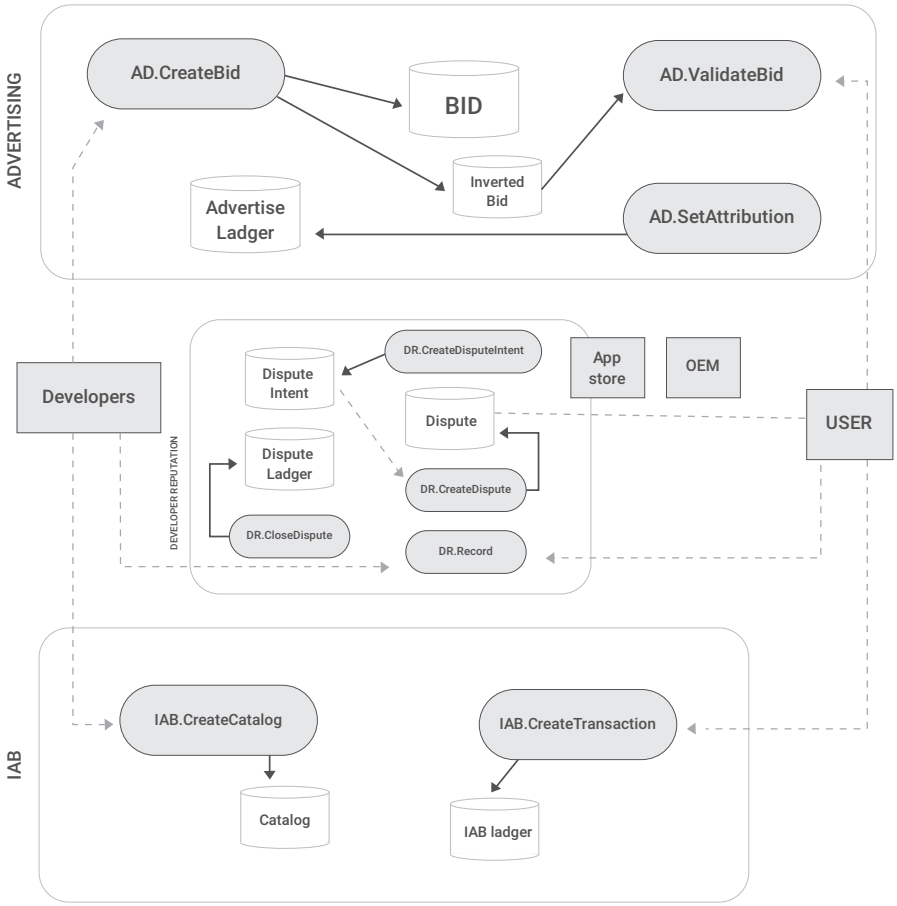

Figure 6: Overall design of AppCoins and blockchain interactions.

Inside each block we see the interactions between the ecosystem players and the blockchain. The rounded squares represent functions / methods of smart contracts that implement business

logic. The cylinders represent data stored in smart contracts' own storage or event logs.

Advertising

The protocol was defined such that most risks in Section 1.3 are avoided or at least mitigated. By leveraging blockchain technology and its requirements of transparency, accountability and verifiability of processes, we avoid creating a middlemen-dependent economy and add value both for the advertisers/developers and the users.

As explained in Section 1, there are three moments in an advertising model: campaign creation, impression and attribution (which can be an installation, opening of the app, etc).

In AppCoins protocol, attribution does not happen when the user installs the app or game, but after the app has been opened for 2 minutes. This poses the problem of how can the app store prove that a user did indeed open the app and had it opened for the required period of time. Our protocol introduces a *proof-of-attention* PoA, which allows the developers to verify that users did actually paid attention to the app for the required time. Since the smartphone is an untrusted environment, there is not a way to ensure that the user is real or that the app store stub installed in the phone was not tampered. However, identity is server-side checked by the app store using network fingerprint (IP, routing information, etc) as it is done today by the tracking platforms. Similar challenges to PoA are addressed in BAT [4].

From the requirement of privacy, the protocol itself does not expose any user information at any time and interaction with app stores. The only information that can be seen by others are the wallet addresses when a transaction occurs. Concerning risk R1.4, since the addresses are not linked to any sensitive user information (e.g. email), there is no leakage of user data.

Regarding risk R1.6, developers can claim that a certain user did not actually do the required action to be given the attribution, i.e. that either the app store, the user or both are being dishonest. By providing a PoA and storing the transactions in the blockchain, they become verifiable by anyone, including the developer. Since the developer can verify the proof, it becomes clear that the protocol avoids the risk of repudiation.

Since our solution proposes that no middlemen is needed within the advertising model, there is the need to have the protocol being able to enforce the payments between the developer and the other parties, which are the user, the app store and the OEM. The only way to make sure the entirety of the funds that the developer wants to put available for the campaign exist is to lock them in a different wallet, which is where the smart contract for the campaign will be running. Since they are locked, the developer cannot spend them before the campaign is over and there is no possibility for the developer to be in default. In addition, when there is an impression, the amount of tokens to be paid for that conversion need to be locked as well. This serves to avoid race conditions, which can occur when the funds still available in the campaign are for X attributions but Y users are getting impressions for the campaign (with $X < Y$). If this would be possible, some of the X users would install the app but when they opened it and paid attention to it for the required time, thus being eligible for the attribution, they would not get the attributed because there would be no more funds available. This means that

different wallets need to exist that serve to lock funds for the already made impressions. If the attribution for a certain impression does not occur after some designated amount of time, then the funds would be unlocked and placed back in the campaign wallet. This funds captivation scheme is exemplified in Figure 8 and avoids the risk R1.5.

IAB

The IAB use case has the risks outlined in Section 1.4, which the protocol needs to either avoid or mitigate.

As in the Advertising use case, since the protocol follows the requirements of blockchain technology, and in particular the requirement of privacy, the protocol only exposes wallet addresses when transactions occur. These need to be publicly available in order to have the possibility of verifying transactions but do not carry any link to personal user information (for example, the email of the user). Therefore, risk R2.1 is partially avoided.

The protocol is built to allow for transparency between users, developers and app stores. Transparency means that transactions between parties are public and verifiable, but only regarding token exchange. This means that the protocol is not intended to store items or any other form of goods/value. This tracking, as it is today, is to be done directly by users and developers, i.e. when the user pays for an in-app item, it is the user's responsibility to check if the item is transferred or not. The store, retrieve, and exchange of digital items between users is out of the scope of the AppCoins solution. Nonetheless, having the transactions publicly available mitigates this problem, which is referred to as risk R2.2, but does not completely avoid it. However, because of having the transactions publicly available, the protocol avoids risk R2.3 because the user only sends more tokens in exchange for an in-app item if wanted.

Regarding item cloning, as it has been said, the protocol itself does not deal with in-app items or their exchange, only with the token transfers in the scope of in-app purchases. Hence, there is no possibility within the protocol to clone items and neither to send them to other users. Therefore, risk R2.4 is not addressed by the protocol at this version.

Developer Reputation

This use case is the one that may dictate if a developer is trustable or not. Therefore, if the protocol defines ways to mitigate and avoid malware attacks across the app stores operating within the protocol, then the adoption of the protocol by app stores can be greatly increased.

As it was said in Section 1.5, the malware scanning process is different for all the app stores, as are the reasons for blacklisting apps. An app store that is not showing an app to its users because it is from a competitor or for censorship purposes, although the app was considered "trusted" in the blockchain, is then exposed in the community. If the malware knowledge is not shared across app stores - the current state - will fin act harm users because one app with malware may have already been blacklisted in an app store but it still available on others. Our protocol deals with this problem by attributing a reputation level to the developers, which is then extended to all their apps.

The reputation of a developer has two components: the rank and the rank level. The rank can have the values of *"Unknown"*, *"Trusted"* or *"Critical"* and it states if a developer is new to the community, if it is already known and considered honest or if it is considered dishonest, respectively. The rank level is represented by an integer starting at 1 and does not have a maximum level for the *"Trusted"* rank. However, for the other ranks it is fixed at 1. This derives from the fact that when a developer is *"Unknown"*, it is just because the developer is new in the network and the goal is that the developer leaves the rank to become *"Trusted"*. On the other hand, being a *"Critical"* developer means the developer is dishonest and we do not intent to have different levels of dishonesty in the network. Figure 7 shows the different ranks a developer can have and how to move from one rank to another.

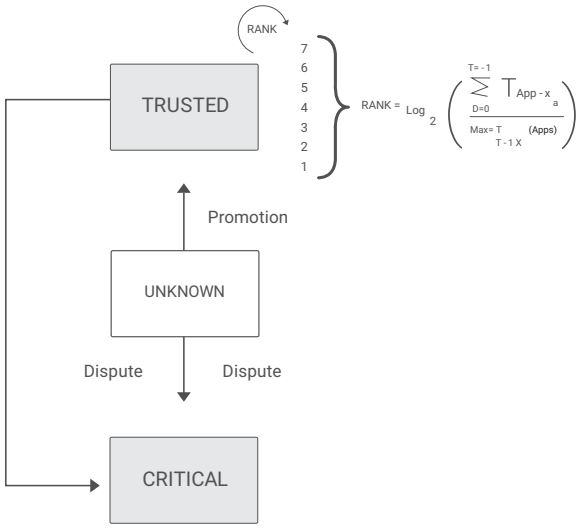

Figure 7: Approval state changes.

As it can be seen, there are two processes by which the rank can change: *promotion* and *dispute*. A *promotion* is an automated process which takes into account the number of transactions occurred in the developer's apps and compares them to the transactions of other popular apps from trusted developers. If the developer's apps compare well against others (this comparison is detailed in Section 3.3), the rank is either increased to *"Trusted"* or the rank level goes up one level in the case where the developer is already *"Trusted"*.

Regarding the *dispute*, it is a mechanism to punish developers that deliver bad apps, either because they contain malware or they are fake, i.e. do not do anything and may contain only ads. This process of judging developers has been a centralised process in app stores and our protocol introduces a way to have it done by the community, since it is the community that uses the apps. The protocol provides a way to open a dispute against a developer, i.e. any user can claim that a developer is dishonest. When a dispute is started, which at this point we call as a *dispute intent*, any user can answer it within the next 7 days. The user answering it,

which would mean that the user is claiming the developer is honest, can be any user and does not need to be the developer. If the dispute intent is answered, a dispute is opened and stays open for 30 days. Within this time period, any user can join any side of the dispute, either the *Contestants* - users who claim that the developer is dishonest - or the *Pleaders* - users who claim the developer is in fact honest. If the dispute intent is not answered by anyone, the dispute intent is closed without opening a dispute and the developer's rank changes to *"Critical"* along with the rank of all the apps of that developer.

In order to join a dispute, a user needs to put an amount of tokens into stake. The amount of tokens is chosen by the user and express the confidence in the side the user is joining. When the 30 days of the dispute are over, the side which collected the biggest amount of tokens wins. The winning side gets a refund of the tokens pledged plus 10% of the pledge of the losing side. This 10% is divided according to the percentage each user had pledged within the winning side pledge. The losing side gets a refund, where the pledge of each user has a cut of 10%. Regarding the change in the developer's rank, if the *Contestants* win, the developer's rank changes to *"Critical"* and all the apps of the developer also have their rank changed. On the other hand, if the *Pleaders* win, the developer's rank remains unchanged.

The dispute mechanism transfers power back to the community, which can also include the app stores. Since it is the community deciding whether a developer and the corresponding apps are to trusted or not, the developers reputation and the reasons for it are public to everyone. If there is a change in the reputation of a developer, the community knows when it happened and knows it was a decision made by them, instead of being obscurely made by one app store. This mitigates the risks R3.3 and R3.4.

Risk R3.2 is similar to risks R1.4 and R2.1 and the reason why it is avoided by the protocol is the same.

3 AppCoins: Protocol Definition

As we have seen before, the AppCoins protocol addresses 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.

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 core flow and subsections explaining the data structures, algorithms and wallets flows.

3.1 Advertising

Advertising campaigns in the context of a blockchain must be constructed in a way as to overcome the risks elaborated before.

3.1.1 Data Structures

Bid. A *bid* is a statement of intent to pay for the attention of users. Developers create *bids* and submit them to app stores, which in turn match and propagate them to users. Bids are composed of an amount of funds, a duration, the amount of tokens per attribution, the filters (app name, app version, geolocation,...), and a mapping between developers and the users in an app store that match specific filters set by the developers. Please refer to Table 2 for more details.

Inverted bid. An *inverted bid* is a mapping between users and the bids they are participating in for a specific app store. See Table 2 for more details.

Advertising Ledger. The *advertising ledger* is the record of attributions, i.e. users that completed the required action (e.g. 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*, i.e. 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	
Bid bid $B_a : \langle F_{ij}, \Delta t_{ij}, T_{ij}, filters, D_i \rightarrow (U_1..U_n) \rangle_{D_i, AS_j}$ • Funds F_{ij}, the amount of tokens the developer D_i is willing to spend for C_{ij} in an app store AS_j • Duration Δt_{ij}, the duration of C_{ij} in AS_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 • Developer D_i, developer that submitted a bid B_i to the app store AS_j • User U_i, user matching the filters of campaign C_{ij} associated with bid B_i in app store AS_j	**Inverted Bid** inverted bid $IB_i : \{U_1 \rightarrow (B_1..B_n), U_2..\}_{AS_j}$ • User U_i, user matching one or more bids $B_{i..n}$ in app store AS_j • Bid B_i, bids submitted to app store AS_j
Advertising Ledger advertising ledger $L_{Ad} : (A_t^1..A_t^n)$ • Attribution A_t^i, i-th attribution in the advertising 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 2: Data Structures for Advertising Use Case

3.1.2 Algorithms' Pseudo-code

Table 3 presents in pseudo-code a more in-depth definition of the following methods.

Create bid. When a bid is submitted to the app store, the CreateBid method creates the bid B containing the mapping M between the developer D and the set of users U of the app store that match the filters, as well as all the parameters describing the bid as the funds F, duration Δt, etc. The inverted bids $IB_{i...n}$ of the app store containing the mapping between users and the bids they are in are created or updated, if they already exist. In addition, the funds F the developer wishes to allocate to the campaign are sent from the developer's wallet W_D to the bid's contract wallet W_B. Since the created bid is in the blockchain accessible to every user, it is used to overcome the *bid refutation* problem.

AD.CreateBid

- INPUTS:
  - Campaign parameters:
    - Funds F
    - Duration Δt
    - Tokens paid per attribution T
    - Filters (geolocation, app name, app version,...)
  - Developer D
  - App Store AS
- OUTPUTS: Bid B and inverted bid IB

Validate bid. When a bid is to be shown to the user u, ValidateBid validates that bid in the blockchain to confirm if it is a valid bid and if the user u matches its filters. In order to avoid the scenario where there are not enough funds available in the bid for the user, the tokens per attribution T defined in the bid are sent to the bid's captivation wallet W_C for a period of time. The method is used to overcome the problem of *bid validation*.

AD.ValidateBid

- INPUTS:
  - User u
  - App Store AS
- OUTPUTS: Result R (0 or 1)

Set attribution. When a user has been attributed to a bid B, i.e. the user performed the required action (e.g. had the app open for at least 2 minutes), SetAttribution checks the advertising ledger L_{Ad} to make sure the user u has not yet been attributed to the bid B and if so, the attribution is written in the ledger and each participant receives the correspondent tokens,

i.e. the user, OEM and app store receive T_u, T_{OEM} and T_{AS}, respectively. Since the method is constructed in a way such that the same user is unable to be attributed the same bid in different app stores, it avoids the *double attribution problem*.

AD.SetAttribution

- INPUTS:
  - User u
  - Bid B
- OUTPUTS: Result R (0 or 1)

Advertising Use Case

AD.CreateBid

- INPUTS:
  - Campaign parameters:
    - Funds F
    - Duration Δt
    - Tokens paid per attribution T
    - Filters (geolocation, app name,...)
  - Developer D
  - App Store AS
- OUTPUTS: Bid B and inverted bid IB

1. Compute $U :=$ GetUsers(AS, $filters$)
2. Compute $M :=$ Mapping(D, U)
3. Compute $B :=$ CreateBid(F, D, Δt, T, $filters$, M)
4. Send F from developer's wallet W_D to bid's wallet W_B
5. For each u in U:
   (a) Compute $IB_u :=$ GetIB(u)
   (b) If $IB_u = -1$:
      i. Compute $IB_u :=$ CreateIB(u)
   (c) Else:
      i. Compute IB_u.append(B)

AD.ValidateBid

- INPUTS:
  - User u
  - App Store AS
- OUTPUTS: Result R

1. Compute $B_u^{'} :=$ GetBids(u, AS)
2. Compute $IB_u =$ GetIB(u)
3. Set $IB_u^{'} := \{u \rightarrow (B_u^{'1}..B_u^{'n}) = B_u^{'}\}$
4. Compute $R :=$ CheckMatch(IB_u, $IB_u^{'}$)
5. If $R = 1$:
   (a) Send T from bid's wallet W_B to bid captivation wallet W_C

AD.SetAttribution

- INPUTS:
  - User u
  - Bid B
- OUTPUTS: Result R

1. Compute $InLedger :=$ CheckAdvertisingLedger(u, B)
2. If $InLedger = 1$:
   - Set $R := 0$
3. If $InLedger = 0$:
   (a) Compute $TX :=$ Transaction(u, B)
   (b) Compute $R :=$ WriteAdvertisingLedger(TX)
   (c) Compute $(T_u, T_{OEM}, T_{AS}) :=$ DivideTokens(T)
   (d) Send T_u to user's wallet W_U
   (e) Send T_{OEM} to OEM's wallet W_{OEM}
   (f) Send T_{AS} to user's wallet W_{AS}

Table 3: Advertising Use Case

3.1.3 Wallet Transactions

The transfers between wallets in the Advertising flow have to address some risks that were previously described in Chapter 1.3.

There is a wallet that contains the budget of the created campaign, which is meant to lock the budget, addressing the risk of default (R1.5) by the developer. To address the same risk,

there is also a wallet that will temporarily store the value of the impression, to make sure that if the attribution occurs within a certain time, i.e. the user installs and opens the app within a certain time interval, there are funds available to pay for the conversion. Otherwise, there could be a scenario of serving an impression to many users but the campaign only had enough funds for one more attribution. The users would try to get attribution, creating a race condition, but only one would get it. An example of this schema is shown in Figure 8.

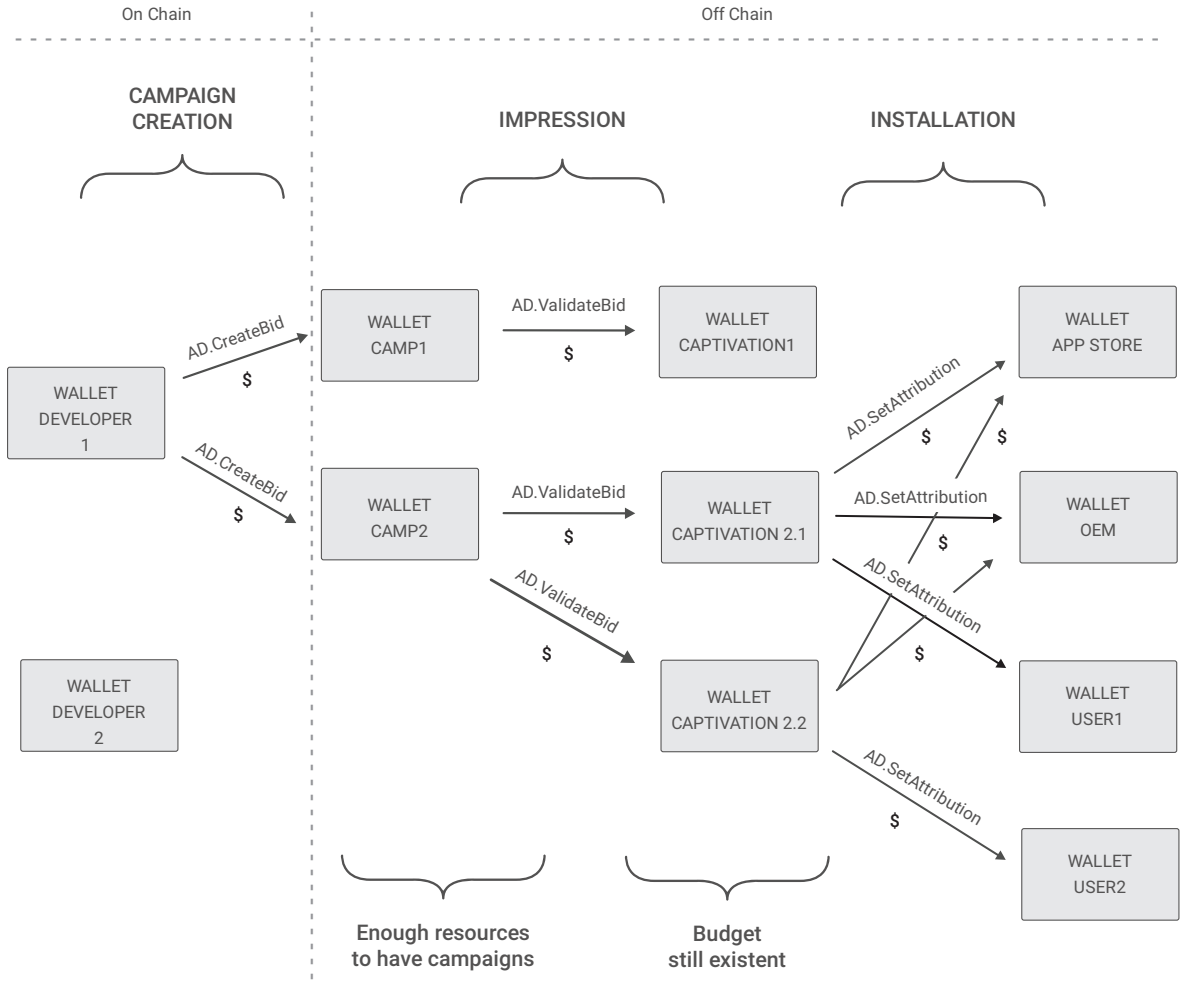

Figure 8: Wallet transfers in the CPI advertising flow.

3.2 In-App Billing

3.2.1 Data Structures

Catalog. A *catalog* is a mapping between items available in an app and their prices. Each item has only one price and the mapping is bonded to an app store, i.e. the price can be set

differently for different app stores.

IAB Ledger. The *IAB ledger* is the record of items bought by users. It records transactions in a way that anyone can verify an anonymous user bought a quantity Q of an item I for a price P from an app that integrated the IAB solution from app store AS.

Data Structures
Catalog catalog $C_{ij} : \{I_1 \rightarrow P_1..I_n \rightarrow P_n\}_{A_i,AS_j}$ • Item I_i, an item available in app A_i which integrated IAB solution from app store AS_j • Price P_i, the price of item I_i **IAB Ledger** IAB ledger $L_{IAB} : (TX_1..TX_n)$ • Transaction TX_i, a transaction stating that an anonymous user u bought a quantity Q an item I with price P from an app A that integrated the IAB solution from app store AS

Table 4: Data Structures for IAB Use Case

3.2.2 Algorithms' Pseudo-code

Create catalog. When a developer D wants to integrate in-app purchases, a catalog C is created containing the mapping between the items I_N that are to be available in the app A and their respective prices P_N.

IAB.CreateCatalog

- INPUTS:
  - Set of items $I_N = (I_1..I_n)$
  - Set of prices $P_N = (P_1..P_n)$
  - App A
  - App store AS
- OUTPUTS: Catalog C

Create transaction. When a user u wants to buy a certain amount Q of items I, a transaction is created stating that the user u bought a quantity Q of an item I for a price P in a app A that integrated the IAB solution from app store AS.

IAB.CreateTransaction

- INPUTS:
  - User u
  - Item I
  - Quantity Q
  - App A
  - App store AS
- OUTPUTS: Result R (0 or 1)

IAB Use Case	
IAB.CreateCatalog • INPUTS: – Set of items $I_N = (I_1..I_n)$ – Set of prices $P_N = (P_1..P_n)$ – App A – App store AS • OUTPUTS: Catalog C 1. Compute $M :=$ Mapping(I_N, P_N) 2. Compute $C :=$ Catalog(M, A, AS)	IAB.CreateTransaction • INPUTS: – User u – Item I – Quantity Q – App A – App store AS • OUTPUTS: Result R (0 or 1) 1. Compute $TX :=$ Transaction(u,I,P,Q,A, AS) 2. Compute $R :=$ WriteIABLedger(TX) 3. if $R = 1$: (a) App A issues items to user (b) Compute $(T_D, T_{OEM}, T_{AS}) :=$ DivideTokens(F) (c) Send T_D to developer's wallet W_D (d) Send T_{OEM} to OEM's wallet W_{OEM} (e) Send T_{AS} to user's wallet W_{AS}

Table 5: IAB Use Case. In CreateTransaction, the issuing of items in certain app A is purely done in the app based on the result of the method, since the items are not in the blockchain and there is no real blockchain transaction happening.

3.2.3 Wallet Transactions

In In-App Billing the transactions between wallets occur off-chain. The main transactions are between the user that is buying the digital item and 3 recipients: the developer, the OEM and the app store. Figure 9 presents the different flows of IAB transactions.

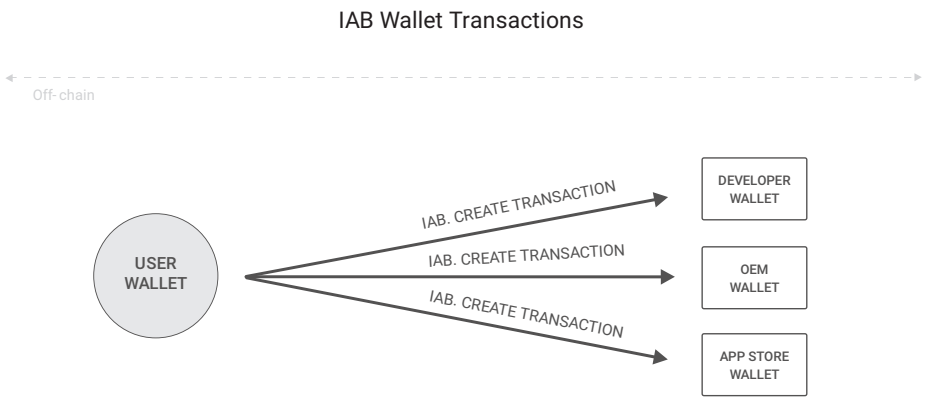

Figure 9: Wallet transfers in IAB flow.

3.3 Developer Rank

There is the need to create trust between the different players in the app economy, namely between users, the developers and their apps.

In the AppCoins protocol, trust is characterised by a developer's rank, which is then propagated to all his apps. This rank can have the values of $\{"Unknown", "Trusted", "Critical"\}$. A user has the rank *"Unknown"* only when joining the network for the first time, i.e. once the rank is changed from *"Unknown"*, it can never have this value again. Changes in rank happen either through disputes, where the rank of the developer can change to *"Critical"* in case of loss or remain the same in case of win, or by promotions, where the rank of the developer can change to *"Trusted"*.

Promotions depend on the number of transactions in each of the developer's apps compared to the number of transactions in other popular apps. Promotions are automatic and depend on the following condition:

$$\sum_{t=0}^{T}\sum_{i=1}^{N} TX_{A_{ij},t} \geq TX_{A_M^T} \tag{1}$$

Where $A_i = \{A_{i1}..A_{iN}\}$ is the set of N apps of developer D_i, meaning that A_{ij} is the app j in the set A_i, t is the day with $t = 0$ being the moment the developer joined the network, T is the current day, A_M^T is the app with the highest number of transactions on day T. Given these variables definitions, one can see that the left side in Equation 1 represents the amount of transactions in all the apps of developer D_i since he joined the network and the right side represents the number of transactions of the app with the highest number of transactions on day T.

Contrary to the promotions, disputes are not done automatically and require explicit actions from users. Any user can open a dispute with a developer stating that the developer is dishonest. After the dispute is opened, any other user can join either side, depending on if they want to support the accusation of dishonesty or if they want to defend the developer.

Additionally, each rank value has a level associated with it. For *"Unknown"* and *"Critical"* rank values, the level is always set to 1. When the rank is *"Trusted"*, we do not set a maximum level and it is expressed by:

$$S_l = \log_2 \frac{\sum_{t=0}^{T} \sum_{i=1}^{N} TX_{A_{ij},t}}{TX_{A_M^T}} \tag{2}$$

Because of the use of the logarithm function in Equation 2, it becomes harder to gain higher rank levels as the rank level increases.

3.3.1 Developer Reputation

It is assumed that a known developer is more trustworthy than a developer who recently joined the apps distribution business.

3.3.2 Data Structures

Dispute Intent. A *dispute intent* happens when a user claims that a developer is dishonest and no dispute against that developer is open. The developer or any other user then has 7 days to answer the dispute. If someone answers the dispute, be it the developer or any other user, the dispute is opened and the minimum fees needed to open it are captivated. If no user answers the dispute, the *dispute intent* closes and the developer status changes to *critical*.

Dispute. A *dispute* is a conflict between two parties, where one party - the *contestants* - claim that a developer is dishonest, i.e. the developer uploads apps with malware, too many ads, or non-working apps and the other party - the *pleaders* - claim the developer is honest. The pleaders include the developer being accused of dishonesty by the contestants. Both parties place tokens in the dispute and the party holding the most amount of tokens by the end of the dispute wins. Therefore, the *dispute* includes the developer it regards to, the participants in both parties and their respective stake in the dispute.

DisputeLedger. The *dispute ledger* is the record of users joining disputes. It stores that a user joined a dispute on behalf of one of the sides with a certain stake (amount of tokens). The entries in the *dispute ledger* are used to settle disputes when they end.

RankLedger. The *rank ledger* is the record of rank changes of developers. Whenever there is a change in a developer's rank, be it an increase in the *"Trusted"* rank levels or a change to *"Critical"*, it is recorded in the *rank ledger*.

Data Structures
DisputeIntent dispute intent $K_x^0 := \langle D_i, C_j, T_{min}, S \rangle$ • Developer D_i, the developer being accused of being dishonest • Contestant C_j, user claiming developer D_i is dishonest • Minimum fee T_{min}, the minimum fee needed to open the dispute that may result from this dispute intent • status S, the current status of the dispute intent, which can take the values of *"Open"* or *"Closed"* **Dispute** dispute $K_x := \langle D_i, S, T_{min} \rangle$ • Developer D_i, the developer being accused of being dishonest • status S, the current status of the dispute, which can take the values of *"Open"* or *"Closed"* • Minimum fee T_{min}, the minimum fee needed to open the dispute **Dispute Ledger** dispute ledger $L_K : (E_1..E_n)$ • entry E_i, an entry containing information about a user joining a dispute in the form $E_i := \langle U_i, P, T, K_x \rangle$, where U_i is the user, P is the position the user is taking (can be either *"Contestants"* or *"Pleaders"*), T is the stake (amount of tokens) the user U_i is willing to use to defend position P and K_x is the dispute user U_i is joining **Rank Ledger** rank ledger $L_R : (E_1..E_n)$ • entry E_i, an entry containing information about a change in a developer's rank in the form $E_i := \langle D_i, S_b, S_a, S_l \rangle$, where D_i is the developer, S_b is the developer's rank before the change, S_a is the rank after the change and S_l is the level of the rank. S_b can take the values of $\{"Unknown", "Trusted"\}$ and S_a can take the values of $\{"Trusted", "Critical"\}$. For further details regarding the possible states of S_b and S_a and the possible values of S_l, please refer to Figure **[INCLUDE REF TO FIG]**.

Table 6: Data Structures for Developers Rank Use Case

3.3.3 Algorithms' Pseudo-code

Create dispute intent. When a user C_i claims a developer D_j is dishonest, an intent of dispute is created, which may result in a dispute being opened, depending on whether someone answers the *dispute intent* within 7 days or not. The user answering the dispute may not be the developer.

DR.CreateDisputeIntent

- INPUTS:
  - User C_z
  - Developer D_j
- OUTPUTS: Dispute intent K_x^0

Create dispute. When a dispute intent is answered, a disputed is created. Within the following 30 days, any user may join the contestants side, which is composed by users claiming the developer D_j is dishonest, or the pleaders side, which contains users stating that the developer D_j is honest. The dispute intent that originated the new dispute is closed.

DR.CreateDispute

- INPUTS:
  - Developer D_j
  - Minimum fee T_{min}
  - Dispute intent K_x^0
- OUTPUTS: Dispute K_x

Close dispute. When the dispute is over (after 30 days), the winning side has its stakes refunded, while also receiving 10% of each pledge from the losing side, with each winning member getting a winning stake proportional to their stake in the overall winning side pledge. Each member from the losing side gets a refund from the respective pledge subtracted by 10%. Please refer to Table 7 for more details.

DR.CloseDispute

- INPUTS:
  - Dispute K_x
- OUTPUTS: None

Compute rank. The rank of the developer is periodically checked to assess changes in its value. For example, the rank can change from *"Unknown"* to *"Trusted"* or the *"Trusted"* level can increase.

DR.ComputeRank

- INPUTS:
  - Developer D_i
  - Set of transactions in developer's apps TX_i
  - Set of developer's apps A_i
  - App with the highest amount of transactions A_M
- OUTPUTS: None

Record dispute. When a user joins a side on a dispute, the event is recorded in the dispute ledger and can later be used to settle the dispute.

DR.RecordDispute

- INPUTS:
  - User U_i
  - Position P
  - Stake T
  - Dispute K_x
- OUTPUTS: None

Record rank change. When there is a developer's rank change, it is recorded in the rank ledger.

DR.RecordRank

- INPUTS:
  - Developer D_i
  - New rank S_a
  - New rank level S_l
- OUTPUTS: None

Developers' Rank Use Case

DR.CreateDisputeIntent

- INPUTS:
  - User C_z
  - Developer D_j
- OUTPUTS: Dispute intent K_x^0

1. Compute K_x^0 := DisputeIntent(D_j, C_z)
2. Set K_x^0.Status := *"Open"*

DR.CreateDispute

- INPUTS:
  - Developer D_j
  - Minimum fee T_{min}
  - Dispute intent K_x^0
- OUTPUTS: Dispute K_x

1. Set K_x^0.Status := *"Closed"*
2. Compute K_x := Dispute(D_j, T_{min})
3. Set K_x.Status := *"Open"*

DR.CloseDispute

- INPUTS:
  - Dispute K_x
- OUTPUTS: None

1. Compute $WinSide$:= WinningSide(K_x)
2. Compute DistributePledges(K_x)
3. Set K_x.Status := *"Closed"*
4. If $WinSide$ = *"Contestants"*:
   (a) Compute DR.RecordRank(K_x.D, *"Critical"*, 1)

DR.ComputeRank

- INPUTS:
  - Developer D_i
  - Set of transactions in developer's apps TX_i
  - Set of developer's apps A_i
  - App with the highest amount of transactions A_M
- OUTPUTS: None

1. Compute S_l := RankLevel(TX_i, A_M)
2. If $S_l \geq 1$ and D_i.Rank = *"Unknown"*:
   (a) Compute DR.RecordRank(D_i, *"Trusted"*, S_l)
   (b) Return
3. If $S_l > D_i$.RankLevel and D_i.Rank = *"Trusted"*:
   (a) Compute DR.RecordRank(D_i, *"Trusted"*, S_l)
   (b) Return

DR.RecordDispute

- INPUTS:
  - User U_i
  - Position P
  - Stake T
  - Dispute K_x
- OUTPUTS: None

1. Send T from user U_i wallet W_{U_i} to dispute's wallet W_{K_x}
2. Set $E := \langle U_i, P, T, K_x \rangle$
3. Compute WriteDisputeLedger(E)

DR.RecordRank

- INPUTS:
  - Developer D_i
  - New rank S_a
  - New rank level S_l
- OUTPUTS: None

1. Set $S_b := D_i$.Rank
2. Set $E := \langle D_i, S_b, S_a, S_l \rangle$
3. Compute WriteRankLedger(E)

Table 7: Developers Rank Use Case

3.3.4 Wallet Transactions

The reputation of a developer is built based on blockchain transactions that can be associated to him. However, when there is a dispute, as presented in the previous section, the dispute mechanism is solved by receiving positive / negative endorsements of the community members to the developer. The final decision is based on the total sum of tokens that each side has. Figure 10 shows how users create disputes and join sides by placing tokens on the dispute.

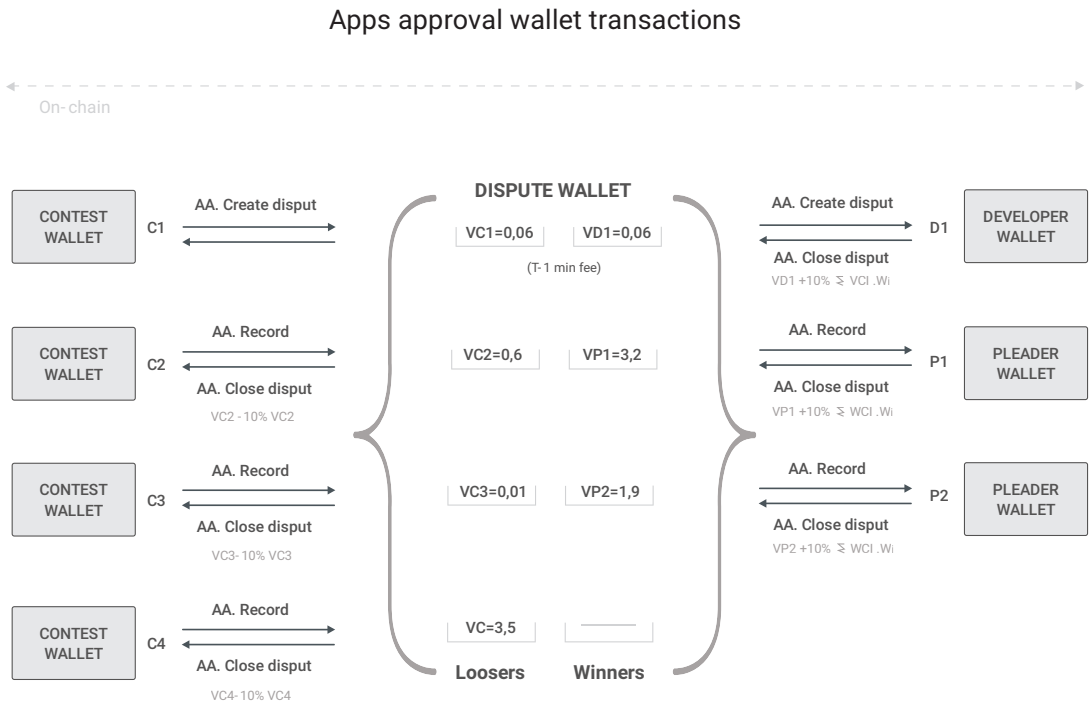

Figure 10: Wallet transfers in the Developers Approval process.

4 Blockchain Limitations and Proposed Approach

With the current position of the blockchain industry, business use cases are reliant on technology supported by Bitcoin and Ethereum, which can delay the development of projects. There are three use cases which the AppCoin protocol tries to solve in order to enable a fully working AppCoin economy:

- Apps advertising in app store
- In-App Billing
- Developer reputation leading to approval of their apps

From the aforementioned use cases a list of requirements on the technology can be established.

- Store digital currency / value securely
- Couple value storage and their transfers with logic
- Scalability for millions of users

The first requirement is solved by using a blockchain technology. Either Bitcoin or Ethereum work for this. For the second requirement only a type of blockchain technology supporting smart contracts will work, i.e. Ethereum, Nxt or Tezos.

Unfortunately, neither basic Ethereum nor any other current blockchain for that matter are able to scale for the needs of an app marketplace. This problem is worsened further considering a platform that should potentially work for all the app marketplaces.

First let us have a detailed look at the challenges in question.

4.1 Blockchain limits

Let us imagine a scenario of a working AppCoin economy using a blockchain technology like Ethereum. At the present there are about 2 billion Android users. The average Ethereum transaction is around 160 bytes [7]. If we wanted to create an AppCoin economy directly on Ethereum using these numbers, we would get a system with the characteristics shown in Table 8.

Users	2 000 000 000
Avg. daily TX	1
Avg. TX size	160 bytes
Daily TX	2 000 000 000
TX per second	23 148
Daily TX size	298.02 GB
TX size per second	3.53 MB
TX size per month	8.73 TB
TX size per year	104.77 TB

Table 8: Scalability requirements for AppCoin economy

Transaction Volume

Scalability in terms of volume is the first problem of Ethereum. At present it handles only up to 20 transactions per second [41]. That is nowhere near the required 23 000 average transactions per second. At peak times the system should support a number of transactions several orders of magnitudes higher; for example VISA normally handles around 1667 transactions per second and at peak times it claims it can complete around 56 000 transactions per second [41].

Fees

The topic of fees is closely connected to the possibility to support micro-transactions - transfers of value in the range of a few cents. At the time of writing the cost for one transaction in Ethereum is around 10 cents of a US Dollar for a transaction that takes around 1 minute to execute [8]. If the user is willing to wait around 10 minutes, the cost is 1 cent. This is both too slow and too expensive to facilitate micro-transactions.

Latency

As mentioned in the previous section, in Ethereum, one has to wait until a transaction is part of the blockchain. To have a reasonable guarantee that the transaction is part of the main blockchain and not in one of a number of possible branches, one needs to wait until 7 blocks have been created. This is because two miners could submit the next block at the same time and it will take some time to decide whose block will continue the chain. While this might not be an issue for high value money transfers in the AppCoin economy the transactions have to be executed within few seconds or faster (preferably instantly) in order to enable meaningful IAB and user experience.

4.2 Existing technology

Scalability restrictions are present in any blockchain technology using mining by *proof-of-work*, such as Bitcoin and Ethereum. For Bitcoin - the oldest blockchain implementation - the proposed solution is the Lightning Network [32].

4.3 Ethereum and Bitcoin based

Effectively there is one possibility for how to tackle the scalability problem with Bitcoin and two solutions with Ethereum. The first one is keeping the proof-of-work based blockchain and enhancing it with a direct-channel-payment solution like the Lightning Network. For Ethereum there are two major technologies that are going to be introduced also in this section.

4.4 Lightning Network

In brief the Lightning Network allows for creation of bidirectional payment channels that are off-chain [32]. Using such a channel two parties agree to deposit money into a common entity called a channel and this information is stored on the blockchain. There is a 2-signature entry for the channel with the total sum of the deposit saved. From this moment on the two parties can exchange between them any number of payments in the form of signed receipts. The only condition is that the balance for one party doesn't exceed the total of the deposit. Should a case like this happen, the parties can increase the deposit. These off-chain transactions are signed and sent direct. They are not broadcast. In contrast all the transactions on the main blockchain need to be broadcast to all participants.

This approach solves the blockchain scalability problem, such as:

- The messages are direct and not broadcast. The volume of transactions between the two parties is only limited by their processing capabilities. Also, compared to blockchain, not all the transactions have to be stored but only the latest receipts, i.e. the latest balance

for both users, must be stored instead of all transactions. This solves the problem of storing huge amounts of old transactional data that is inherent in blockchain.

- The only fees are for opening and closing the payment channel on the blockchain. For payments over the payment channel there are no extra fees. Once it is established micro transactions are possible without further limitations.
- Finally by using direct peer-to-peer (P2P) communication, transactions can be exchanged as fast as the underlying network between the two parties will enable it. Thus the problem of latency is solved.

Thus the original blockchain scalability limitations are resolved for the case of two directly connected parties. Of course having open channels between all the participants in the AppCoin economy is both not feasible and economically viable (because of opening and settling fees on the blockchain). Therefore the Lightning Network proposes a solution, where the payment channels are stored in a graph and payment can be routed securely keeping the above mentioned characteristics among multiple parties. This is enabled using hashlocks. For more detailed information, please have a look at either the specification of Lightning Network or the excellent introductory article [38].

4.4.1 Raiden

Raiden network is an early implementation of the Lightning Network protocol for Ethereum. Therefore, it is an off-chain scaling solution for Ethereum. It promises near-instant, low-fee and scalable payments and enables micro transactions [35]. Raiden runs as a network of nodes establishing payment channels among users. It uses locked funds in a smart contract in Ethereum and payments happen inside of Raiden until one party chooses to settle. Raiden will work for any Ethereum tokens implementing the ERC20 standard.

One of the key characteristics of Raiden is its maturity. Currently it is the only known technology for Ethereum with existing implementation (i.e. beyond white paper stage) enabling a working App Coin economy. It has been under development for the past two years and it is reaching a Minimal Viable Product stage, currently being in a developer preview stage.

Privacy is another key characteristic of Raiden. All the transactions are known only to the involved parties and not public knowledge like in Ethereum.

Finally the maturity can also be seen as a weak point for Raiden. Even though it is beyond proof-of-concept stage it is not recommended to be used on the live Ethereum network by its creators.

4.4.2 Plasma

Plasma is defined as a platform for "scalable autonomous smart contracts" [31] and it is another attempt to bring the Lightning Network technology from Bitcoin to the Ethereum world. This is achieved by means of a tree of hierarchical chains where most of the transactions can be settled in the sub-chains and do not need to go to the expensive and slow main chain. The

main chain will be needed only to settle disputes coming from sub chains and would serve as final validation.

A strong point for Plasma is that it is backed by key people from both Ethereum and Lightning Network, i.e. Vitalik Buterin and Joseph Poon. Another aspect supporting Plasma is that it has been endorsed by OmiseGO to be used for their decentralised exchange and payment platform[29]. What speaks against Plasma at this time is the fact that the project has only a white paper, without any working code base yet.

4.4.3 OmiseGO

OmiseGO is an Asian payment gateway. It plans to use Ethereum and Plasma as a foundation for creating a decentralised exchange and payment platform [20] and the company intends to launch its own blockchain called OMG that will be complementary to Ethereum. Ether will be then used as a medium of interchange for various assets being exchanged on this blockchain.

Here are some of the strong points of OmiseGO:

- Consensus rules for high performance activity;
- Rapid execution and clearing. Proof-of-Stake;
- Ability to handle extremely high volumes of transactions with final delivery in Ethereum;
- Availability as white label e-Wallet solution

The weak points include:

- The project is at a very early stage. It was only announced at the beginning of August 2017. Currently, there is only a white paper available.
- As a result of the coupling with Ethereum, the OmiseGO white paper recommends to validate transactions simultaneously on the Ethereum blockchain for maximum security.

4.5 Independent blockchains

The second solution to achieve global consensus on scalability for blockchain is the use of a proof-of-stake as a block minting algorithm. For Ethereum this would be the Casper protocol which has been commonly talked about in the past years in the Ethereum community but is not implemented yet. As Casper is not in use yet we will examine here other projects that have implemented minting instead of mining blocks. A brief look will be also given to IOTA, which solves the scalability problem with a directed acyclic graph called Tangle instead of blockchain.

4.5.1 Nxt

One of solutions not based upon Ethereum is Nxt. It is described as "an open source cryptocurrency and payment network launched in November 2013 by anonymous software developer BCNext" [43]. Compared to Ethereum one can see Nxt as a monolith trying to include major features from all coins. It describes itself as an modern economic system based on cryptography

and blockchain technology. On the other hand Ethereum built a low platform and programming language in which one can program generic contracts. Other major differentiation of Nxt to Ethereum is the fact that it is based on proof-of-stake for creating new blocks.

Nxt is unsuitable for creating an App Coin Economy as it targets a different scenario. Focusing on managing a business, assets and customers and interaction between them are recorded in a secure way instead of a blockchain backed technology enabling users to create generic decentralised apps. Another point against Nxt is the fact that its beginning was obscure. 73 anonymous accounts received stakes in exchange for Bitcoin accounts and it is these people who can mint new blocks. And also that there has not been much buzz around and development for Nxt recently.

4.5.2 Tezos

Compared to Nxt the next examined project has received a great deal of attention recently and finished its ICO having raised more than $200 million.

Tezos describes its ultimate aim to create blockchain technology working better than both Ethereum and Bitcoin. It wants to provide users with financial incentives for maintaining on-chain consensus mechanism. The main differences to Ethereum are:

- Built-in consensus mechanism for governance.
- Decoupled from Ethereum. Separate blockchain with proof-of-stake consensus for block minting.
- OCaml as programming language for creating smart contracts.
- Design-wise Ethereum is a generic and low-level solution while Tezos aims to provide a fat protocol with a lot of features [5], not unlike Nxt.

4.5.3 IOTA

The goal of IOTA is to create a lightweight network that allows a machine economy. A machine economy is a situation where IoT (Internet of Things) connected devices communicate and execute payments between each other. IOTA does not use blockchain but a directed acyclic graph called Tangle. Currently it is in Beta stage with a reference implementation. One of the major advertised features is that there are no fees. In Tangle, each new transaction confirms at least 2 previous transactions; that is the "fee" to participate. This might delay the transaction, especially if the network is small. There are no miners. The IOTA white paper states it is quantum proof (invulnerable to encryption-breaking attacks), that it has a high level of supply and the tokens are not divisible, unlike Bitcoins or Ether. For a primer on IOTA see [36].

4.6 Proposed Approach

As seen in the previous section, there are blockchain currencies that can securely store digital value. But as we have seen, doing transactions over them does not scale, is slow and requires fees. Therefore alternatives like direct payment channels have to be evaluated.

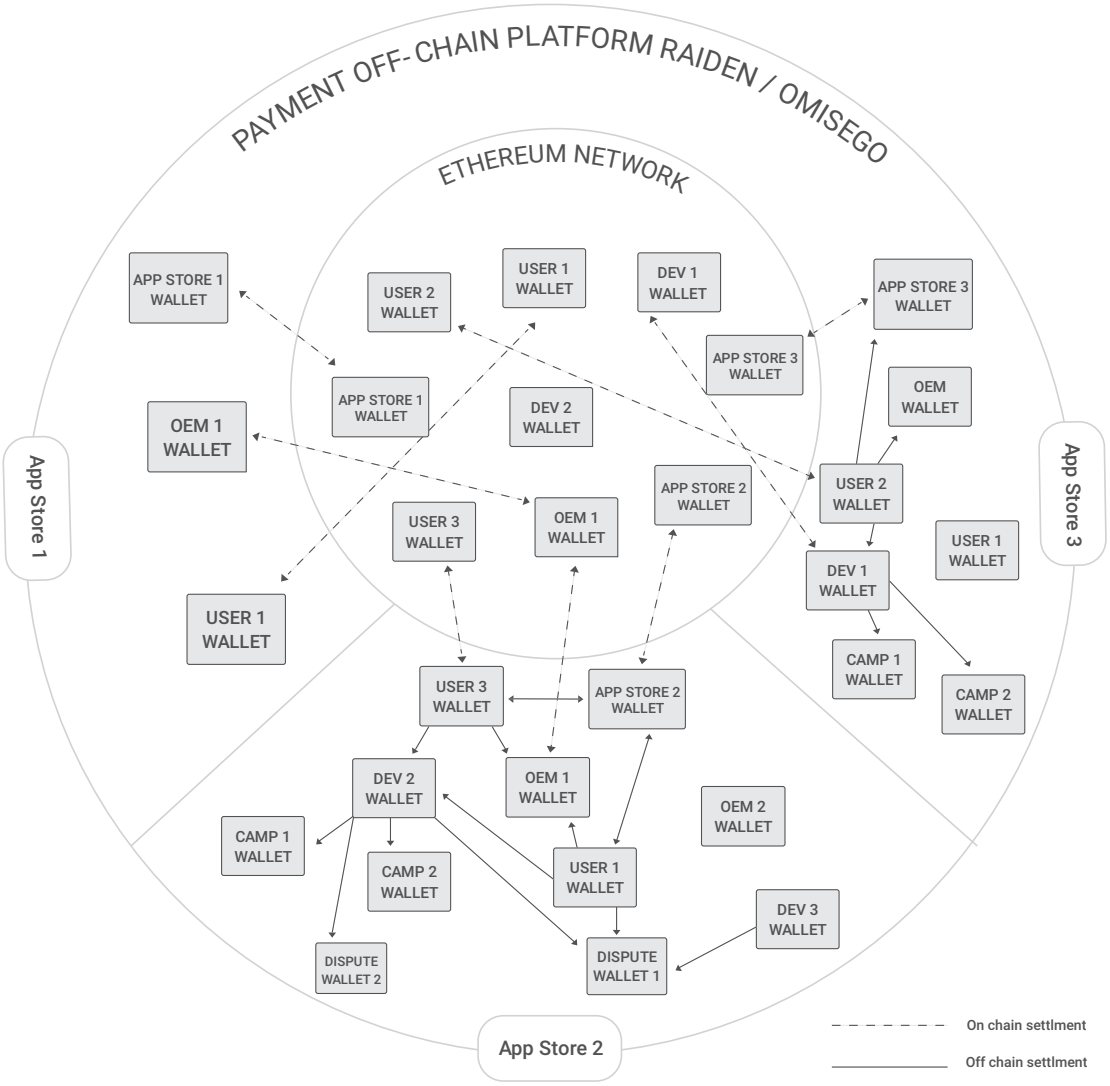

Figure 11: On-chain and off-chain transactions.

In general terms, the App Store acts as the gateway to the digital payment world for the user. He deposits his valuables (preferably Ether) with the store in a payment channel using a smart contract. The same happens for developers. And from this moment on they can utilise the micro-payments within the App Infrastructure. The payments are instant, cheap and there can be any number of them.

The user has to pay fees on each Ether deposit to his account with the App Store and on checking out, i.e. settling his payment channel. Further fees may be the taxation done by the App Store used to pay for its infrastructure. If there are several App Stores in the same ecosystem like we envision, there might be further (minimal) fees to forward the payments from one to another.

Figure 11 depicts a schema of the wallets on-chain (Ethereum network) and the wallets off-chain using Raiden or OmiseGO.

Returning to the requirements from the beginning of this chapter, we plan to use Ethereum for storing value and smart contracts. To have scalability, Raiden is being evaluated for the first Beta version of the reference implementation of the protocol that is due six months after the ICO.

The choice for Raiden is because - at the time of writing - it is the only platform that has a test network and can be used. Yet OmiseGO has a promising outlook and in the future could be adopted depending on its progress. The company will be kept under close monitoring as it might develop the potential to substitute Raiden.

5 Related Work

This section presents the projects that inspired the AppCoins protocol, either because of the technology employed or by presenting concepts that empower the protocol. We first give a brief overview of each project and after that we explain how each of our use cases benefit from the contributions of each of these projects.

5.1 Related Projects

5.1.1 Basic Attention Token

The BAT project aims to revolutionise the digital advertising landscape by proposing a "decentralised, transparent digital ad exchange based on Blockchain" [4]. Their proposal has two main components:

- **Brave**: a browser that blocks third-party ads and trackers, which decreases webpages load time and ensures anonymity, while also building a ledger system that tracks users' attention to ads in order to correctly reward publishers and advertisers
- **BAT**: a token for the decentralised ad exchange connecting advertisers, publishers and users, while rewarding users for their attention.

In their proposal, BAT wants to eliminate the middlemen between advertisers and publishers, pay for user attention instead of CPM/clicks, provide faster webpage loads and ads tuned to user preferences, amongst other features.

By taking out the middlemen, BAT avoids draining of resources by agencies, DSPs, exchanges, ad networks and others, while also eliminating part of the complexity of having to deal with this huge ecosystem that is in place today. The gain in resources by eliminating the resource-draining players enables the sharing of resources by the fundamental, value-generating players in the flow: advertisers, publishers and users. Since there are fewer resources being wasted in the middlemen, there are more available to be employed in processes that increase the value to the end user. BAT also proposes to use machine learning at the browser level to serve tailored ads to users, instead of serving ads with questionable value. In addition, users

are rewarded by their attention, which the project states as a "rare quantity". This statement comes from the fact that information available to users is far greater that the attention each user has to give.

Today, publishers are paid based on clicks on ads. BAT proposes to start rewarding publishers based on the attention users give to ads, by keeping track of the user attention on a ledger system implemented in Brave, while always maintaining users' anonymity. User attention - a very valuable asset - is not being rewarded correctly and users do not get anything while they navigate webpages and see the ads. The solution proposes that users also start receiving rewards for time spent seeing the ads while navigating, based on the amount of time they spend looking at them.

In order to reduce fraud, they propose approaches - calling them Basic Attention Metrics (BAM) - to correctly identify users paying attention to ads. When the user attention is identified, it is saved in an anonymous way. At the same time BAT also ensures that users do not get rewarded by paying attention to the same ad more than once. They define a *proof-of-attention* algorithm, which ensures that a user paid attention for a certain number of minutes to an ad and is verifiable by anyone in the network. Using ANONIZE [14], BAT ensures that a user can only see and get attributed to an ad once and users' anonymity is maintained. According to the authors, the algorithm is "the first implementation of a provably-secure multiparty protocol that scales to handle millions of users". The BAT team says that they may also invest into using algorithms such as BOLT [27], zkSNARKs [13] and STARKs [2] to protect users' privacy.

5.1.2 Kin

The Kin project intends to create the "first open and sustainable alternative ecosystem of digital services for our daily lives" [19]. In order to achieve this, a new cryptocurrency Kin is created to be used within this ecosystem of digital services.

Since Kin is being developed by Kik, a popular chat app with already millions of users, Kik will integrate Kin to showcase the possibilities of having an ecosystem of connected digital services. New partners joining the ecosystem will create a network effected, boosting the value of Kin. Kik will develop two main components of the new ecosystem:

- Kin Reward Engine
- Kin Foundation

The Kin Reward Engine is going to create incentives for other digital services to adopt Kin. The majority of the Kin supply will be allocated to Kin Reward Engine and periodically will unlock and distribute a certain amount of Kin amongst the digital services within the ecosystem. The amount each digital service receives depends on the amount of Kin used by them.

Kin Foundation is the entity that will oversee the growth of the ecosystem, as well as administer the Kin Reward Engine. In time, the Kin Foundation will transition the entire ecosystem, including the Kin Reward Engine to a fully decentralised and autonomous network. When this happens, the Kin Foundation's main responsibilities will be helping onboarding

new partners and overseeing development of fundamental components such as identity and reputation management, cryptocurrency wallets and compliance solutions.

In the end, Kin wants to develop an ecosystem that is open and fair, where users benefit from a vast and diverse digital experience, being able to transition between services with almost no effort. Providers will be able to compete for compensation within the ecosystem.

5.1.3 Monetha

The Monetha project aims to change how e-commerce is done and how merchants can reach customers by "creating a universal decentralised trust and reputation solution working flawlessly together with mobile payments processing on the Ethereum blockchain leveraging smart contract technology" [28].

Currently, merchants wanting to sell online face two options: creating their own website or joining the big marketplaces, such as Amazon, Ebay, Alibaba, etc. The former requires a very significant investment in brand creation and advertising in order to have customers going directly to their site. This is due to the fact that customers tend to buy on places they trust and that have good reputation. If there is a merchant with a website no-one has heard about, there will be a trust barrier for the customers. On the other hand, joining big marketplaces has the advantage of not needing much advertising and branding but there is still the need to create reputation and trust amongst customers. Merchants accomplish this by providing high quality products delivered within the agreed time and, in turn, receive positive reviews. The problem with this approach is that the reputation a merchant is able to build on one marketplace is not transferable to others. If a merchant wants to sell on several marketplaces, since they are all disconnected, the merchant needs to build a reputation amongst customers on all of them.

In addition to the reputation problem, big marketplaces also impose very complex transaction processes. The transaction settlements are troublesome for the customer due to the high number of steps required, which Monetha states that can go up to 16. An additional problem is the high transaction fees, both regarding the number of individual fee types, which Monetha claims can go as high as 15, and the amount needed to be paid by the merchant. High fees can deter a merchant with small margins to sell products in the marketplaces, leaving the merchant with the option to create a brand and website, which is also an expensive option as explained above. Not only do merchants need to pay high fees but there is also long transaction times between the marketplace and the merchant. Since there are so many parties involved in a transaction, the settlement can take up to 3 days, or a week for international payments. Moreover, marketplaces often hold payments for a week because of the high probability of chargebacks.

Monetha proposes a solution composed of a decentralised trust and reputation system together with payments powered by blockchain technology using Ethereum. Instead of having marketplaces with their own reputation system, Monetha proposes that merchants in the network build their reputation from transactions, claims and reviews. Furthermore, since the network is shared by all merchants, it is as if the network would be a huge marketplace where any customer can see the reputation and trust score of every merchant and vice-versa. Instead of having to build reputation in several different systems, their reputation is built automatically and available to everyone.

Since payments are done through Ethereum, settlements are much simpler and the connection is direct between the merchant and the customer. With fewer steps in the settlement comes the advantage of smaller settlement times, with the merchant getting the money within just a few minutes. Fees are also much smaller, with a fixed transaction fee of of 1.5%. Finally, since there is no centralised marketplace, there is also no holding of payments.

5.2 Projects Affinity

5.2.1 Advertising

The Advertising use case, as described in Section 1, states that in app stores there are too many resource-draining middlemen and the risk of fraud is too high because of the complexity of the systems in place today. As we propose in Section 2, we want to create a decentralised network where any App Store can join and where users can have a seamless App Store experience, as well as provide developers with the tools to directly reach App Stores and the users they want by creating advertising campaigns in a leaner and transparent way without having to go through DSPs, Ad networks, etc.

The concept of having campaigns directed to a set of anonymous users that can only be attributed once, with a concrete and verifiable proof that the user did indeed complete the action required (i.e. opened the app for at least 2 minutes), closely relates to BAT's concept of user paid attention proven with a *proof-of-attention*.

Furthermore, we expect that campaigns can reach millions of attributions, meaning that there will be millions of transactions between developers who create campaigns and users, app stores and OEMs. In our solution, we propose to be able to do this very quickly, without holding funds from developers. In order to achieve this, we need to overcome the known problems of current blockchain technology of scalability and high transaction fees, as the transaction in advertising campaigns are categorised as micro-transactions, which means that current fees are too high compared to the actual transaction value. BAT also address this problem but relies on probabilistic payments [37, 12] instead of referring to solutions as Raiden or Plasma.

5.2.2 IAB

IAB is a use case relying heavily on micro-transactions. Most in-app purchases are small purchases completed several times potentially by millions of users. If users have to pay fees in the same order of magnitude of the transaction, they will not rely on the system to pay for in-app items

BAT project's micro-transactions, as referred to in the previous section, are done using probabilistic payments. Neither Kin nor Monetha projects specify how they want to enable efficient micro-transactions.

We look for projects like Raiden and Plasma for a solution to this problem, as they also offer solutions for blockchain scalability and transaction times.

5.2.3 Developer Reputation

In this use case, we define the need to have a better method for developer evaluation, which avoids centralisation in the decision-making process in the App Stores, which can introduce censorship for dubious reasons or anti-competitive actions. In addition, today there is no knowledge-sharing between app store with respect to which developers are honest and provide good apps to users and developers providing apps with malware or too many ads, or just non-functional apps. The protocol proposes a way to compute the reputation of developers by the transactions that occur on their apps (from advertising campaigns or in-app purchases) and by claims from users.

Monetha proposes a similar approach to compute reputation of merchants and customers based on transactions, claims and reviews from users. Although the company does not specify how each of these actions modifies the reputation, they say that some actions have more relevance than others and introduce ways to motivate merchants and customers to be honest.

Kin briefly mentions that the network will be responsible for regulating itself, i.e. digital services providers will be managed automatically by the network with regards to the quality of their service. Kin does not mention how to achieve this but the concept of having the network automatically regulating its players empowered our concept.

6 Acknowledgements

The AppCoins protocol had the contributions of several people. Our apologies if we have unfairly left someone out of the deserved credits.

The client-side support in Section 2.2 was a contribution of Marcelo Benites. The app approval detailed in Section 1.5 was written by João Carneiro. The current blockchain limitations presented in Section 4 was contributed by Martin Uzak. Matthew Boyle reviewed the document for language inconsistencies. The rest of Aptoide team provided important feedback and ideas that were integrated in the protocol and in the document.

The following people provided much appreciated suggestions and corrections: Jonathan Becker and Thomas Gieselmann (e.Ventures), Jun Hasegawa (OmiseGO), and Alexander Maier (ICO Advisories).

References

[1] Augustine, A. 5 Amazing Apps Banned from Play Store. `http://techlomedia.in/2017/01/5-amazing-apps-banned-play-store-46238/`.

[2] Ben-Tov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., BenSasson, E., and Virza, M. Computational integrity with a public random string from quasi-linear pcps. In *EUROCRYPT 2017 (36th International Conference on the Theory and Applications of Cryptographic Techniques)* (2017).

[3] Bhattacharya, A., and Goswami, R. T. *Comparative Analysis of Different Feature Ranking Techniques in Data Mining-Based Android Malware Detection.* Springer Singapore, Singapore, 2017, pp. 39–49.

[4] Brave. Basic attention token - blockchain based digital advertising. White paper, 2016.

[5] Breitman, A. White paper, 2017. [Online; accessed 21-September-2017].

[6] ConnerForrest. Android malware bypassed Google Play store security, could have infected 4.2 million devices. `http://www.techrepublic.com/article/android-malware-bypassed-google-play-store-security-could-have-infected-4-2-devices/`. [Accessed: 2017-09-18].

[7] Etherscan.io. Ethereum Transaction Chart. `https://etherscan.io/chart/tx`.

[8] ethgasstation.info. ETH Gas Station. `http://ethgasstation.info/`.

[9] Google. Google Play Developer Distribution Agreement. `https://play.google.com/intl/en_us/about/developer-distribution-agreement.html`. [Accessed: 2017-09-29].

[10] Google. Let's build the world's most trusted source for apps and games. `https://play.google.com/about/developer-content-policy/`.

[11] Google. How we keep harmful apps out of Google Play and keep your Android device safe. White Paper, 2016.

[12] Green, M. D., Liu, J., Miers, I., Miao, P., Mishra, P., and Chiesa, A. Decentralized anonymous micropayments. In *EUROCRYPT 2017 (36th International Conference on the Theory and Applications of Cryptographic Techniques)* (2017).

[13] Groth, J. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT* (2010), pp. 321–340.

[14] Hohenberger, S., Myers, S., Pass, R., and abhi shelat. Anonize - a large-scale anonymous survey system. White paper, 2015.

[15] Hughes, N. Apple's App Store approval process gets partially automated. `http://appleinsider.com/articles/09/11/20/apples_app_store_approval_process_gets_partially_automated`. [Accessed: 2017-09-20].

[16] Inc., A. App Review. `https://developer.apple.com/support/app-review/`. [Accessed: 2017-09-20].

[17] Inc., A. App Store Review Guidelines. `https://developer.apple.com/app-store/review/guidelines/`. [Accessed: 2017-09-20].

[18] Inc., A. Common App Rejections. `https://developer.apple.com/app-store/review/rejections/`. [Accessed: 2017-09-20].

[19] Inc., K. I. Kin: a decentralized ecosystem of digital services for daily life. White paper, 2017.

[20] Joseph Poon, O. T. White paper, 2017.

[21] Kafka, P. Spotify says Apple won't approve a new version of its app because it doesn't want competition for Apple Music. https://www.recode.net/2016/6/30/12067578/spotify-apple-app-store-rejection.

[22] King, A. New Android malware infected 21 million devices via Google Play apps. http://phandroid.com/2017/09/18/android-malware-google-play/.

[23] Kokalitcheva, K. Apple's App Approval Process Just Got A Lot Faster. http://fortune.com/2016/05/12/apple-app-store-faster-approval-2/.

[24] Lee, T. B. Twitter rival Gab sues Google over app store rejection. https://arstechnica.com/tech-policy/2017/09/twitter-rival-gab-sues-google-over-app-store-rejection/.

[25] Lin, J. How to Make $80,000 Per Month on the Apple App Store. https://medium.com/@johnnylin/how-to-make-80-000-per-month-on-the-apple-app-store-bdb943862e88.

[26] Lucas, G. [...] Races home aboard her starship, custodian of the stolen plans that can save her people and restore freedom to the galaxy..., 1977. Easter Egg Editions.

[27] Miers, I., and Green, M. Bolt: Anonymous payment channels for decentralized currencies. In *IACR Cryptology ePrint Archive* (2016).

[28] Monetha. Monetha. White paper, 2017.

[29] natra. OmiseGo is the first plasma.io project! https://steemit.com/bitcoin/@natra/omisego-is-the-first-plasma-io-project, 2017. [Accessed: 2017-08-20].

[30] Palmer, D. Over 500 Android apps with a combined 100 million downloads found to secretly contain spyware. http://www.zdnet.com/article/500-android-apps-found-to-secretly-contain-data-stealing-spyware/.

[31] Poon, J., and Buterin, V. Plasma: Scalable Autonomous Smart Contracts. White paper, 2017.

[32] Poon, J., and Dryja, T. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. White paper, 2016.

[33] PYMNTS. Spotify, Others Claim Apple, Google Anticompetitive. https://www.pymnts.com/news/regulation/2017/spotify-others-claim-apple-google-anticompetitive-european-commission/.

[34] Rai, S. Apple's Refusal to Approve India's Anti-Spam App Angers Regulators. https://www.bloomberg.com/news/articles/2017-09-06/apple-refusal-to-approve-india-spam-app-antagonizes-regulator.

[35] Raiden. White paper. [Online; accessed 21-September-2017].

[36] Schiener, D. White paper, 2017. [Online; accessed 21-September-2017].

[37] Shelat, A., and Pass, R. Micropayments for decentralized currencies. In *CCS '15: Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 207–218.

[38] Stark, E. What is the Lightning Network and how can it help Bitcoin scale? https://coincenter.org/entry/what-is-the-lightning-network, 2015. [Accessed: 2017-08-20].

[39] Surowiecki, J. *The Wisdom of Crowds*. Anchor, 2005.

[40] Talbot, D. Remotely Assembled Malware Blows Past Apple's Screening Process. https://www.technologyreview.com/s/518096/remotely-assembled-malware-blows-past-apples-screening-process/.

[41] Vermeulen, J. Bitcoin and Ethereum vs Visa and PayPal Transactions per second. https://mybroadband.co.za/news/banking/206742-bitcoin-and-ethereum-vs-visa-and-paypal-transactions-per-second.html, 2017. [Accessed: 2017-08-20].

[42] Wikipedia. Google Play. https://en.wikipedia.org/wiki/Google_Play. [Accessed: 2017-08-12].

[43] Wikipedia. Nxt. White paper, 2017. [Online; accessed 21-September-2017].

Legal notice to Investors

The AppCoins are not securities or units in a collective investment scheme or business trust, each as defined under Singapore's Securities and Futures Act (Cap. 289) ("SFA"). Accordingly, the SFA does not apply to the offer and sale of the AppCoins. For the avoidance of doubt, this initial offering of AppCoins need not be accompanied by any prospectus or profile statement and no prospectus or profile statement needs to be lodged with the Monetary Authority of Singapore ("MAS").

This white paper does not constitute an offer of, or an invitation to purchase, the AppCoins in any jurisdiction in which such offer or sale would be unlawful. No regulatory authority in Singapore, including the MAS, has reviewed or approved or disapproved of the AppCoins or this document. This white paper document and any part hereof may not be distributed or otherwise disseminated in any jurisdiction where offering tokens in the manner set out this white paper document is regulated or prohibited.

The information in this white paper document is current only as of the date on the cover hereof. For any time after the cover date of this white paper document, the information, including information concerning Aptoide's business operations and financial condition may have changed. Neither the delivery of this white paper document or any sale made in the related initial token offering shall, under any circumstances, constitute a representation that no changes have occurred in relation to Aptoide's affairs after the date of this white paper document.

Whether taken as a whole or read in part, this white paper document is not, and should not be regarded as, any form of legal, financial, tax, or other professional advice. You should seek independent professional advice before making your own decision as to whether or not to purchase any AppCoins. You are responsible for any and all evaluations, assessments, and decisions you make in relation to investing in the AppCoins. You may request for additional information from Aptoide in relation to this offer of AppCoins. Aptoide may, but is not obliged to, disclose such information depending on whether (i) it is legal to do so and (ii) the requested information is reasonably necessary to verify the information contained in this white paper document.

Upon purchasing any AppCoins, you will be deemed to have reviewed this white paper document (and any information requested and obtained from Aptoide) in full and to have agreed to the terms of this offering of AppCoins, including to the fact that this offering does not fall within the scope of any securities laws in Singapore and is not regulated by the MAS. You further acknowledge and agree that the AppCoins are not securities and are not meant to generate any form of investment return.