# Rise of the Planet of the Apps: A Systematic Study of the Mobile App Ecosystem

Thanasis Petsas
FORTH-ICS, Greece
petsas@ics.forth.gr

Antonis Papadogiannakis
FORTH-ICS, Greece
papadog@ics.forth.gr

Michalis Polychronakis
Columbia University, USA
mikepo@cs.columbia.edu

Evangelos P. Markatos
FORTH-ICS, Greece
markatos@ics.forth.gr

Thomas Karagiannis
Microsoft Research, UK
thomkar@microsoft.com

## ABSTRACT

Mobile applications (apps) have been gaining rising popularity due to the advances in mobile technologies and the large increase in the number of mobile users. Consequently, several app distribution platforms, which provide a new way for developing, downloading, and updating software applications in modern mobile devices, have recently emerged. To better understand the download patterns, popularity trends, and development strategies in this rapidly evolving mobile app ecosystem, we systematically monitored and analyzed four popular third-party Android app marketplaces. Our study focuses on measuring, analyzing, and modeling the app popularity distribution, and explores how pricing and revenue strategies affect app popularity and developers' income.

Our results indicate that unlike web and peer-to-peer file sharing workloads, the app popularity distribution deviates from commonly observed Zipf-like models. We verify that these deviations can be mainly attributed to a new download pattern, to which we refer as the *clustering effect*. We validate the existence of this effect by revealing a strong temporal affinity of user downloads to app categories. Based on these observations, we propose a new formal clustering model for the distribution of app downloads, and demonstrate that it closely fits measured data. Moreover, we observe that paid apps follow a different popularity distribution than free apps, and show how free apps with an ad-based revenue strategy may result in higher financial benefits than paid apps. We believe that this study can be useful to appstore designers for improving content delivery and recommendation systems, as well as to app developers for selecting proper pricing policies to increase their income.

## Categories and Subject Descriptors

D.4.8 [**Operating Systems**]: Performance—*Measurements; Modeling and prediction; Simulation*

## Keywords

Appstores; Mobile Apps; Workload Characterization; App Popularity; Clustering effect; App Pricing; Revenue


*IMC'13,* October 23–25, 2013, Barcelona, Spain.

http://dx.doi.org/10.1145/2504730.2504749.

## 1. INTRODUCTION

With more than 640 million active mobile devices as of July 2012 [9], the market of mobile devices is one of the fastest growing. Along with this growth, recent advances in mobile technologies facilitated a plethora of *mobile applications (apps)*; for example, the Android Market experienced a sevenfold increase in the number of offered apps from 100,000 to 700,000 within the last two years [8]. Typically, apps are hosted and distributed to end users through an increasing number of *mobile application stores (appstores)*. Appstores provide a new way of developing, downloading, and updating software applications for mobile devices, and create new business opportunities for application developers.

Despite this rapid evolution, there have been only a few large-scale studies of mobile appstores, mostly focusing on security and privacy aspects of mobile apps [32, 33, 44, 45]. However, appstores provide essentially a public marketplace matching mobile users and developers, and hence can offer valuable insights across a diverse set of dimensions ranging from app download patterns and popularity trends, to developers' revenue strategies. To this end, through a study of four popular third-party Android appstores over several months, our work aims to better understand appstore properties by focusing on two main aspects:

*Q1:* How can we characterize the popularity of different mobile applications?

*Q2:* What are the pricing and revenue strategies followed by app developers?

To examine *Q1*, we first characterize the app popularity distribution. Our results highlight a typical "Pareto effect," with 10% of the apps accounting for 70–90% of the total downloads. While such phenomena have also been observed in the popularity of web objects [25] and video content [21], the app popularity distribution appears truncated at both head and tail, and this observation is consistent across all examined appstores. Our analysis shows that truncation at the head of the distribution is caused by the "fetch-at-most-once" property, also found in peer-to-peer file sharing systems [34]. Contrary, we attribute tail truncation to a phenomenon we refer to as the "clustering effect," which captures the tendency of users to show a strong temporal affinity to app categories, by selectively downloading apps from certain categories over time. These insights lead us to propose a new model to approximate the popularity distributions observed in appstores, and through simulation results we show that it closely captures the observed behaviors.

Typically, appstores offer two categories of mobile apps: paid and free. In *Q2*, we examine whether offering a paid or free ad-based app is a better revenue-making strategy for developers. To this end, we first study the differences between paid and free apps.

While as expected free apps are far more popular, interestingly, paid apps appear to follow a different popularity distribution, which more closely resembles a Zipf distribution. We conjecture that this is the result of users being more selective when paying for apps.

Looking at developers' income from paid apps, we find that the majority of them earn a very small income, and offering a large number of apps does not lead to larger income; quality is more important than quantity. In contrast, by inspecting application binaries, we find that 67.7% of the free apps include at least one library that belongs to the 20 most popular advertising networks. Our analysis suggests that, on average, a free app needs to make just $0.21 per download through ads to match the income offered by a paid app. For popular apps, this number drops to $0.033. Hence, opting for a free app appears to be a more lucrative strategy.

Overall, we believe that our study offers multiple insights to both appstore designers and app developers. By understanding app popularity patterns, appstore and mobile operators can improve performance (e.g., by caching the most popular applications as shown by the Pareto effect), usability (e.g., by pre-installing very popular applications to mobile devices), and the relevance of recommendation and search systems (e.g., by taking into account the clustering effect). Similarly, insights on app monetization can be crucial for developers. Indeed, our study further shows that choosing an appropriate app category significantly affects the expected income.

The main contributions of this work are:

- We study the popularity distribution of mobile apps and compare it to the popularity distributions seen in web, peer-to-peer file sharing, and user-generated content. We show that app downloads follow a power-law distribution with truncated edges and have a highly-skewed Pareto effect, with 10% of apps accounting for 70–90% of total downloads.
- We provide a model of app download distribution and validate it using data from four Android appstores. We show that our model fits the measured data much more closely than previously proposed models for web and peer-to-peer file sharing. We attribute this difference to the "clustering effect," which we validate with a study of user behavior patterns.
- We study the differences between paid and free apps and show that (i) they have different download distributions, and (ii) they lead to fundamentally different revenue patterns for developers. We show that, in most cases, free apps that follow an ad-based revenue strategy may result in higher financial benefits than paid apps.

# 2. DATA COLLECTION

In this section we describe the data sources we used, our data collection architecture, and summarize the collected data, which are analyzed in the rest of the paper.

## 2.1 Monitored Appstores

For our study we systematically collected information from four popular Android app marketplaces: SlideMe [16], 1Mobile [13], AppChina [15], and Anzhi [14]. SlideMe is one of the oldest Android marketplaces, founded in 2008, containing more than 20,000 apps. 1Mobile is one of the largest third-party appstores, with more than 150,000 apps. AppChina and Anzhi are very popular appstores located in China, with more than 50,000 apps each. All marketplaces provide a website through which users can browse, search, download, and buy apps. They also provide an application manager tool that allows users to manage, search, and download apps directly from their Android device, in a similar way as the official Google Play Store App. Anzhi is often found as the pre-

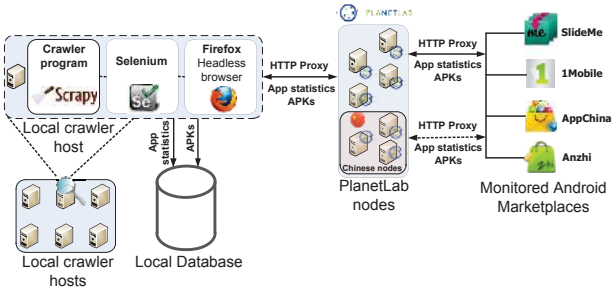



**Figure 1: *Data collection architecture.* Local crawler hosts use Planetlab nodes as proxies to download per-app statistics and APKs, which are stored in a local database.**

configured appstore on HTC smartphones in China. Many tablet manufacturing companies like *Disgo*, as well as many telecommunication providers like *Vodafone*, offer their devices pre-installed with SlideMe [5, 17]. More than 120 OEM device models use SlideMe as their default appstore [35].

To explore similarities and differences among various appstores, we chose a representative and diverse sample of the available marketplaces. In terms of geographical distribution, Anzhi and AppChina target the quickly rising mobile app market of China, while 1Mobile and SlideMe are mostly used by developers and clients located in Europe and the United States. In terms of the number of available apps, 1Mobile is one of the largest appstores, Anzhi and AppChina are considered as medium-size appstores, while SlideMe is a smaller one. The appstores also differ in the degree of popularity in terms of daily downloads. SlideMe provides both free and paid apps, while the rest provide only free apps.

One of the main reasons for choosing these appstores is that all of them provide the accurate number of downloads for each app, an aspect that is of great interest for our study. On the other hand, the official Android market Google Play [6], as well as other third-party marketplaces [1, 3, 4], do not provide a precise number of downloads or installations per app. Instead, they give only range approximations, which make any kind of systematic analysis very difficult. Besides the coarse-grained available information, Google's terms of service do not allow access to Google Play through any type of automated means.

## 2.2 Data Collection Architecture

For each appstore, we collect information in two phases. Initially, we gather a snapshot of the available apps by crawling the appstore through its web interface, and store all data into a local database. Then, the crawling process is repeated on a daily basis. Each day, our crawling system re-visits the web pages for the already indexed apps to update their daily statistics. It also collects information for all the newly added apps, which are also stored into the database expanding the initial dataset. For each app, we collect statistics such as the number of downloads, user ratings and comments, current version, category, price, and information about the developer. We also download any updated APK files (i.e., the app itself), collecting this way all versions of every app in the marketplace. Note that we download each app version only once, so we do not affect the actual number of downloads.

Figure 1 shows our data collection architecture. The system consists of a set of local machines, each hosting several crawler instances. We also use a set of nodes (roughly around 100) from the PlanetLab infrastructure [23], which act as HTTP proxies. Before sending an HTTP request to an appstore, each crawler instance

| Appstore | Crawling period | Total apps (first day / last day) | New apps per day (average) | Total downloads (first day / last day) | Daily downloads (average) |
|---|---|---|---|---|---|
| Anzhi | 2012-06-04 – 2012-08-03 | 58,423 / 60,196 | 29.6 | 1,396 M / 2,816 M | 23.7 M |
| AppChina | 2012-03-30 – 2012-06-03 | 33,183 / 55,357 | 336.0 | 1,033 M / 2,623 M | 24.1 M |
| 1Mobile | 2012-03-21 – 2012-08-01 | 128,455 / 156,221 | 210.4 | 367 M / 453 M | 651.5 K |
| SlideMe (free) | 2012-03-01 – 2012-08-01 | 12,296 / 16,578 | 28.0 | 63 M / 96 M | 215.7 K |
| SlideMe (paid) | 2012-03-01 – 2012-08-01 | 4,606 / 5,606 | 6.5 | 111 K / 914 K | 5.2 K |

**Table 1: *Summary of collected data.* We collected data about more than 300,000 apps while monitoring four different appstores for 2–5 months. AppChina has the largest number of daily downloads and new apps per day.**

randomly selects one of these proxies. This distributed crawling scheme was necessary to avoid any IP address blacklisting. The Chinese appstores apply rate limiting to hosts away from China, so we used only the PlanetLab nodes that are located in China for proxying their crawling connections.

We implemented a specific crawler for each appstore using the Scrapy framework [10]. We designed our crawlers to comply with the thresholds set by each appstore in terms of requests per time unit. To gather data from pages with dynamically generated content with JavaScript, we used the Selenium Remote Control (RC) [11], a browser automation tool, combined with a headless Firefox browser running on an X virtual framebuffer. For such pages, the crawler was altered to make the HTTP requests through Selenium with the controlled headless browser, so that Scrapy collects all the required information from the already rendered HTML page.

## 2.3 Data Set

Table 1 summarizes the collected data. For SlideMe, 74.7% of the apps we collected are free, while the rest 25.3% are paid. As of August 2013, the examined appstores had 316,143 apps and 7 billion downloads in total; this is roughly 52.7% of the 600,000 apps and 35% of the 20 billion downloads featured in Google Play at the same time [7]. This implies that our dataset comprises a significant part of the mobile app ecosystem.

We see that a small number of new apps are added to SlideMe and Anzhi by app developers every day. On the other hand, a higher number of new apps are added every day to AppChina and 1Mobile. This is probably due to the increasing popularity of the AppChina marketplace, while 1Mobile hosts the largest number of apps among these appstores. Indeed, we see that AppChina is the most popular appstore in terms of user downloads, with more than 24 million of downloads per day. It seems that this popularity attracts an increasing number of developers that choose this marketplace to distribute their applications. Although 1Mobile hosts the largest number of apps, it has a lower number of downloads than AppChina and Anzhi, which target smartphone users in China. We also observe that paid apps receive significantly less downloads than free apps, and fewer paid apps are added per day to SlideMe.

# 3. APP POPULARITY

First, we study app popularity based on the distribution of downloads of each app among the different appstores.

## 3.1 Is There a Pareto Effect?

Previous studies have shown that web content [25] and video downloads [21] usually follow the Pareto Principle: 20% of the objects are responsible for 80% of the downloads. Figure 2 shows the CDF of the percentage of app downloads as a function of the app ranking (ranked from the most popular to the least popular) for the different appstores. The results confirm that a small percentage of apps is responsible for a large percentage of downloads. For example, in AppChina and Anzhi, about 10% of the apps are responsible for close to 90% of all downloads. Similarly, 10% of the

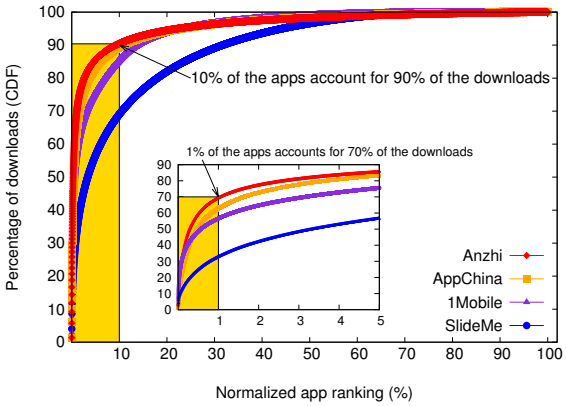


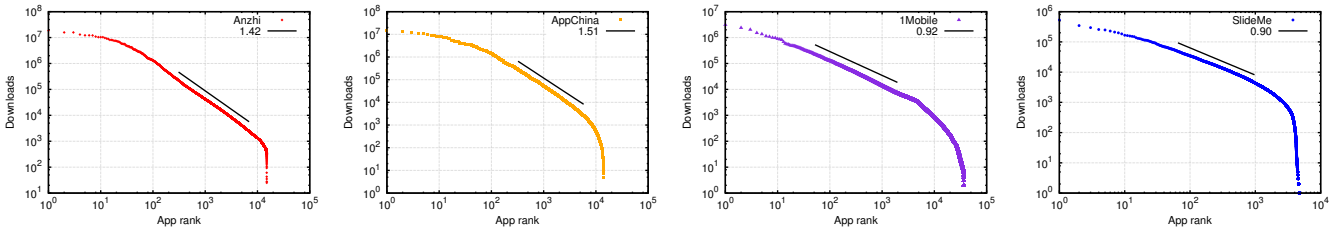

**Figure 2: *A few apps account for most of the downloads.* The CDF of downloads per app shows that 10% of the apps account for 90% of the total downloads.**

1Mobile apps are responsible for more than 85% of the downloads, and 10% of the SlideMe apps are responsible for more than 70% of all downloads in this appstore. We see that this uneven distribution of popularity goes all the way into the top 1% of the applications, which are responsible for more than 70% of downloads in Anzhi, more than 60% in AppChina, more than 55% in 1Mobile, and more than 30% in SlideMe. This evidence for the existence of the Pareto effect is significant for the design of efficient caching mechanisms for application delivery to end users. For example, a cache large enough to hold 1% of the apps will have a hit rate as high as 70%.

## 3.2 Is There a Power-law Behavior?

Although it is clear that app downloads follow a pareto principle, we would like to explore whether the app downloads follow a power law distribution much like web downloads do [20]. Figure 3 shows the number of downloads per app, when all apps are sorted in the $x$ axis according to their rank based on their total downloads. We see that all distributions share a similar pattern: their main "trunk" has a linear slope indicating a Zipf distribution, which is truncated at both ends (i.e., for small and large $x$ values).

The truncation for small $x$ values (i.e., most popular apps) seems to follow similar patterns shown in the downloads of file sharing systems [34] and video downloads in YouTube [21]. This truncation on these systems was attributed to the *fetch-at-most-one* principle. That is, objects shared in a file sharing system, such as music, videos, movies, etc., tend to be downloaded at most once by each user. Therefore, the curve for very popular content (i.e., small values in the $x$ axis) tends to flatten out and reach a value close to the number of users in the system. We see the same behavior in Figure 3: for very small $x$ the number of downloads stays almost horizontal (especially in AppChina and Anzhi appstores), which is probably due to the fetch-at-most-one principle. It is reasonable to expect that smartphone users will also download each app at most once, apart from apps which are updated.




(a) Anzhi (b) AppChina (c) 1Mobile (d) SlideMe

**Figure 3:** ***App popularity distribution deviates from Zipf at both ends.*** **The distribution of total downloads per app as a function of app's rank shows a main trunk that follows a Zipf distribution with truncated ends. The truncation for small $x$ values can be attributed to the *fetch-at-most-once* property observed in most popular apps. We explain the truncation for large $x$ values with another phemomenon we call as *clustering effect*.**

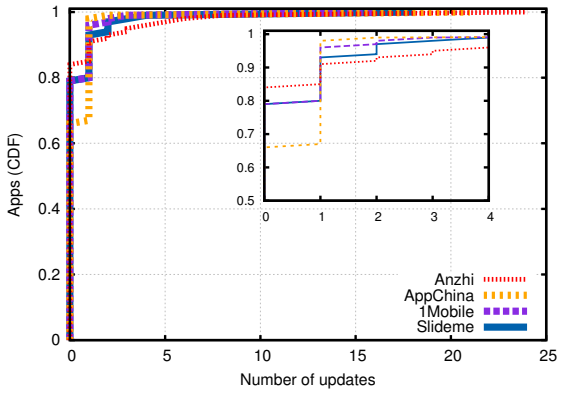



**Figure 4:** ***Apps are not updated often.*** **CDF of the number of updates per app within two months shows that a small percentage of apps was updated in this period. This implies that users download most apps just once, or very few times, verifying the fetch-at-most-once property.**

To validate that the fetch-at-most-once property exists in mobile appstores as well, and it is not affected by application updates, we plot in Figure 4 the CDF of the number of updates per app in the four appstores for a period of two months. We see that more than 80% of the apps are not updated at all within this period. Even the apps that are updated, they have a very small number of updates, e.g., the 99% of the apps have less than four updates. Since the fetch-at-most-once property affects only the most popular apps, limiting their downloads to the number of users in the market, we focused only on the top 10% most popular apps. We found that these apps are also rarely updated: 60%-75% of them have no updates in this period, while 99% of these apps have up to six updates. These results imply that the fetch-at-most-once property applies to mobile app market, as apps are not updated often.

Moreover, Figure 3 clearly shows that there is a significant curvature for large $x$ values (less popular apps) as well. Observed in user-generated content downloads [21], but not in file sharing [34], this curvature has been thought to be attributed to search engines and recommendation systems. Previous works argue that these systems tend to favor the most popular content, due to information filtering, which results to the observed truncation of power law [22,37]. However, we feel that this curvature is an instance of a more general phenomenon, to which we refer as the *clustering effect*.

The clustering effect suggests that apps are grouped into (static or dynamic) sets. Apps within the same set (cluster) are correlated: if a user downloads one of them, then the same user will probably download another app of the same set rather than switching to another set, or to a totally unrelated app. These clusters can be formed, for example, by the semantic classification of apps to categories, by user communities as a result of positive comments, by the appstore as a result of the recommendation algorithm used, or by other grouping forces. To validate our clustering effect hypothesis, we define and measure the affinity of user downloads among the several app categories in Section 4. Then, we propose a model for user downloads based on clustering effect and we validate it with simulations comparing to the actual gathered data in Section 5.

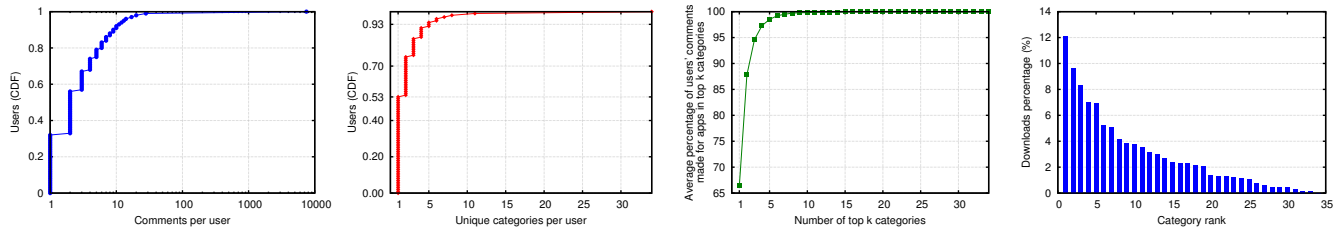
# 4. THE CLUSTERING EFFECT

In our next set of measurements we explore whether the previously suggested "clustering effect" hypothesis can be validated. One way to validate the hypothesis would be to show that (A) apps are categorized into clusters, and (B) once users have downloaded an app from one cluster, it is more likely to download another app from the *same* cluster, rather than switching to another one.

Fortunately, point (A) above is easy to show. Most appstores already group apps into clusters that are usually thematically related to the app's content. For example, there exists a cluster for "games," a cluster for "e-books," a cluster for "business and finance," and so on. To validate point (B), we would like to study the download patterns of individual users and see if they tend to concentrate around a few clusters or not. Unfortunately, appstores tend not to reveal the download patterns of individual users, mostly due to privacy concerns, and thus it is not easy to study them. Instead, we use publicly available user comments to approximate their download patterns and interests. If a user posts a publicly available comment for a particular app, we assume that the user has download the app. Thus, publicly available comments provide us with access to a subset of the download patterns of individual users.

## 4.1 Users Focus on a Few Categories

In this set of measurements we focused only on data from the Anzhi appstore, because it provides user comments with precise timestamps. We considered only user comments that are accompanied by a rating of the app, which is a strong indication of actual downloads. During the crawling process of the Anzhi appstore, we recorded and stored into our database the comments of each user on each app. The crawling process of user comments from the Anzhi marketplace resulted in a dataset of 361,282 users to 60,196 apps in 34 categories. Figure 5(a) shows the distribution of the number of comments per user. We see that 99% of the users made up to 30 comments. In many categories, there were a few users with a very large number of comments. We found that these users were posting spam, possibly using an automated script.




(a) Comments per user (b) Unique categories per user (c) Comments on top k categories (d) Downloads per app category

**Figure 5: *Users focus on a few categories.* (a) shows that 92% of the users made up to 10 comments, while 2% of the users made more than 20 comments. (b) shows that 53% of the users leave feedback on apps from a single category, while 94% comment on up to five categories. (c) shows that 66% of the comments are for a single category, while 95% are made for up to three categories. (d) shows that there is no dominant category in terms of downloads, as the most popular app category has 12% of the total downloads.**

Figure 5(b) shows the CDF of unique categories for which each user comments on, for users that posted at least one comment. We see that 53% of the users commented on apps from a single category, and 94% of the users commented on apps from up to just five categories. Thus, only 6% of the users made comments in more than five categories, which is significantly lower than the percentage of users that made more than 5 comments in general (20%), as shown in Figure 5(a). Similarly, in Figure 5(c) we present the average percentage of user comments made for the top $k$ categories, as a function of $k$. We have excluded users that made comments on a single app in this figure. We observe that the 66% of the comments of an average user were made for a single category, while 95% of the user comments were made for no more than five categories.

These findings indicate that most users tend to download apps from a single or very few categories, which implies a clustering effect. One could argue that the popularity distribution of app categories can influence user downloads. For example, *games* are more likely to be downloaded than apps belonging to the *tools* category, as the *games* category has a higher number of total downloads than *tools*. Therefore, a few popular categories could attract most of the users downloads. Figure 5(d) shows the distribution of app downloads across the different app categories. We see that the most popular category has just 12% of the total downloads, while the majority of categories have less than 4% of the total downloads. This fact indicates that there are no dominant categories in terms of downloads, and the popularity of app categories does not seem to have a considerable influence on users downloads. Consequently, the main reason for the small number of categories observed in each user's comments appears to be the clustering effect.

## 4.2 Temporal Affinity to App Categories

To study the download patterns in more detail and validate the existence of the clustering effect, we will first define and then measure the *temporal affinity* metric of users to app categories. During the crawling process of the Anzhi appstore we have recorded the stream of comments for each user on each app. We suppressed successive comments of the same user on the same app. For example, if a user commented on apps $a_1a_2a_3a_3a_1a_4$ we kept the sequence $a_1a_2a_3a_4$, which we refer to as *app string*. In addition, we know that appstore has already categorized each app $a_i$ into a category $c(a_i)$. Using this knowledge, for each app string $a_1a_2a_3a_3a_4$ we construct the *category string*: $c(a_1)c(a_2)c(a_3)c(a_4)$.

Given a category string $S$ of $n$ elements $c_1c_2c_3...c_n$ with the respective categories of a user's $n$ comments in chronological order, we define the *temporal affinity* metric Aff as the number of elements that are in the same category with their previous element, divided by $n-1$, using the following formula:

$$\text{Aff} = \frac{\sum_{i=2}^{n} \text{Affinity}(c_i, c_{i-1})}{n-1} \quad (1)$$

where $\text{Affinity}(c_i, c_j)$ equals to one if $c_i$ and $c_j$ are the same, and zero otherwise. For example, a user with a category string $c_1c_1c_1c_1$ will have temporal affinity equal to $3/3$, a user with a category string $c_1c_1c_1c_2$ will have temporal affinity equal to $2/3$, and a user with a category string $c_1c_1c_2c_3$ will have affinity equal to $1/3$.

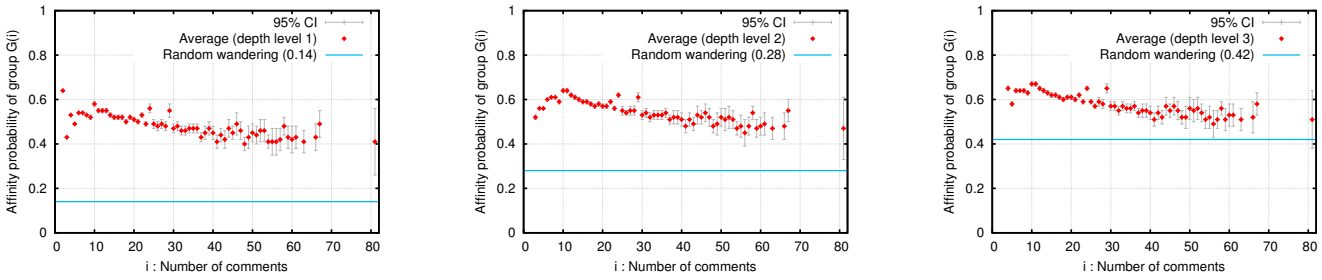
This affinity metric is an indication of whether users tend to comment on apps from the same category or on apps from different categories. Indeed, if affinity approaches the highest value (i.e., one), then users tend to repeatedly comment on apps from the same category, indicating that the users may tend to successively download apps from the same category. On the contrary, if the affinity is low, then users tend to switch from one category to another.

Note that if we have $C$ categories of roughly equal volume, then if users randomly wander from one category to another, the affinity will be around $1/C$. However, in practice, the apps are not evenly distributed among the different categories. To calculate the accurate affinity probability of a random walk in the Anzhi marketplace we used the actual distribution of apps to the $C$ different categories in this appstore from our dataset. Let $A$ be the total number of apps in the appstore and $A(i)$ the number of apps that belong to category $i$. Given this distribution, the random walk affinity probability $\text{Aff}_{\text{random walk}}$, i.e., the probability that two random app choices belong to the same category, is equal to:

$$\text{Aff}_{\text{random walk}} = \frac{\sum_{i=1}^{C} A(i) \times (A(i)-1)}{A \times (A-1)} \quad (2)$$

since out of the $A \times (A-1)$ possible random app choices, the number of app choices for which the two apps belong to the same category are equal to the sum of $A(i) \times (A(i)-1)$ for all categories. We need the affinity probability of a random walk as a base case to compare with the actual affinity we measure from Anzhi.

So far we have defined the affinity metric as the percentage of consecutive app pairs that belong to the same category. In this way, however, we miss users that may oscillate between few categories. For instance, the category string $c_1c_2c_1c_2$ has affinity equal to zero according to our previous definition, but we can see a clear affinity of this user to $c_1$ and $c_2$ categories. To address this issue, we define the affinity metric using the *depth* notion. That is, the affinity Aff with depth $d$ for a category string with $n$ elements is defined as the number of elements in the category string for which at least




(a) Depth=1 (b) Depth=2 (c) Depth=3

**Figure 6:** ***Successive user selections are from the same category with higher probability.*** **Temporal affinity for depth levels up to 3 for users grouped by their number of comments. We see that for depth=1, users select an app of the same category as the previous one with probability 0.55, which is 3.9 times higher than the base case of random walk. As expected, affinity increases with depth level.**

one element among its previous $d$ elements belongs to the same category, divided by $n - d$:

$$\text{Aff} = \frac{\sum_{i=d}^{n} \text{Affinity}(c_i, \{c_{i-1}, c_{i-2}, ..., c_{i-d}\})}{n - d} \quad (3)$$

where $\text{Affinity}(c_i, \{c_{i-1}, c_{i-2}, ..., c_{i-d}\})$ is equal to one if at least one of the $\{c_{i-1}, c_{i-2}, ..., c_{i-d}\}$ belongs to the same category with $c_i$, and zero otherwise. For example, the affinity of depth equal to two is defined as the percentage of app selections that belong to the same category with one of the previous two selections. To calculate the affinity metric for this depth, we first divide the $n$ apps of the category string into $n - 2$ triplets. Then we check which of these triplets have affinity, i.e., if the third element belongs to the same category with the second or with the first one in the triplet. Finally, we sum the triplets that have affinity and divide them with the total number of triplets, as shown in Equation 3.

Using Equation 2 we find the affinity probability of a random walk for depth equal to one. To calculate the affinity for an arbitrary depth $d$ we use the following equation:

$$\text{Aff}_{\text{random walk}} = \frac{\sum_{i=1}^{C} A(i) \times (A(i) - 1) \times d \times \prod_{k=2}^{d} (A - k)}{\prod_{k=0}^{d} (A - k)} \quad (4)$$

## 4.3 Users Exhibit a Strong Temporal Affinity to App Categories

We measure the temporal affinity metric as defined in Equation 3, based on the actual app distribution of the 34 categories of the Anzhi appstrore. Figure 6 shows the measured affinity for depth levels from 1 up to 3, as a function of the number of comments per user. We have grouped together all users that made the same number of comments, and we plot the average values and the $95^{th}$ confidence intervals from each group. We plotted only the groups that had more than 10 samples, excluding, in this way, the spam users as well. In case of depth equal to one, we see that the average affinity for most groups is around 0.55. This implies that once a user comments on an app from one category, the same user will comment on another app of the same category with probability close to 0.55. To place these numbers in context, we also calculated the estimated affinity of the base case as well, i.e., the case where a user wanders from one app to another randomly. The random walk affinity probability in our dataset for depth one is 0.14, shown as a horizontal line in Figure 6(a). Thus, we see that Anzhi users

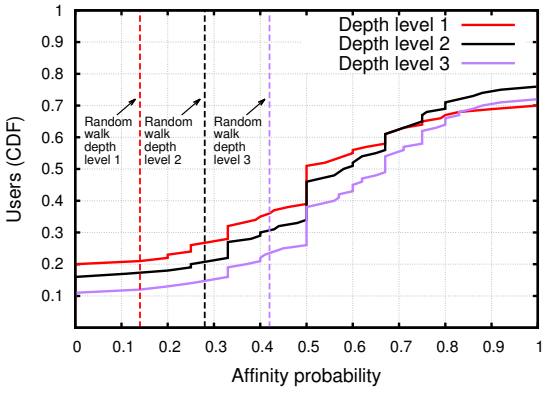


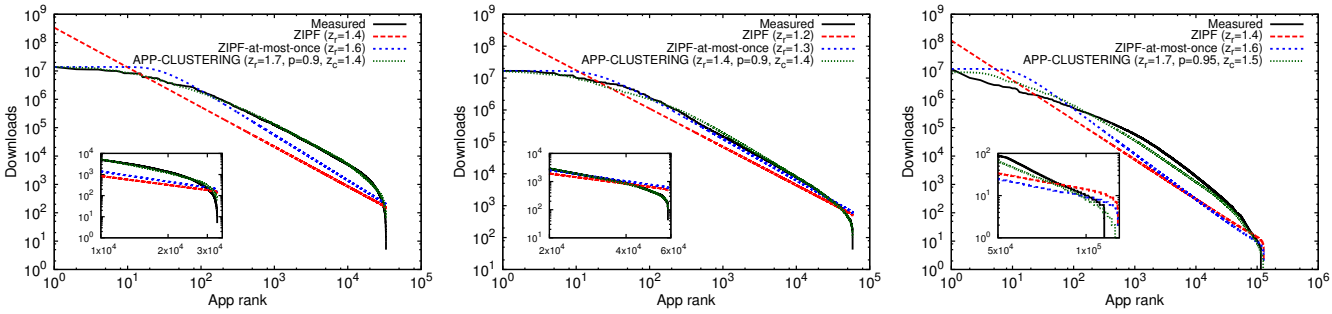

**Figure 7:** ***Most users exhibit a strong temporal affinity to app categories.*** **The CDF of the affinity metric for each depth level shows that 50% of the users have significantly higher affinity than the base case of random walk.**

are 3.9 times more likely to stay in the same category compared to the base case of random walk. This outcome indicates a strong temporal affinity between users and categories.

To explore how the temporal affinity increases for higher depth levels, we plot in Figures 6(b) and 6(c) the affinity metric of the same user groups, as well as the random walk affinity probability, for depths equal to two and three respectively. We see that user affinity to app categories increases with depth, as it was expected, and remains higher than the respective random walk affinity probability. To explore how the affinity varies among users, we plot the CDF of temporal affinity among all Anzhi users for the three different levels in Figure 7. We observe that the median value for depth one is 0.5, while for depths two and three the median values are 0.58 and 0.67 respectively. Overall, our results show that there is a clear temporal affinity of users' comments to app categories, which is a strong indication that users tend to download successive apps from the same categories.These observations validate our hypothesis about the existence of clustering effect in user downloads.

# 5. A MODEL FOR APPSTORE WORKLOADS

We now define a model of appstore usage called APP-CLUSTERING, based on fetch-at-most-once and clustering effect properties. The model will enable us to further explore our hypothesis and validate the effect of app clustering on app popularity distribution.




(a) AppChina (b) Anzhi (c) 1Mobile

**Figure 8:** ***Predicted versus measured app popularity for each appstore.*** **We see that APP-CLUSTERING fits very closely the measured data. ZIPF-at-most-once fits the data better than a pure ZIPF distribution, but diverges for large $x$ values.**

| Symbol | Parameter Description |
|---|---|
| $A$ | Number of apps |
| $U$ | Number of users |
| $D$ | Total downloads |
| $d$ | Downloads per user (average) |
| $z_r$ | Zipf exponent for overall app ranking |
| $Z_G$ | Overall Zipf distribution of all apps |
| $C$ | Number of clusters |
| $p$ | Percentage of downloads based on the clustering effect |
| $z_c$ | Zipf exponent for cluster's app ranking |
| $Z_c$ | Zipf distribution of apps in cluster $c$ |
| $D(i,j)$ | Predicted downloads for app with total rank $i$ and rank $j$ in its cluster |

**Table 2: APP-CLUSTERING model parameters.**

## 5.1 Model Description and Analysis

Table 2 summarizes the key parameters used in our model. We assume that all apps are categorised in $C$ clusters so that each app belongs to exactly one cluster. When a user downloads the first app, this download is drawn from a Zipf-like distribution $Z_G$. Once a user has downloaded at least one app, subsequent downloads will be from the same category of a previous download with probability $p$, and from a different category with probability $1-p$. Downloads from the same category of a previous download also follow a Zipf-like distribution $Z_c$. Therefore, each app has two rankings: its overall ranking $i$, taking values between 1 and $A$, and its ranking $j$ within its category, taking values between 1 and the number of apps in that category. Thus, the APP-CLUSTERING model suggests that each user behaves as follows:

**1.** Download the first app according to the $Z_G$ distribution.

**2.** Download another app:

- **2.1.** with probability $p$ the app will be downloaded from the same cluster $c$ of a previously downloaded app. The cluster $c$ is randomly chosen from previous downloads with a uniform probability. The app from cluster $c$ is drawn from distribution $Z_c$. If the app has been downloaded go to 2.1.
- **2.2.** with probability $1-p$ the app will be drawn from $Z_G$. If the app has been downloaded go to 2.2.

**3.** If user's downloads are less than $d$ go to 2.

**Analysis.** We assume an appstore with $U$ users, where each user downloads $d$ apps. Out of the $d$ downloads per user, $(1-p)\times d$ are app selections based on a pure Zipf distribution. That is, an app with rank $i$ has a probability to be selected for downloading equal to $(1/i^{z_r})/(\sum_{k=1}^{A} 1/k^{z_r})$. Similarly, when apps are selected from a specific cluster $C_a$, an app with rank $j$ in this cluster will be selected with probability equal to $(1/j^{z_c})/(\sum_{l=1}^{S_{C_a}} 1/l^{z_c})$, where $S_{C_a}$ the size of cluster $C_a$. For simplicity we assume that all $C$ clusters have the same size $S_C$. Overall, the expected downloads $D(i,j)$ for an app with overall rank $i$ and rank $j$ in its respective cluster with the APP-CLUSTERING model are:

$$D(i,j) = \sum_{u=1}^{U} 1 - \left(1 - \frac{1/i^{z_r}}{\sum_{k=1}^{A} 1/k^{z_r}}\right)^{(1-p)\times d} \times \left(1 - \frac{1/j^{z_c}}{\sum_{l=1}^{S_C} 1/l^{z_c}}\right)^{p\times d} \quad (5)$$

The expected downloads per app are estimated by adding the probability of all users to download this app. The probability of a single user to download this app is one minus the probability for not downloading this app with $(1-p)\times d$ Zipf-based selections and $p\times d$ clustering-based selections. We see that the number of downloads per app is limited by the number of users $U$.

## 5.2 Simulation-based Model Validation

To validate our model and understand the impact of clustering effect on the app downloads distribution, we developed three Monte Carlo simulators of an appstore, using ZIPF, ZIPF-at-most-once, and APP-CLUSTERING models. In the ZIPF simulator all downloads are drawn from $Z_G$. In the ZIPF-at-most-once simulator all downloads are drawn from $Z_G$, but each user can never download the same app more than once. Then we ran these simulators for all appstores in our dataset while varying their key parameters, in order to approximate the observed distribution of app downloads as close as possible. To measure how close each simulation approaches the actual downloads distribution of $A$ apps we calculate the distance between the observed and simulated downloads of each app using the mean relative error:

$$distance = \frac{1}{A}\sum_{i=1}^{A} \frac{|D_o(i) - D_s(i)|}{D_o(i)} \quad (6)$$

where $D_o(i)$ and $D_s(i)$ are the observed and simulated downloads, respectively, for the app with overall rank $i$.

### 5.2.1 Comparing Modeled and Actual Downloads

Figure 8 compares the simulation results of the ZIPF, ZIPF-at-most-once, and APP-CLUSTERING models with the measured downloads in AppChina, Anzhi, and 1Mobile. We tuned the parameters of each model to produce the best data fit, by running simulations with all parameter combinations, and measuring the distance from actual data. We plot the results of each model using

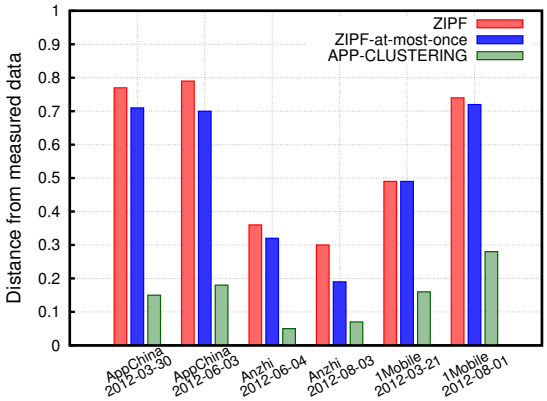



**Figure 9: *APP-CLUSTERING has the smallest distance from measured data.* APP-CLUSTERING approximates the actual downloads up to 7.2 times closer than ZIPF and up to 6.4 times closer than ZIPF-at-most-once.**

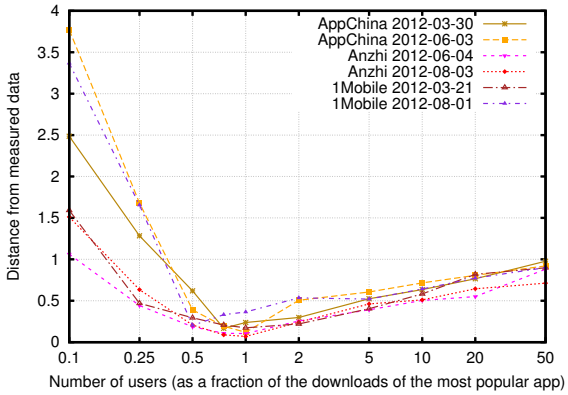



**Figure 10: *Downloads of the most popular app are a good estimate of the number of users.* We see the minimum distance from the actual downloads when the number of users is close to the downloads of the most popular app.**

the parameters that produced the minimum distance from actual downloads. We first see that in all figures the pure ZIPF distribution (dashed red line) clearly deviates from the measured data. The deviation is obvious: ZIPF is a straight line (in log-log scale) while the measured data follow a curved line at both ends of the x spectrum. Indeed, for small $x$ (i.e., popular apps) ZIPF overshoots the measured data by more than an order of magnitude. ZIPF-at-most-once (dashed blue line) fits the data better than ZIPF, especially for small x (i.e., for popular apps), but deviates from the actual data mainly for large $x$ values. On the other hand, APP-CLUSTERING matches the data very close, better than any other model, both for small $x$ and especially for large $x$ values (i.e., least popular apps).

As we increase $p$, i.e., the percentage of simulated downloads based on clustering, we see that the distance of APP-CLUSTERING from the actual data is reduced, and data fitting is improved. The best approximations of the actual data are achieved when the percentage $p$ of clustering-based downloads is 90–95%. This outcome provides strong evidence that clustering affects app downloads distribution and is responsible for the observed tail behavior.

Figure 9 shows the distance of the three models from the measured data for the first and last day of our crawling period in AppChina, Anzhi, and 1Mobile. We see that APP-CLUSTERING has the smallest distance from measured data for all appstores. For instance, in the first day AppChina dataset, APP-CLUSTERING predictions result in 0.15 distance from the actual downloads, which is 4.7 times better than ZIPF-at-most-once and 5.1 times better than ZIPF. We see similar results from simulations of other appstores as well. Overall, we observe that APP-CLUSTERING model is able to approximate the actual downloads of all appstores very well from the first day up to the last day of our measurement period.

### 5.2.2 *Choosing the Right Number of Users*

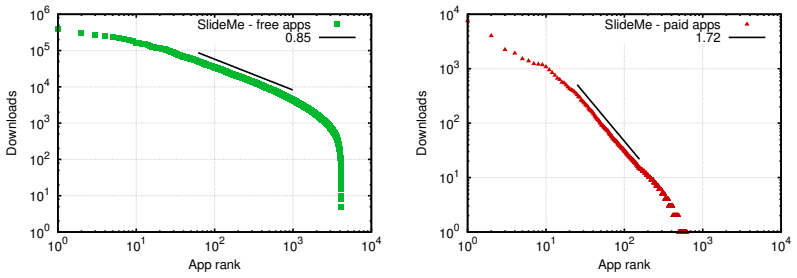
Using the APP-CLUSTERING simulator we also explore how the number of users $U$ influences the simulation results. Since we do not have access to the logs of the appstores, we do not know the actual number of users who have accessed each appstore. We know the total number of downloads, the total number of apps, but we do not know the total number of users. Nevertheless, we conduct simulations in order to explore whether there is a reasonable range for the number of users of the appstores studied that results to very close approximations of the actual downloads distribution per app. Given that different appstores have a different actual number of users, we express the number of users as a function of the total number of downloads of the most popular app.

Figure 10 shows our simulation results while varying the number of users $U$ and setting the rest of the simulation parameters to the values that produce the minimum distance from the actual data. The $x$-axis is the number of the simulated users, as a ratio of the total number of downloads of the most popular app, and the $y$-axis reports the distance between the simulation results and the measured downloads. We plot the simulation results for the app downloads of the first and last day of the AppChina, Anzhi, and 1Mobile appstores. We see that in all cases the minimum distance from the measured data is achieved when the number of users is very close to the number of downloads of the most popular app.

## 6. APP PRICING AND INCOME

Among the four appstores we studied, AppChina, Anzhi, and 1Mobile offer solely free apps—only SlideMe provides both free and *paid* apps. One might wonder why developers choose third-party marketplaces to upload paid apps instead of Google Play, which could result in increased app visibility. There are several reasons to do so. First, many developers upload their apps in as many alternative markets as possible in order to gain more popularity, increase their income, and attract users with devices that have third-party appstores pre-installed, such as SlideMe, as explained in Section 2. Also, Google Play supports only specific locations for merchants [12]. Since developers living in countries not supported by Google cannot upload paid apps to Google Play, they usually choose an alternative market like SlideMe. Finally, third-party marketplaces may provide benefits to developers, e.g., SlideMe does not charge transaction fees besides a payment processing fee [35].

Paid apps usually have more advanced functionality and do not include advertisements. In this section, we focus on the SlideMe appstore to study how price affects app popularity, how revenue is distributed among developers and categories, and how different pricing strategies may affect developers' income. The main questions that drive our analysis regarding app pricing and developers' income are the following: (Q1) Which are the main differences between paid and free apps? (Q2) What is the developers' income range? (Q3) Which are the common strategies when offering an app, and how do these strategies affect revenue? In the rest of this section we explore these questions in more detail using the dataset of the SlideMe appstore.




(a) SlideMe - free apps (b) SlideMe - paid apps

**Figure 11:** ***Paid apps follow a clear Zipf distribution.*** **The downloads distribution of free and paid apps shows that paid apps clearly follow a power-law distribution, while free apps follow the same distribution we observed in Section 3.2.**

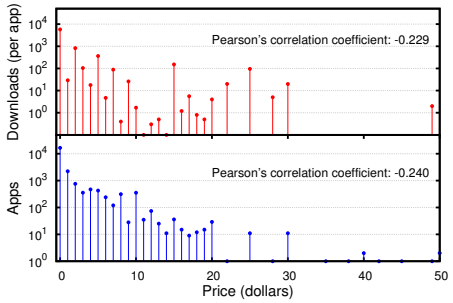



**Figure 12:** ***Expensive apps are less popular.*** **Number of downloads and apps as a function of price. We see that the number of downloads and number of apps are negatively correlated with price.**

## 6.1 The Influence of Cost on App Popularity

First, we explore how cost affects app popularity. To understand the different download patterns between paid and free apps (Q1), Figure 11 shows the distribution of downloads for free and paid apps separately. Free apps follow a similar distribution as the one observed in Section 3.2. Interestingly, we see that paid apps clearly follow a pure power-law distribution, with no significant deviations. This is probably due to the fact that users are more selective when downloading paid apps: they are less influenced by recommendation systems and more considerate about cost. Thus, less popular apps tend to stay that way and are deprived from any casual downloads. Contrary, free apps enjoy much more downloads and even less popular apps get a decent number of downloads.

Figure 12 shows the average number of downloads per app, as well as the number of apps as a function of price. Since the app price may change, we use the average price of each app we observed within our measurement period. We group together all apps that their price is within a range of one dollar, e.g., apps with price between two and three dollars belong to the same bin, and we plot the number of apps for each bin (in the lower part) and the average number of downloads among the apps of each bin (upper part). It is expected that apps with higher prices will get less downloads. Indeed, we observe a negative correlation between price and downloads, with Pearson's correlation coefficient equal to -0.229. This means that apps with higher prices are less likely to become popular. Similarly, the results indicate that there is a negative correlation between price and number of apps as well, i.e., we see more apps with lower prices, with a correlation coefficient equal to -0.240. This means that most developers offer their apps in lower prices, hoping for increased popularity.

## 6.2 Developers' Income

Next, we set out to explore how the app market's revenue from paid apps are distributed among the developers, and which parameters affect the developers' income (Q2). To estimate the income of each developer we rely on the total number of downloads (purchases) of each paid app offered by this developer, multiplied by the average price of this app during our measurement period. The average revenue of a single paid app is $3.9. Appstores usually get a commission from purchases, typically around 20–30% of the app's price, and the remaining amount is the actual developer's income. However, SlideMe has one of the lowest commissions on app purchases: it gets 5% of the app's price for each download. From the total revenue, which is close to 4 million dollars in our dataset, SlideMe should get around 200 thousand dollars as commission. For simplicity in our measurements we assume that developers get the whole amount from each download of a paid app.

Figure 13 shows the CDF of total income per developer. We see that half of the developers made less than $10, while 27% of them had no income from their paid apps. The 80% of the developers made less than $100, and 95% of them less than $1,500. However, we observe that there is a small percentage of developers, roughly around 1%, with income higher than 2 million dollars from very popular applications. Overall, it is clear that the majority of developers in the SlideMe appstore had a negligible income by selling paid apps, while only a very small percentage of them made a significant income.

An interesting question that comes up is whether the total income of a developer is correlated with the number of apps that this developer creates. In other words, can a developer earn more by creating more applications? To answer this question we plot in Figure 14 the number of paid apps per developer versus the average total income of the developers with this number of paid apps. We see that the developer's income does not increase with the number of apps, and there is no clear relation between them. The Pearson's correlation coefficient between income and number of apps per developer is equal to 0.008, which validates that these values are not correlated. Therefore, it seems that a developer will not necessarily reach a wider set of users by simply offering more apps. This is very interesting, because it shows that quality is more important than quantity for the developers' income.

We also measure how the revenue from paid app installations is distributed to app categories, and whether the percentage of apps and developers per category are related to the category's revenue fraction. Figure 15 shows the percentage of the total revenue per app category, along with the percentage of apps and developers for the same categories. We see that 67.7% of the revenue comes from paid apps of the music category, 19.7% from games, while 3.9% and 3.7% of the revenue comes from utilities and productivity categories respectively. Overall, 95% of the revenue comes from these four categories, while the rest 16 categories contribute only with 5% of the developers' income. These four categories include the 30.4% of the apps and 72% of the developers. Despite the fact that music produces 67.7% of the total revenue, we see that it contains just 1.6% of the paid apps, and only 4.3% of the developers have made at least one app in this category. We also observe that 33.2% of the apps are in the e-books category, created by only 1.6% of the developers, and they result to just 0.1% of the total revenue. On the other hand, 18.3% of the apps are games, offered by 36.2% of the developers, resulting to 19.7% of the total revenue, which is a reasonable ratio. Overall, we see that the fraction of revenue per

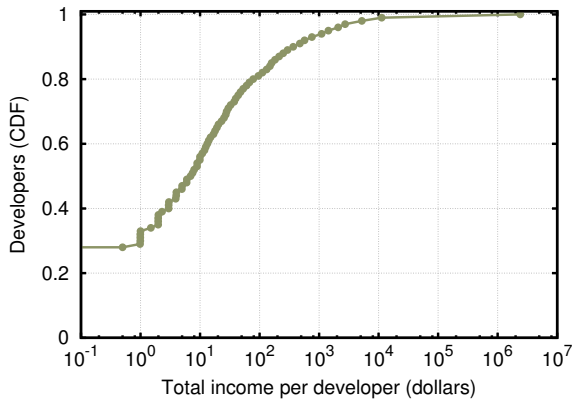



**Figure 13:** ***Most developers have negligible income from paid apps.*** **80% of the developers made less than $100, while 1% of them gained more than 2 million dollars.**

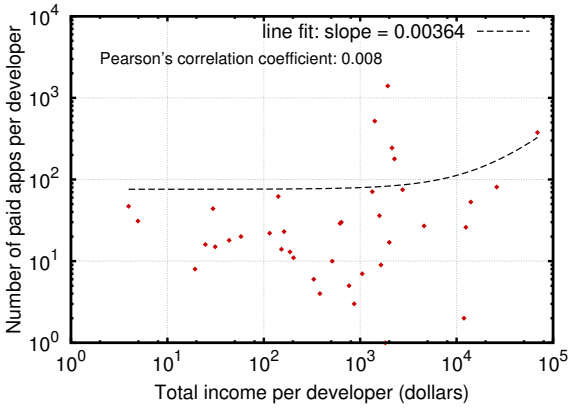



**Figure 14:** ***Quality is more important than quantity.*** **We see that developer's income is not correlated with the number of apps that this developer offers.**

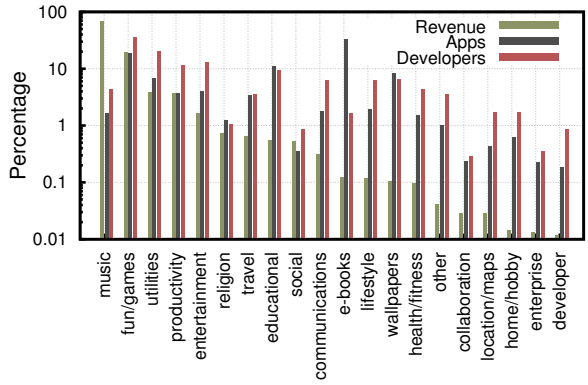
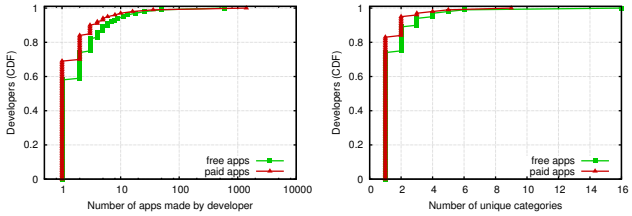



**Figure 15:** ***Revenue comes from few categories.*** **We see that 95% of the total revenue comes from the music, games, utilities, and productivity categories. The revenue per category is not correlated with the number of apps per category, and it is slightly correlated with the number of developers in these categories.**




(a) Number of apps (b) Number of categories

**Figure 16:** ***Developers create a small number of apps focused on few categories.*** **CDF of the number of free and paid apps per developer shows that 95% of the developers offer less than 10 apps. CDF of the number of unique categories per developer shows that 99% of the developers focus on 1–5 categories.**

category is not related with the percentage of apps in the same category, with Pearson's correlation coefficient equal to 0.014, and it is slightly related with the percentage of developers in these categories, with correlation coefficient equal to 0.198. The percentage of apps and developers per category are more correlated, with Pearson's correlation coefficient equal to 0.378.

## 6.3 Revenue Stategies

In this section we set out to explore the common developers' strategies, and how these strategies may affect the developers' income (Q3). From our dataset we see that there exist 5,106 developers in the SlideMe marketplace for 22,184 applications. Thus, each developer builds 4.3 apps on average. Figure 16(a) shows the CDF of the number of free and paid apps offered by each developer. We see that 60% of the free app developers, and 70% of the paid app developers, create only a single app. Overall, 95% of the developers offer less than 10 apps. We also observe that there is a small fraction of developers that release a large number of apps, e.g., one developer with 1,402 apps and another with 592 apps. We found that these developer accounts correspond to two different companies that develop many mobile apps systematically. From these 5,106 developers we see that 75% of them offer only free apps, 15% of them offer only paid apps, while there is a 10% of developers that offer both free and paid apps. This outcome implies that most developers choose a single pricing strategy: either they offer their apps as free, aiming at income from advertisements or in-app billing, or they choose to sell their apps at a pre-defined price.

Figure 16(b) shows the CDF of the number of unique categories for which each developer create apps. We see that developers usually focus on one or just few categories: 75% of the free app developers and 85% of the paid app developers create apps belonging to a single category, while 99% of the developers in both cases focus on 1 up to 5 categories. This outcome shows that most developers are also interested in specific app categories.

In Section 6.2 we focused on the income made by paid apps in SlideMe, based on the apps' price and downloads. However, free apps also give opportunities for financial benefits to their developers, mainly due to advertisements (ads) and in-app billing. To validate this argument, we measured the percentage of free apps in SlideMe that include at least one of the 20 most popular advertising networks [33], using the *Androguard* [2] reverse engineering analysis tool, which attempts to detect advertising libraries within APK files. The results of this analysis show that 67% of the free apps in SlideMe include ads, i.e., at least one of the 20 most popular ad networks. The SlideMe web page also indicates whether each app

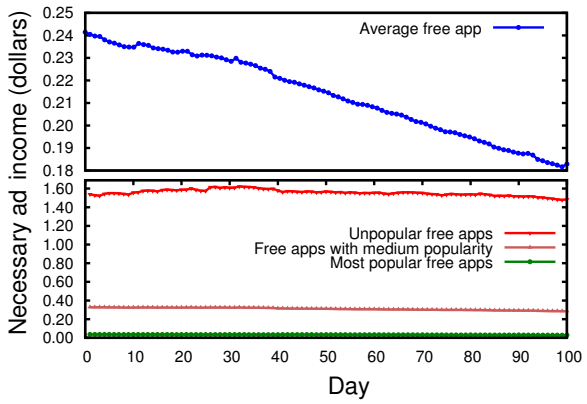



**Figure 17:** ***Free apps with ads may have higher financial benefits than paid apps.*** **An average free app needs about $0.21 per download to match the income of an average paid app. Break-even ad income drops over time because downloads increase faster for free apps. Popular free apps have a lower break-even ad income, thus better chances for increased revenue.**

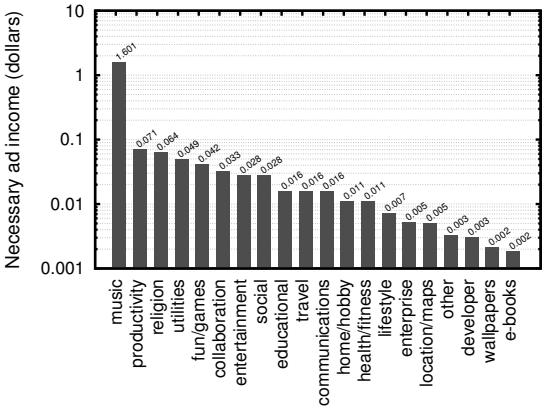



**Figure 18:** ***Some app categories may be more profitable for developers that focus on free apps with ads.*** **A free app in categories like wallpapers or e-books needs to make just $0.002 per download to match the income of a paid app in the same category. In contrast, free apps in the music category need to make more than $1.6 to gain a higher profit.**

(free or paid) contains ads. We observed that this information is generally true, compared with our analysis on the actual APK files for free apps, with just a few exceptions. We also observed that very few paid apps state that they use ads, which implies that there are two different revenue strategies: either paid apps with no ads, or free apps with advertisements.

Using our dataset, we evaluate the two different revenue strategies: (i) paid apps, which require a payment before downloading, and (ii) free apps using advertisements, which make profit after installation. However, we do not have any data about the usage of free apps upon installation, i.e., clicks and impressions, to approximate the actual income from those apps. Therefore, to compare the two alternative revenue strategies, we estimate the ad income that a free app needs to make per download to match the income of a paid app. We can estimate this *break-even* ad income per download for a free app based on the average number of downloads measured for free and paid apps in the SlideMe appstore, and the average price of paid apps, using the following formula:

$$Ad\,Income = \frac{\sum_{i=1}^{N_{paid}} Downloads(i) \times Price(i)/N_{paid}}{\sum_{j=1}^{N_{free}} Downloads(j)/N_{free}} \quad (7)$$

where $N_{paid}$ and $N_{free}$ the number of paid and free apps respectively. We consider only free apps with ads in this analysis.

Figure 17 shows the break-even ad income per download for an average free app as a function of time, for the last three months of our measurement period. We see that overall, an average free app needs to make just $0.21 per download from ads in order to match the income of an average paid app. Moreover, we see that the break-even ad income is dropping over time. This happens because the average downloads of free apps increase much faster than the average downloads of paid apps. Thus, for many apps, where users are expected to spend some time using the application, the ad-based revenue strategy seems more promising for increased income. While the paid apps collect money once per download, a free app may continue to earn money for a long period after the download, depending on app usage.

In Figure 17 we also see the break-even ad income for the most popular free apps, which are the 20% of apps with most downloads, for free apps with medium popularity, which are the next 50% of apps, and for the less popular free apps, which are the last 30% of apps with the fewer downloads. We see that a popular app needs an even lower ad income, just $0.033 per download, to outreach the income of a paid app. Even for the less popular apps, the break-even ad income is $1.56 per download, which is still 2.7 times lower than the average price of a paid app. Therefore, especially for free apps with high and medium popularity, the ad-based revenue strategy seems to be able to offer significantly higher financial benefits than asking for payment only once before downloading.

We performed the same analysis for each app category, as pricing depends on several factors that may vary across different categories. Such factors, for example, include the effort of developing an app (e.g., games in general are more complex to be developed than e-books or wallpaper apps) or the popularity of the app category. Figure 18 shows the break-even ad income per app category. We see that break-even ad income differs among categories. For instance, the highest break-even ad income, roughly $1.6, is observed for music apps, while the lowest is observed for the wallpaper and e-book categories (close to $0.002). This is because music is the dominant category of paid apps in terms of revenue, as we see in Figure 15. Moreover, it is interesting to note that a free app of the fun or games category, which is the second more lucrative category, needs to make only $0.04 per download on average to reach the revenue of a category-equivalent paid app.

# 7. IMPLICATIONS

We believe that our study can be useful for both appstore operators and app developers in several ways. In this section, we briefly discuss some practical implications.

**New replacement policies for improved app caching.** The existence of locality in user downloads, as manifested by the Pareto effect, can help appstore operators to design efficient caching mechanisms for improving the speed of app delivery to end users, e.g., by caching the most popular apps in fast memory or local network caches. To explore the benefits of this locality, we simulated a typical cache with a *Least Recently Used* (LRU) replacement pol-

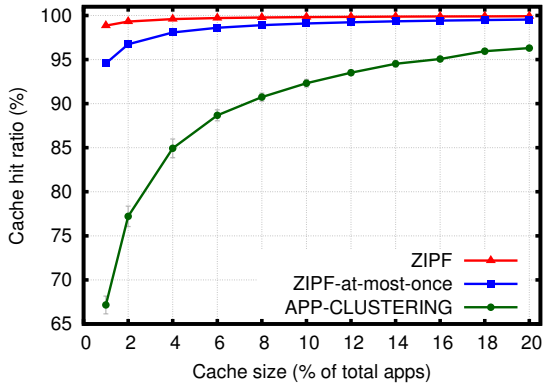



**Figure 19:** ***Clustering-based user behavior has a negative impact on LRU cache performance.*** **Cache hit ratio as a function of cache size for an LRU app cache. Although the LRU cache achieves a high hit ratio, APP-CLUSTERING results in a significantly lower hit ratio than ZIPF and ZIPF-at-most-once.**

icy. We varied the cache size in terms of apps, assuming that all apps have the same size. In our dataset, the average app size is 3.5 MB. For each cache size, the cache was initialized with the respective number of most popular apps. To examine the effect of the observed clustering-based user behavior on a typical LRU cache, we simulated user downloads based on the three different models: ZIPF, ZIPF-at-most-once, and APP-CLUSTERING. We simulated an appstore similar to Anzhi, i.e., with 60,000 apps divided into 30 categories, 600,000 users, and 2 million downloads. We used a ZIPF exponent $z_r = 1.7$ for overall app ranking, and a ZIPF exponent $z_c = 1.4$ for each cluster's app ranking. The percentage of downloads based on clustering was set to 90% ($p = 0.9$). In case of a cache miss, the requested app was replacing the least recently downloaded app from the cache.

Figure 19 shows the cache hit ratio as a function of cache size, which varies from 1% to 20% of the number of total apps (i.e., from 6,000 up to 120,000 apps, which roughly corresponds to 2.1–42 GB of actual cache size), when generating user downloads with each one of the three different models. In general, we see that app caching with an LRU policy achieves a high hit ratio. For example, when caching 10% of the apps, the cache hit ratio is higher than 90% for all models.

However, we see that the workload generated by the APP-CLUSTERING model, which simulates the observed clustering-based user behavior, results in a significantly reduced hit ratio compared to the other two models. While ZIPF-based workload generation (e.g., like web downloads) leads to a hit ratio higher than 99% for all cache sizes, and ZIPF-at-most-once workloads (e.g., like peer-to-peer file sharing downloads) result in a high hit ratio starting at 94.5% and exceeding 99% for cache sizes larger than 10% of the total apps, we see a significantly lower hit ratio for APP-CLUSTERING: from 67.1% up to 96.3% when the cache size varies from 1% to 20% of total apps. Thus, we conclude that although an LRU cache can exploit locality of user downloads to improve the performance of app delivery, the clustering behavior we have identified in this work has a negative effect on the performance of a typical LRU cache. To improve cache performance in the context of app delivery, we believe that new replacement policies should be used, taking into account the clustering-based user behavior.

**Effective prefetching.** Understanding the temporary affinity of users to app categories can help the design of more efficient prefetching methods. As we showed in Section 4, a user that downloads an app from a given category is more likely to download the next few apps from the same category. Thus, the most popular apps from this category that have not been downloaded by the user can be prefetched to a local place in order to improve user experience and app delivery performance.

**Better recommendation systems.** The understanding of user download patterns, like the clustering effect, can help appstore operators to design better recommendation systems, which can benefit (i) apps and developers by providing better opportunities for more suggestions and downloads, and (ii) end users by providing more useful suggestions. A recommendation system can benefit apps and developers by identifying the apps that need to be recommended to increase their popularity, and by identifying users interested in the respective app category. Also, end users can benefit from a better recommendation system by improving their user experience. A typical recommendation system follows a collaborative filtering method [18]. The main idea of this method is to find groups of users who share similar interests. These users can be determined based on their similar app downloads. If an app is downloaded by most users in a group, then it is likely to be of interest for another user in the same group that has not yet downloaded it. Thus, this is a possible suggestion from the recommendation system.

The clustering effect, however, can also suggest apps that have not necessarily been downloaded by users who share similar interests. For example, a user who downloaded an app from a given category may be interested in other popular apps of the same category. Thus, the clustering effect provides a richer set of choices than the limited choices of current recommendation systems, which are mainly based on the common apps downloaded by a set of users. Moreover, users who prefer a large variety of choices will have a better experience with recommendation systems that do not bombard them with the same set of popular apps. In addition, capitalizing on the temporal affinity of users to app categories, as we explained in Section 4, the recommendation system can suggest apps related to the most recent interests of a user, instead of apps related to older downloads.

**Identify and help problematic apps.** Our model of app downloads can be used by appstores to estimate future app downloads based on app popularity. This will enable appstores to pinpoint problematic apps and favor them through better recommendations.

**Larger category diversity.** Another possible implication of the clustering effect is that, currently, users appear to be adherent to one specific category. This may be because they were initially interested in a specific app and they ended up downloading more apps from a single category, based on appstore's suggestions. Hence, there might be a need to expose apps from more categories through recommendations. This may also incentivize developers to offer a wider set of apps across multiple categories.

**Understand and improve app popularity.** Understanding the parameters that affect app popularity is of high interest for developers who want to predict, understand, and most importantly, increase the popularity of their apps. To this end, we provide insights on how user behavior, app category, and other parameters affect app downloads in four popular third-party Android appstores.

**Maximize income.** Understanding which pricing models result in higher revenue can help developers to choose the appropriate pricing policies for increasing app popularity and, eventually, their income. To this end, we examine how cost, app popularity, and app categories affect a developer's income in our dataset. We also compare two popular revenue strategies: paid apps versus free apps with advertisements. Our results indicate that the most profitable revenue strategy depends on app popularity, app category, and ad income per download.

## 8. RELATED WORK

**Workload Characterization.** There are numerous studies focusing on the workload characterization of networks and distributed systems, aiming to improve the understanding of their complex behavior by identifying high-level trends and summarizing statistics that explain significant properties of their workloads. Many of these studies concentrate on web workloads. Breslau et al. [20] show that web requests follow a Zipf-like distribution, and propose a simple model based on this distribution to explain the observed behavior. Crovella and Bestavros [25] show that web traffic exhibits self-similar characteristics, which can be modeled using heavy-tailed distributions. Based on web workload characteristics, Barford and Crovella [19] propose a model for representative web workload generation. In a similar spirit, we show that mobile app popularity follows a Zipf-like distribution with truncated ends, and we propose a model that approximates the observed distribution.

Besides the web, there are also several studies that focus on other systems' workload characterization. Gummadi et al. [34] show that popularity distribution in peer-to-peer file sharing systems deviates from Zipf for very popular objects, which is also observed in our study, and introduce a model that explains this behavior. Cha et al. [21] study YouTube and find that the popularity distribution of user-generated video content follows power-law with an exponential cutoff, which is similar to the app popularity distribution we observe in our study. Power-law distributions with exponential cutoffs have been also identified in other networks, such as live streaming media networks [24], protein, e-mail, actor, and collaboration networks [31]. There are also many works focusing on the explanation and understanding of the power-law distributions [37, 38].

**Systematic studies of mobile apps.** Recently, mobile systems and applications have attracted the interest of researchers that try to understand the behavior and functionality of this emerging technology. Several research efforts were made for collecting and analyzing large sets of mobile apps from multiple marketplaces. These efforts are mainly focused on security and privacy-related analysis, such as malware detection [45], malware analysis [44], overprivilege identification [30], detection of privacy leaks [26, 32], malicious ad libraries [33], and vulnerability assessment [27]. In this work we collected and analyzed a similar large-scale dataset, but our analysis was focused on characterizing and modeling the workload of the monitored marketplaces.

In a work closely related to ours, Xu et al. [42] study the usage behavior of smartphone apps by analyzing IP-level traces from a tier-1 cellular network provider. Their analysis is mostly focused on spatial and temporal locality, geographic coverage, and diurnal usage patterns. On the other hand, our analysis focuses on app popularity and pricing strategies. We also use a different dataset, by systematically crawling four third-party appstores.

Other related approaches focus on mobile traffic analysis as well, but they do not study their relation with mobile applications. Maier et al. [36] perform a study of residential DSL lines of a European ISP and find that mobile devices' traffic is dominated by multimedia content and applications' downloads. Falaki et al. [28] conduct a traffic analysis on 43 different smartphones. They show that browsing contributes most of the traffic, and lower layer protocols impose high overheads due to small transfer sizes. They also study the factors that impact performance and power consumption on smartphone communications and propose improvements.

Falaki et al. [29] analyze user behavior in two different smartphone platforms in order to understand and characterize user activities and their impact on network and battery. They observe a diversity in user patterns, which implies that techniques for improving user experience or power consumption on the average case may be inefficient for a large fraction of users. Wei et al. [41] present a multi-layer system for monitoring and profiling Android apps.

Vallina-Rodriguez et al. [40] characterize the mobile ad traffic by analyzing a large network trace of mobile users. They focus on the volume, frequency, and content of the mobile ad traffic, and they show the distribution of ad traffic among the different platforms, ad networks, and mobile apps. They also study the energy implications of mobile advertising in smartphone devices. Tongaonkar et al. [39] study mobile apps' usage patterns based on mobile ad traffic found in network traces. On the other hand, we study the app download patterns, the app popularity based on the number of downloads from each marketplace, and the relation of app downloads with developers' income.

Another related work by Zhong and Michahelles [43] study app popularity patterns using data from *appaware*, an application that is optionally installed by Android users in their devices. This application captures information regarding installations, updates and removal of apps at real time. Their main outcome is that 10% of the apps account for the 90% of the total downloads, very similar to our results. Our work presents a more detailed study on app popularity distribution, along with a model for app downloads. Moreover, we gathered and analyzed data from all the apps of four different third-party Android appstores, while Zhong and Michahelles have a limited view using data only from the users that have installed the *appaware* application. To the best of our knowledge, our work is the first systematic and large-scale study that focuses on the workload characterization of mobile application marketplaces.

## 9. CONCLUSION

The mobile application market has recently gained increasing popularity, yet the main characteristics and parameters affecting its workload are still poorly understood. In this paper we presented a systematic study of the mobile app ecosystem by collecting and analyzing data from four popular third-party Android appstores. The analysis was focused on app popularity distribution and app pricing. We also briefly discuss possible implications of our findings.

Our results show a highly-skewed Pareto effect where 10% of the apps account for 70-90% of the total downloads. The app popularity follows a Zipf-like distribution with truncated ends, unlike web and peer-to-peer workloads, but similar to user-generated video content. We attributed the observed distribution to a new download pattern we refer to as "clustering effect." The clustering effect implies that users will download the next apps from the same categories as their previous downloads. We validated the existence of clustering effect with a user behavior study, which shows a strong temporal affinity of users to app categories. Then, we introduced a new model for app downloads based on clustering effect, and we validated with simulations that it approximates very close the actual downloads. Our model is able to approximate very well the downloads distributions of all appstores, which implies that the proposed clustering effect is a more general phenomenon rather than a random property of a single appstore.

We also studied the effect of cost on app popularity. We observed that paid apps follow a clear power law distribution, probably due to the more selective behavior of users when they are paying to download an app. We measured how the revenue of paid apps are distributed to developers. The results show that the large majority of developers have very low income, while a very small percentage of them make a significantly higher income. We also found that the number of developer's apps do not increase developer's income. Finally, we found that free apps with ads need to make just $0.21 per download to match the income of paid apps, which seems a more promising strategy especially for popular apps.

## Acknowledgements

We would like to thank our shepherd Chen-Nee Chuah and the anonymous reviewers for their valuable feedback. We would also like to thank Daniel Song for helping with the implementation of crawler prototypes for AppChina and Anzhi. This work was supported in part by the FP7 project SysSec and the FP7-PEOPLE-2009-IOF project MALCODE, funded by the European Commission under Grant Agreements No. 254116 and No. 257007. Antonis Papadogiannakis and Evangelos Markatos are also with the University of Crete.

## 10. REFERENCES

[1] Amazon Appstore. http://www.amazon.com/appstore/.

[2] Androguard. http://code.google.com/p/androguard/.

[3] AndroLib. http://www.androlib.com/.

[4] AppBrain. http://www.appbrain.com/.

[5] Disgo, FAQ. http://www.mydisgo.info/index.php/faq?faqid=5.

[6] Google Play. http://en.wikipedia.org/wiki/Google_Play.

[7] Google Play hits 600,000 apps, 20 billion total installs. http://www.engadget.com/2012/06/27/google-play-hits-600000-apps/.

[8] Google Says 700,000 Applications Available for Android. http://www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices.

[9] iOS and Android Adoption Explodes Internationally. http://blog.flurry.com/bid/88867/iOS-and-Android-Adoption-Explodes-Internationally.

[10] Scrapy framework. http://scrapy.org/.

[11] Selenium Remote Control (RC), a web application testing system. http://seleniumhq.org/projects/remote-control/.

[12] Supported locations for merchants, Google Play. https://support.google.com/googleplay/android-developer/answer/150324?hl=en&ref_topic=15867.

[13] The 1Mobile Marketplace website. http://www.1mobile.com/.

[14] The Anzhi Marketplace website. http://www.anzhi.com/.

[15] The AppChina Marketplace website. http://www.appchina.com/.

[16] The SlideMe Marketplace website. http://slideme.org/.

[17] Vodafone Egypt Ships Android Phones With SlideME App Store. http://www.distimo.com/blog/2009_11_vodafone-egypt-ships-android-phones-with-slideme-app-store/.

[18] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 2005.

[19] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 1998.

[20] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *IEEE International Conference on Computer Communications (INFOCOM)*, 1999.

[21] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's largest User Generated Content Video System. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2007.

[22] J. Cho and S. Roy. Impact of Search Engines on Page Popularity. In *International Conference on World wide web (WWW)*, 2004.

[23] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(3), 2003.

[24] C. P. Costa, I. S. Cunha, A. Borges, C. V. Ramos, M. M. Rocha, J. M. Almeida, and B. Ribeiro-Neto. Analyzing Client Interactivity in Streaming Media. In *International Conference on World wide web (WWW)*, 2004.

[25] M. E. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6), 1997.

[26] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2010.

[27] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *USENIX Security Symposium*, 2011.

[28] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A First Look at Traffic on Smartphones. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2010.

[29] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in Smartphone Usage. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[30] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[31] T. Fenner, M. Levene, and G. Loizou. A Stochastic Evolutionary Model Exhibiting Power-Law Behaviour with an Exponential Cutoff. *Physica A: Statistical Mechanics and its Applications*, 355(2), 2005.

[32] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic Detection of Capability Leaks in Stock Android Smartphones. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2012.

[33] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe Exposure Analysis of Mobile In-App Advertisements. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*, 2012.

[34] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 314–329, 2003.

[35] S. Jansen and E. Bloemendal. Defining App Stores: The Role of Curated Marketplaces in Software Ecosystems. In *International Conference on Software Business (ICSOB)*, 2013.

[36] G. Maier, F. Schneider, and A. Feldmann. A First Look at Mobile Hand-Held Device Traffic. In *International Conference on Passive and Active Measurement (PAM)*, 2010.

[37] S. Mossa, M. Barthelemy, H. E. Stanley, and L. A. N. Amaral. Truncation of Power Law Behavior in "Scale-Free" Network Models due to Information Filtering. *PHYS.REV.LETT*, 2002.

[38] M. E. J. Newman. Power Laws, Pareto Distributions and Zipf's Law. *Contemporary Physics*, 2005.

[39] A. Tongaonkar, S. Dai, A. Nucci, and D. Song. Understanding Mobile App Usage Patterns Using In-App Advertisements. In *International Conference on Passive and Active Measurement (PAM)*, 2013.

[40] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for Commercials: Characterizing Mobile Advertising. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2012.

[41] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: Multi-Layer Profiling of Android Applications. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2012.

[42] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2011.

[43] N. Zhong and F. Michahelles. Google Play Is Not A Long Tail Market: An Empirical Analysis of App Adoption on the Google Play App Market. In *ACM Symposium on Applied Computing (SAC)*, 2013.

[44] Y. Zhou and X. Jiang. Dissecting Android Malware: Characterization and Evolution. In *IEEE Symposium on Security and Privacy*, 2012.

[45] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2012.