GroupTie: Toward Hidden Collusion Group Discovery in App Stores

Zhen Xie
Dept of Computer Science and Engineering
The Pennsylvania State University
zhenxie@cse.psu.edu

Sencun Zhu
Dept of Computer Science and Engineering &
College of Information Sciences and Technology
The Pennsylvania State University
szhu@cse.psu.edu

ABSTRACT

The current centralized application (or app) markets provide convenient ways to distribute mobile apps. Their vendors maintain rating systems, which allow customers to leave ratings and reviews. Since positive ratings and reviews can lead to more downloads/installations and hence more monetary benefit, the rating systems have become a target of manipulation by some collusion groups hired by app developers. In this paper, we thoroughly analyze the features of hidden collusion groups and propose a novel method called *GroupTie* to narrow down the suspect list of collusive reviewers for further investigation by app stores. As members of a hidden collusion group have to work together more frequently and their ratings often deviate more from apps' quality, collusive actions will enhance their relation over time. We build a relation graph named *tie graph* and detect collusion groups by applying graph clustering. Simulation results show that the precision of GroupTie approaches to 99.70% and the recall is about 91.50%. We also apply our method to detect hidden collusion groups among the reviewers of 89 apps in Apple's China App Store. A large number of reviewers are discovered belonging to a large collusion group and several small groups.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: [Security and Protection]; D.2.8 [**Software Engineering**]: Metrics

Keywords

App Stores; Collusion Groups; Tie Graph; Correlation Coefficient; Clusters

1. INTRODUCTION

To help customers find high quality apps, app stores like Apple's App Store, Google Play allow customers to write ratings and reviews for the apps they have installed, and display average ratings and list reviews in their stores. Under this centralized environment, ratings and reviews are critical information to distinguish high quality apps from low quality ones. Positive ratings and reviews will potentially lead to more downloads/installations and more monetary benefit. Moreover, app store vendors often show various ranking charts on the front page and apps with higher ranking will attract more attentions and more downloads. Specifically, their ranking algorithms [8] take review ratings as an important factor. As such, often one would have the incentive to promote one's own app while demoting the competitors' apps. In reality, this could be achieved through hiring a group of users to give very high ratings (e.g., 5) to one's own app while giving very low ratings (e.g., 1) to the competing ones.

Recently, a number of companies have been found conducting their businesses on promoting apps (e.g., itunestop, appqibu). As advertised in their websites (www.itunestop.com[1], www.appqibu.com), these companies claim that they are able to move the ranking of an app into top 5, top 10, or top 50 in a few hours, subject to how much one is willing to pay. The way they achieve this goal is by hiring a large number of users or registering multiple accounts that collectively offer false reviews and ratings. In fact, to combat these fraud apps reviews and ratings, app stores have released an announcement on February 2012 warning the developer not to manipulate the App Store chart rankings [11]. Indeed, when we looked at the top 10 apps in Apple Store China on May 21, 2012, we found two of them clearly had their ratings manipulated. Clearly, as the key part of the smart phone ecosystem, the centralized marketplaces could be damaged by such collusive misbehavior.

In this paper, we aim to discover hidden collusion groups and systematically analyze the problems in App Stores. Specifically, we are searching for answers to the following questions: *Do hidden collusion group actually exist in current app stores? what are the characteristics of collusion groups? how to discover the hidden collusion groups efficiently and accurately?* These questions are concerned by platform vendors. Correct answers will guide the vendors to clean up app stores and catch developers who have manipulated the feedbacks. It also can help customers decide whether to trust ratings and reviews of an app. As for developers, it is useful to monitor whether their apps' feedbacks are manipulated by opponents or not.

[1]This website has been closed recently.

WiSec'14, July 23–25, 2014, Oxford, UK.
Copyright 2014 ACM 978-1-4503-2972-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2627393.2627409.

Table 1: Confusion matrix of rating behaviors

Quality \Rating	Positive	Negative
High	Honest rating	Demoted rating
Low	Promoted rating	Honest rating

To discover collusion groups, the most difficult task is to identify group members. Salehi-Abari et al. [16] defined collusion as "A collaborative activity that gives to members of a colluding group benefits they would not be able to gain as individuals". The definition shows two essential characteristics of collusion groups. One is that members of a collusion group need to work together to fulfill their purposes. They have to rate together more frequently than independent reviewers because they only hold a small portion of total accounts. The other one is that their ratings consistently deviate to the same side from apps' actual quality; for example, group members usually provide high ratings to promote an app or low ratings to demote it. But for honest reviewers, their deviations are distributed more randomly.

Based on the above observations, we propose a novel method called *GroupTie* to identify group members. We first prove that correlation coefficient between the variation of average ratings in a time period (e.g., weekly) and the variation of its reviewer numbers in each period across different versions is equal to zero if all the reviewers rate independently. Then, we design a method to estimate apps' quality considering their correlation coefficient. We further calculate the pairwise tie strength between users and build a tie graph to represent reviewers' relation. By applying graph clustering on tie graph, we can classify all the nodes to several clusters which are considered to be collusion groups. Simulation results show that the precision of GroupTie approaches to 99.70% and the recall is about 91.50%.

We have further applied our method to discover collusive reviewers in Apple app store. On May 21th, 2012, we collected 200 apps from its China market, including 100 top paid apps and 100 top free apps. GroupTie identified $8,853$ reviewers among all $818,545$ reviewers as possible collusion group members, which account for 1.08%. We also find that 3,677 reviewers belong to a large group, which contains $2,652$ members, and several small groups.

2. PRELIMINARIES

2.1 Rating Behaviors

For honest reviewers, we assume their ratings are *independent*. An independent rating means its variance is independent of other ratings. As for collusion group members, their ratings are correlated and usually deviate far away from the quality. For example, most of their ratings are the same, irrespective of the app's actual quality.

All rating behaviors can be classified into three types w.r.t. apps' quality. As shown in Table 1, they are *honest rating*, *promoted rating* and *demoted rating*. Honest rating, denoted by H, is to leave ratings around apps' actual quality. Demoted rating, denoted by D, is to rate much below apps' actual quality. Promoted rating, denoted by P, is to rate much above than apps' actual quality.

The above rating behaviors form the basic behaviors set $\{H, D, P\}$. We define *role* as a combination of different behaviors and the possible roles are enumerated in Table. 2. Every reviewer chooses a strategy on what kind of role she/he will follow each time. Similar to the player's strategy in game theory [4], there are two types of strategies. One is the pure strategy, which means that a reviewer chooses a specific behavior from the behaviors set to follow. For example, to promote an app, the reviewer chooses a behavior like P and leaves a high rating. The other type of strategy is a mix strategy, which means that the reviewer chooses actions based on probabilities. For example, in the same scenario, the reviewer may choose the behavior H with probability 20%, D with probability 10% and P with probability 70%. Clearly, the pure strategy is a special case of the mix strategy. In this paper, we adopt the pure strategy to simplify the definition of attack models.

2.2 Attack Model

To promote an app, members of a collusion group will collaborate and rate high no matter how bad its real quality is. Similarly, to demote an app, they work together to provide low ratings. What is more, some members could randomly act as honest reviewers to hide their purposes. These collective rating actions neglecting apps' real quality are called *collusion attacks*. According to the roles described in Section 2.1, the possible roles for collusion group members are $\{(0, 0, P)$, $(0, D, 0)$, $(0, D, P)$, $(H, 0, P)$, $(H, D, 0)$, $(H, D, P)\}$, as shown in Table. 2.

In this paper, we classify all the collusion attacks into three types: promotion only attack, demotion only attack and orchestrated attack.

Promotion only attack (PoA): The collusion groups only promote apps' rating by providing higher ratings than apps' quality. The roles of group members could be $\{(0, 0, P)$, $(H, 0, P)\}$.

Demotion only attack (DoA) The collusion groups only demote apps' rating by providing lower ratings than the real quality. The roles of group members could be $\{(0, D, 0)$, $(H, D, 0)\}$.

Orchestrated attack (OA) The collusion groups promote the target apps and demote the opponents of the target apps at the same time. The possible roles of group members are $\{(0, 0, P)$, $(0, D, 0)$, $(0, D, P)$, $(H, 0, P)$, $(H, D, 0)$, $(H, D, P)\}$.

For example, as shown in Fig. 1, reivewer1 and reviewer2 take roles $(H, 0, P)$ and $(0, 0, P)$, respectively and they together perform PoA against App2. Reviewer3 and reviewer4 take roles (H, D, P) and $(0, D, 0)$, respectively and they together perform DoA against App4. Reviewer5 and reviewer6 take roles (H, D, P) and $(0, D, P)$, respectively and they perform both PoA and DoA. All the reviewers can form a group to single perform an OA.

In this paper, we aim to discover attackers that form collusion groups to achieve their goals like raising average scores. We do not target at detecting other types of attackers like independent attackers, one-time attackers, or random attackers. Independent attacker is the reviewer that attacks an app on his/her own. Considering the massive number of reviewers for an app, independent attack has little influence on the app's overall score. One-time attackers are those promoting or demoting an app only once. For example, a developer might ask her friends and relatives to promote her app. Theoretically, there could also be attacks launched by a collusion group that behaviors randomly. For example, they may demote an app in the first week and then promote

Table 2: Possible reviewer roles

role	Role description
(0, 0, 0)	A reviewer who never provides ratings.
(0, 0, P)	A reviewer who always provides positive ratings.
(0, D, 0)	A reviewer who always provides negative ratings.
(0, D, P)	A reviewer who will only provide positive ratings to his partners' apps and negative ratings to his opponents' apps.
(H, 0, 0)	An honest reviewer.
(H, 0, P)	A reviewer who will provide positive ratings to his partners' apps and honest ratings to others.
(H, D, 0)	A reviewer who will provide negative ratings to his opponents' apps and honest ratings to others.
(H, D, P)	A reviewer who will provide positive ratings to his partners' apps, negative ratings to his opponents' apps and honest ratings to other apps.

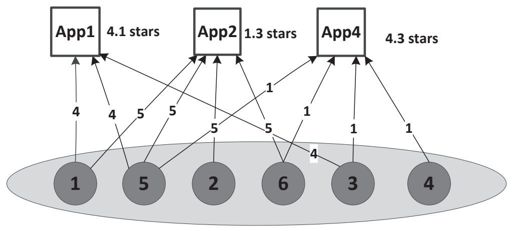


Figure 1: Collusion Attack. App1 and App4 are high quality apps and App2 is a low quality app. Several groups have been hired by the developers to promote App2 or hired by the competitors of App4 to demote App4.

it in the following weeks, for whatever reasons. We will not address these types of attacks in this work.

Note that in our work we do not claim the detection results are 100% accurate due to the extreme difficulty of getting the ground truth. Rather, our algorithm output can provide a much-narrowed-down suspect list for further investigation by app stores. App stores may use additional evidences which we do not have (e.g., information about reviewer accounts) to further pinpoint the abused apps and the collusion groups.

3. GROUPTIE

3.1 Overview

User accounts in app stores (e.g., Google Play or Apple i-Tunes) are bound with hardware devices (e.g., smartphones or tablets), so collusion groups are only likely to account for a small portion among all user accounts. Hence, they often have to collaborate in order to impact the rating of an app significantly. More specifically, their collaboration has two essential characteristics. First, for either promotion or demotion, they provide similar ratings deliberately and collectively. Their ratings significantly deviate from the real quality in the same way, e.g., giving 5 to low quality apps of 1 or 2 stars or the other way around. But for an honest reviewer, his ratings' deviations are likely to be distributed randomly. Even though for some special apps, an individual user's ratings may have similar deviation skewness with collusion group members, such a phenomenon would not happen for most of the apps.

Second, the collaboration of collusion group members could occur many times; consequently, they have a much higher probability to rate the same set of apps than any other individual reviewers. In general, co-rating (i.e., rating the same apps) is not common. To see this, we collected 553,005 reviewers from Apple's China App Store and we found that 0.163% pairs of the reviewers had two commonly rated apps and this number decreases to 0.012% for three commonly rated apps. As illustrated in Fig. 2, co-rating probabilities decrease sharply with the increase of the number of commonly rated apps.

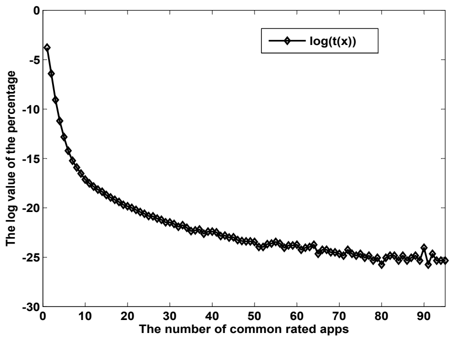


Figure 2: The probability of co-rating apps. Here the x-axis represents the number of commonly rated apps, and the y-axis represents $ln(t(x))$ where $t(x) = \frac{\#\text{pairs of reviewers co-rating } x \text{ apps}}{\text{total pairs of reviewers}}$

Because of the above characteristics, the relations of group members are increasingly strengthened when they co-rate more and more apps, and eventually their relations become very close. Detection of group members is to detect such close relations. Here we outline the three challenges to be solved for collusion group discovery.

- First, how to define and quantify users' relations and model their rating skewness?
- Second, how to discover collusion groups from pairwise relations?
- Third, how to improve detection accuracy under the influence of biased reviewers, adaptive attackers, version updating, etc.? For example, biased reviewers might be misidentified as attackers, hence decreasing the precision (See Section. 3.2). Adaptive attackers might try hard to disguise their roles and avoid being exposed. It would raise the recall (See Section. 3.2). Besides, developers usually update their apps and upload new versions for downloading. Some of the versions might include good features and thus will receive better reviews. Some of them might bear bugs acci-

dently and get low ratings. This will increase the complexity of discover collusion attacks (See Section.3.3.2).

Next, we propose a model to deal with the above three challenges. We borrow the concept of *tie* [10] to represent the relation between two reviewers and construct *tie graph* to represent the relations among all reviewers. Then, we propose a novel method to estimate apps' quality. Finally, given tie graphs, we apply a k-clique communities clustering algorithm to discover collusion groups.

3.2 Tie and Tie Graph

DEFINITION 1. ***Tie*** *is the relation between two reviewers.*

In this paper, *tie* represents the possible relation between two reviewers, where the term *relation* means the probability of two reviewers belonging to the same collusion group.

DEFINITION 2. ***Deviation*** *measures how much a rating deviates from an app's real quality.*

Formally, let the lower case letter denote a reviewer and the upper case letter denote an app. $R(i, K)$ represents the rating of reviewer i to app K and $Q(K)$ denotes the real quality of app K (we will show how to estimate $Q(K)$ later). $Dev(i, K)$ denotes the deviation of reviewer i's rating to the quality of app K and it follows Eq. 1.

$$Dev(i,K) = \begin{cases} 0 & R(i,K) \text{ does not exist} \\ R(i,K) - Q(K) & \text{Otherwise} \end{cases} \tag{1}$$

DEFINITION 3. ***Tie Strength*** *measures the closeness of a relation.*

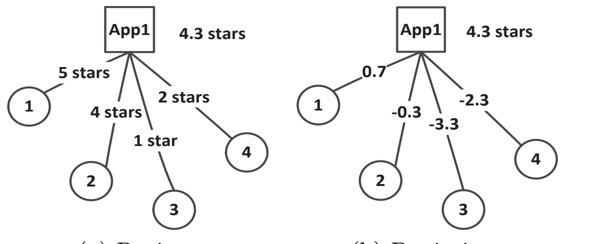
The tie strength between reviewer i and reviewer j is denoted by $Tie(i, j)$, which follows Eq. 2

$$Tie(i,j) = \sum_{K=0}^{N} Dev(i,K) * Dev(j,K), \tag{2}$$

where N is the number of commonly rated apps by i and j. For members of a collusion group, their tie strengths would be high positive values due to their common rating actions. Based on tie strength, we classify ties into three types: *strong ties*, *weak ties* and *absent ties* by setting a tie threshold (e.g., 16 in our experiments). A strong positive tie has tie strength larger than the threshold, which indicates attraction between two reviewers. A strong negative tie has tie strength lower than the negative threshold (e.g., -16), which indicates repulsion between two reviewers. Weak ties are neutral ties with low tie strength (e.g., between -16 and 16), which indicate the relation is uncertain between two reviewers. Absent ties means the relation information between two reviewers is missing.

DEFINITION 4. ***Tie graph*** *is a weighted undirected graph which contains all the pairwise relations among all reviewers. The vertex set is all the reviewers and the edges are the ties between them.*

For example, as shown in Fig. 3 and Fig. 4, four reviewers rate a high quality app and a low quality one, respectively. Among them, two reviewers are honest and the other two are members of a collusion group. Honest reviewers give ratings around the quality, and collusive ones perform promotion and demotion, respectively. We can calculate the tie strength for each pair of them.

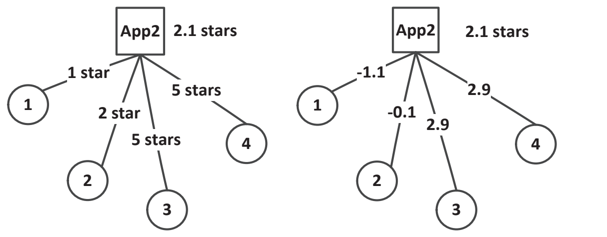


(a) Ratings (b) Deviations

Figure 3: DoA toward an app with quality 4.3 stars.

By following Eq. 2, we can calculate the tie strengths in Fig. 3.

$Tie(1,2) = -0.21$ $Tie(1,3) = -2.31$ $Tie(1,4) = -1.61$
$Tie(2,3) = -0.99$ $Tie(2,4) = 0.69$ $Tie(3,4) = 7.59$



(a) Ratings (b) Deviations

Figure 4: PoA toward an app with quality 2.1 stars.

Similarly, we can get the tie strengths in Fig. 4.

$Tie(1,2) = 0.11$ $Tie(1,3) = -3.19$ $Tie(1,4) = -3.19$
$Tie(2,3) = -0.29$ $Tie(2,4) = -0.29$ $Tie(3,4) = 8.41$

Finally, by combining the tie strengths generated in both figures, we can get the overall tie strengths between each pair of the four reviewers. We can further generate the tie graph shown in Fig. 5.

$Tie(1,2) = -0.1$ $Tie(1,3) = -5.5$ $Tie(1,4) = -4.8$
$Tie(2,3) = -1.28$ $Tie(2,4) = 0.4$ $Tie(3,4) = 16$

Obviously, the tie between two members in a collusion group is the strongest and the other ties can be considered to be weak ties compared to the strong positive ties.

In practice, biased reviewers who always rate high/low may form two different types of strong ties: one case is that when they happen to rate the same set of apps with other biased reviewers, and the other case is when they rate the same set of apps as a collusion group do. In the first case, these biased reviewers form a group. Considering the chance of co-rating illustrated in Fig. 2, the size of the group formed by chance will not be very large, and we can easily filter them out by the algorithm proposed in Section. 3.4. In reality, biased reviewers are prone to rate popular apps. However, popular apps are usually not the targets of collusion groups because popular apps have many more reviewers than a collusion group has. Thus, the attack has little influence. But theoretically, the second case could happen and

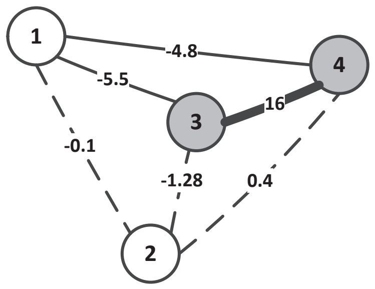


Figure 5: The tie graph including all the nodes. Thick solid line represents a strong positive tie, thin solid line represents a strong negative tie, and dashed lines represent weak ties.

these biased reviewers would be misidentified as attackers. This case will cause precision to decrease. However, if we further check other features mentioned in Section. 4.2.1, like whether they have similar review history, consecutive reviewer IDs as attackers, other type of account information, biased reviewers can also be identified and removed from suspect lists.

On the other hand, if knowing GroupTie beforehand, some attackers might choose adaptive strategies to avoid forming strong ties. For example, each attack of a collusion group will increase tie strength between any two group members. For any two of them, they have to rate another app and leave two totally different ratings (i.e., one is high and the other is low) to annihilate the past impact on tie strength. As a result, if a collusion group has n members, after co-rating a single app, they have to rate $\binom{n}{2}$ different apps to reset their tie strengths caused by such a co-rating, which is very difficult to execute in reality. For example, if a collusion group has 200 members, they have to choose another 20100 apps and build negative ties to cancel out previous positive ties. This would remarkably increase the cost of their organization and what is worse, this strategy could also lead to the risk of exposing their roles.

3.3 App Quality Estimation

To calculate tie strength, we need to know apps' real quality. However, in reality, app's quality measurement is subjective, and its score displayed in the app store could have been severely skewed by attacks. To estimate apps' quality, our idea is to first detect the abnormality of the rating behavior for an app and then remove such abnormality. In this section, we first prove that there is no linear relation between the average rating and the number of reviewers in each period (e.g., in each week) when no attacks exist. Then, we propose a method to estimate apps' quality.

3.3.1 Correlation Coefficient

Correlation coefficient is a measurement of the linear relationship between two variables. If it is a positive value, the increase of one variable is likely to lead to the increase of the other one, and so is the decrease. That is, their variances are synchronous. Similarly, if the value is a negative one, their variances are opposite. Here we adopt Pearson's correlation coefficient [15] to measure the linear relationship between two variables. It is denoted by $r(Y, X)$ for variables Y and X, as shown in Eq. 3.

$$r(Y,X) = \frac{Cov(Y,X)}{\sigma_Y * \sigma_X} = \frac{E(Y-u_Y)(X-u_X)}{\sigma_Y * \sigma_X}, \qquad (3)$$

where $Cov(Y, X)$ denotes the covariance of X and Y. u_X and u_Y are the mean values of variables X and Y, and σ_X and σ_Y represent the variances of X and Y, respectively.

In this work, we divide time into periods of certain unit, e.g., weeks. Suppose an app has been updated a few times, and each version k has the lifetime of kn periods. Let $\bar{Y}_{ki}$ be its average rating for period i *w.r.t.* version k. Its average ratings are denoted by $\bar{Y}_k = \{\bar{Y}_{k1}, \bar{Y}_{k2}, ..., \bar{Y}_{kn}\}$ with $u_{\bar{Y}_k}$ being the overall average. The number of reviewers during each period is a random variable denoted by $X_k = \{X_{k1}, X_{k2}, ..., X_{kn}\}$ with the overall average u_{X_k}. Then, the correlation coefficient between average rating and number of reviews in each period is $r(\bar{Y}_k, X_k)$.

3.3.2 Correlation Coefficient in Ideal Scenario

Intuitively, if all the reviewers of an app rate independently, then the average rating by one group of its reviewers should be about the same to that by another group if the app's quality does not change (i.e., there is no version update). In other words, the increase or decrease of reviewers' number should not change the average rating. From a different perspective, since each rating reflects the app's quality, it can be considered as a sample on app's quality. As these samples are independent and reflect the same app's quality, it is reasonable to assume they follow the same distribution and this scenario is named the *ideal scenario.*

In the ideal scenario, it can be proven that no linear correlation exists between the average rating (sample mean) and the number of reviewers (sample size). Thus, we have the following theorem.

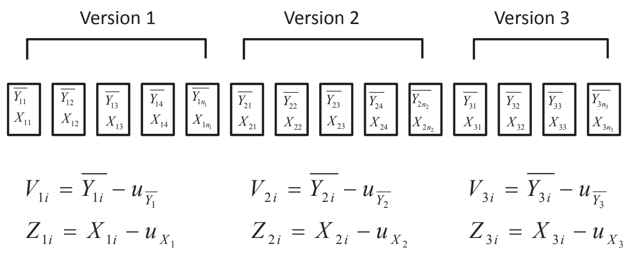
THEOREM 1. *If the ratings of an app* w.r.t. *version k are fully independent and have identical distribution, the correlation coefficient between average rating $\bar{Y}_k$ and number of its reviewers X_k in each period is equal to zero when X_k is large enough.*

PROOF. Let us first divide time into periods of certain unit, e.g., weeks, and suppose there are totally n periods for an app. The average rating of each period is a random variable denoted by $\bar{Y}_k = \{\bar{Y}_{k1}, \bar{Y}_{k2}, ..., \bar{Y}_{kn}\}$. The mean value of $\bar{Y}_k$ is denoted by $u_{\bar{Y}_k}$. The number of reviewers during each period is a random variable denoted by $X_k = \{X_{k1}, X_{k2}, ..., X_{kn}\}$ with mean value u_{X_k}.

Let $Y_{ki} = \{Y_{ki_1}, Y_{ki_2}, ..., Y_{ki_m}\}$ denote the sequence of independent and identical distributed ratings in the ith period having mean u and variance σ^2. $\bar{Y}_{ki}$ is the average rating in this period, which is expressed in Eq. 4. The number of reviewers in ith period is denoted by Y_{ki}.

$$\bar{Y}_{ki} = \frac{Y_{ki_1} + Y_{ki_2} + ... + Y_{ki_m}}{m} \qquad (4)$$

According to the central limit theorem, $\bar{Y}_{ki}$ follows a normal distribution $N(u, \frac{\sigma}{\sqrt{m}})$ given that m is large enough. That is to say, if $Z_{ki} = \bar{Y}_{ki}|X_{ki}$, Z_{ki} will follow a normal distribution $N(u, \frac{\sigma}{\sqrt{m}})$ and $u = E(Z_{ki}) = E(\bar{Y}_{ki}) = E(\bar{Y}_k) = u_{\bar{Y}_k}$. The probability density of Z_{ki} is denoted by $f(Z_{ki})$



Version 1
Version 2
Version 3
$V_{1i} = \overline{Y_{1i}} - u_{\overline{Y_1}}$
$V_{2i} = \overline{Y_{2i}} - u_{\overline{Y_2}}$
$V_{3i} = \overline{Y_{3i}} - u_{\overline{Y_3}}$
$Z_{1i} = X_{1i} - u_{X_1}$
$Z_{2i} = X_{2i} - u_{X_2}$
$Z_{3i} = X_{3i} - u_{X_3}$

Figure 6: One example of version updates.

and $f(Z_{ki}) = f(\bar{Y}_{ki}|X_{ki})$.

$$\begin{aligned} Cov(\bar{Y}_k, X_k) &= E[(\bar{Y}_k - u_{\bar{Y}_k})(X_k - u_{X_k})] \\ &= E(\bar{Y}_k X_k) - u_{X_k} E\bar{Y}_k - u_{\bar{Y}_k} E X_k + u_{X_k} u_{\bar{Y}_k} \\ &= E(\bar{Y}_k X_k) - u_{X_k} u_{\bar{Y}_k} \end{aligned} \tag{5}$$

$$\begin{aligned} E(\bar{Y}_k X_k) &= E(X_k(\bar{Y}_k|X_k)) \\ &= \textstyle\sum_{i=1}^{n}(X_{ki}p(X_{ki})\sum_{j=1}^{m}((\bar{Y}_{kj}|X_{ki})p(\bar{Y}_{kj}|X_{ki}))) \\ &= \textstyle\sum_{i=1}^{n}(X_{ki}p(X_{ki})(\bar{Y}_{ki}|X_{ki})p(\bar{Y}_{ki}|X_{ki})) \\ &= \textstyle\sum_{i=1}^{n}(X_{ki}p(X_{ki}))\int_{Z_{ki}}(Z_{ki}f(Z_{ki}))d_{Z_{ki}} \\ &= \textstyle\sum_{i=1}^{n}(X_{ki}p(X_{ki})EZ_{ki}) \\ &= u_{\bar{Y}_k}\textstyle\sum_{i=1}^{n}(X_{ki}p(X_{ki})) \\ &= u_{\bar{Y}_k} u_{X_k} \end{aligned} \tag{6}$$

Hence, $Cov(\bar{Y}_k, X_k) = E(\bar{Y}_k X_k) - u_{X_k} u_{\bar{Y}_k} = 0$.

Furthermore, according to Eq. 3, Pearson's correlation coefficient $r(\bar{Y}_k, X_k) = \frac{Cov(\bar{Y}_k, X_k)}{\sigma_{\bar{Y}_k} * \sigma_{X_k}} = 0$.

□

In practice, the quality of an app might be changed due to version updates. As a result, the average rating of one group is probably different to another group who rates over another version. For example, before updating a version with good features, the developer did a lot of advertising. Now for the new version, its average rating, even without promotion attack, could be improved. Another example is mentioned at the beginning that a version of an app is updated with bugs. Honest reviewers will leave low ratings to this version even though reviewers will rate high to other versions. In this cases, the correlation coefficient between average ratings and number of reviewers across different versions is not always equal to zero. To eliminate the influence of version updates, we will need to normalize the average ratings. Specifically, we will redefine correlation coefficient as the relation between the *variation* of average ratings and the *variation* of its reviewer numbers in each period across different versions.

Fig. 6 shows how to calculate the variation of average ratings and the variation of reviewer quantities. Here, an app has been updated three times and thus it has three versions. For both version 1 and version 2, reviewers rated the app for five weeks. For version 3, only four weeks have ratings. Each rectangular in the figure represents one week. We first calculate the average ratings (e.g., $\bar{Y}_{11}$ for the first week of version 1) and the number of reviewers in each week (e.g., X_{11}). For version 1, we then calculate the overall average of the weekly average ratings (e.g., $u_{\bar{Y}_1} = (\bar{Y}_{11} + \bar{Y}_{12} + \bar{Y}_{13} + \bar{Y}_{14} + \bar{Y}_{15})/5$) and the overall average of weekly reviewer numbers (e.g., $u_{X_1} = (X_{11} + X_{12} + X_{13} + X_{14} + X_{15})/5$). Let V_{ki} be the variation of the average rating in period i *w.r.t.* version k, then $V_{ki} = \bar{Y}_{ki} - u_{\bar{Y}_k}$. Similarly, we can derive Z_{ki}, which is the variation of the number of reviewers in period i *w.r.t.* version k, as $Z_{ki} = X_{ki} - u_{X_k}$.

Table 3: The correlation coefficient(CC) of top ten apps. Apps denoted by "NA" has lifetime less than nine weeks as of May 21, 2012 and they do not have enough data to calculate a correlation coefficient.

Rank	App Id	App Name	CC
1	*483583569*	Mobilocation	0.414
2	362949845	Fruit Ninja	-0.175
3	449735650	Where's My Water?	0.023
4	439615801	Plants vs. Zombies(Chinese version)	NA
5	414664715	Order & Chaos $^{@}$Online	-0.008
6	491231653	Richman 4 fun	-0.088
7	400973408	Asphalt 6: Adrenaline	0.036
8	508720652	Quick Call Divert	NA
9	*435728194*	Shou Ji Ling Sheng	0.249
10	449595696	Office Assistant Pro	0.082

Formally, let V represent the variation of average ratings of all the versions where $V = \{V_{11}, V_{12}, ..., V_{ki}, ...\}$ and Z represents the variation of reviewer numbers of all the versions where $Z = \{Z_{11}, Z_{12}, ..., Z_{ki}, ...\}$. Based on Theorem 1, we can further derive the following lemma.

LEMMA 1. *If the ratings of an app are fully independent and have identical distribution, the correlation coefficient between the variation of average ratings V and the variation of reviewer quantity Z in each period is equal to zero.*

3.3.3 Correlation Coefficient in General Scenario

Unlike in the ideal scenario, in general scenario honest reviewers and collusion groups may coexist. For honest reviewers, they usually download apps and write comments after playing them for a while directly from mobile devices. Their experiences are independent and they usually do not refer to other reviews while leaving comments. Besides, even though human herding behavior [6] will influence their decision on buying an app, it has no explicit impact on review contents which express their own experience. For these reasons, the independency of user reviews are still held in general scenario. However, the existence of collusion groups will likely change the correlation coefficient of an app.

Since the correlation coefficient in the ideal scenario is equal to zero, its non-zero value in general scenarios could indicate the existence of collusion attacks as well as the type of collusion attacks (i.e., promotion or demotion). For example, for the top 10 paid apps in Apple App Store of China (as of May 21, 2012), we calculated their correlation coefficients (CC) by setting the time period to one week, as listed in Table. 3. We found that the app named "Mobilocation" (Rank #1) has the highest CC score of 0.414 and the app named "Shou Ji Ling Sheng" (rank #9) has the second highest CC score (0.249). We confirmed that these

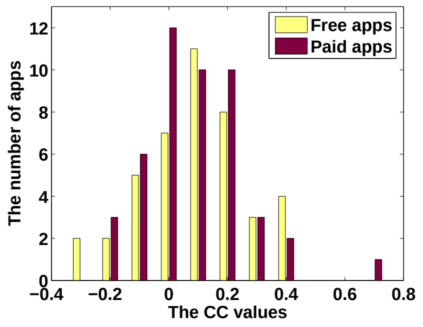


Figure 8: The distribution of CC values (rounded to nearest).

two apps were indeed promoted because of their low quality. They were reported to abuse the rating system in some websites [17][9] and "Mobilocation"[2] has been removed from Apple App Store before May 7, 2013. We further plot the relationship between variance of weekly average rating and variance of reviewer quantities in each week for the first, second, and ninth apps, as shown in Fig. 7. From Fig. 7(a) and Fig. 7(c), it is obvious that the variation of weekly average rating of this app "Mobilocation" and "Shou Ji Ling Sheng" increase synchronously with variation of the weekly number of reviewers. However, this phenomenon does not exist in Fig. 7(b). This app did not show obvious attacks based on our checking.

We have examined the top 100 paid apps and 100 free apps (as of May 21, 2012) and filtered out those apps with lifetimes less than nine weeks (so that our result has more statistical significance). This gave us totally 89 apps and about half for each type of apps. The distributions of these apps' CC values are illustrated in Fig 8. From the shapes we can see that in both free and paid app cases, the distribution approximates to Gaussian Distribution, though skewing slightly to positive of free apps[3]. That is, while majority of the apps look normal, a good portion of them could have been attacked.

We further checked each app looking for signs of promotion or demotion (such signs are discussed in details in Section. 4.2.1). Among the 22 apps with CC larger than 0.2 and 2 apps with CC value less than -0.2, 19 apps were confirmed to be promoted and 2 apps were demoted [4]. Table 4 presents a detailed distribution of CC values for these 89 apps. We can see that large CC values indicate high possibility of attacks. Specifically, a positive CC value indicates a promotion dominating attack whereas a negative CC value indicates a demotion dominating attack.

As we discussed, CC value can reflect the abnormality of an app' rating in most of the time. However, there are

[2]The app's reviews are still accessible in the iTunes through the link `http://ax.phobos.apple.com.edgesuite.net/WebObjects/MZStore.woa/wa/viewContentsUserReviews?id=xxx&pageNumber=0&sortOrdering=1&type=Purple+Software`. Make sure to change your location to China and replace xxx with an app id.

[3]One conjure is that most users are tolerant of free apps' quality. More reviewers will yield a much higher average rating and results a positive correlation coefficient.

[4]Since Apple Inc. had cleaned some of the apps and their reviews, the evidences can no longer be found online.

Table 4: The relation between CC value intervals and the ratio of abused apps in each interval (demoted apps indicated by *)

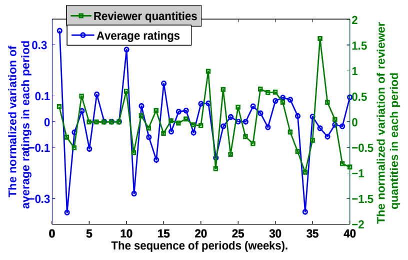
CC Range	Free	Paid	Total
[0.5,1]	0	1/1	1/1
[0.4, 0.5)	1/2	2/2	3/4
[0.3, 0.4)	4/5	2/2	6/7
[0.2, 0.3)	1/3	6/7	7/10
[-0.2, 0.2)	4/30	7/35	11/65
[-1, -0.2)	2*/2	0	2/2

ways to manipulate it and keep it low by attackers. Some sophisticated opponents may act stealthily to avoid the fluctuation of average ratings. They have to avoid generating large amount of ratings in a short time. However, this strategy will not be able to improve average rating quickly, and it also increases their management cost. Some other attackers, for example, can demote the application in the first week of its release and then promote it in the subsequent weeks. If the strength of the promotion and defense are the same, CC of this app will be close to zero and it will not reflect the existence of attack. In these cases, attackers will not achieve their goals and have no much influence on the average ratings. Even though we cannot discover all types of collusive attackers, we can narrow down the suspicious raters as well as increase the bar of the attackers.

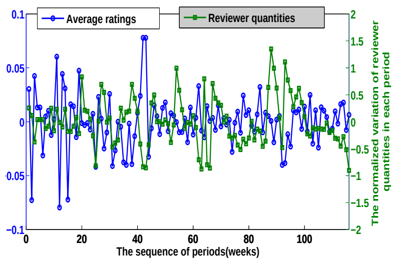
3.3.4 Apps' Quality Estimation

Normally, the square of correlation coefficient (also called *coefficient of determination*, R^2, or R squared) is a statistical measurement of how well a regression line approximates the real data points. In our context, R^2 can help us determine the percentage of average rating variations that have linear relationship with rating population variations in a period. We call the average ratings of such period *abnormal ratings* because normal (honest) rating variation should have no such linear relationship with rating population variation. We aim to estimate an app' real quality by removing the impact (*not removing ratings*) from the abnormality of ratings instead of identifying which rating is abnormal. Note that in our work the goal of app quality estimation is to use estimated scores for tie strength calculation (and then collusion group discovery). We do not use estimated scores as the real scores for apps, so high estimation accuracy is not the goal (To obtain the real scores, we will need to first remove the ratings by identified collusion groups and then recalculate their scores. This is however out of the scope of our current work.) Next we introduce the detail for app quality estimation.

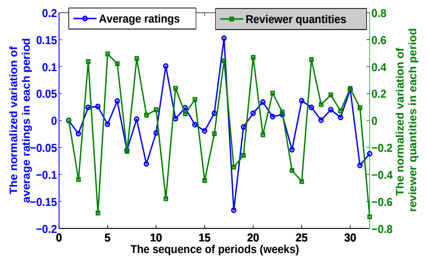
For an app with version k, the abnormal ratings cause its final rating in the app store (i.e., R_k) to deviate from its real quality (i.e., Q_k). When CC is a positive value, R_k is higher than Q_k and Q_k is bounded between [1, R_k]. As shown in Fig. 9, $Q_k \in [1, R_k]$ and $R_k - 1$ is the range of Q_k. Here the range $R_k - Q_k$ is mainly caused by promotion attacks. The more severe the attack is, both R^2 and $R_k - Q_k$ will increase, and vice versa. Let $\frac{R_k - Q_k}{R_k - 1}$ be the normalized value of the range $R_k - Q_k$ indicating the severity of the attack, we can approximate it by a linear function of R^2; that is, $\frac{R_k - Q_k}{R_k - 1} = p * r^2(V, Z)$, where p is a system parameter reflecting the linear relation between $\frac{R_k - Q_k}{R_k - 1}$ and $R^2 = r^2(V, Z)$. On the



(a) App id: 483583569 (Rank #1)



(b) App id: 362949845 (Rank #2)



(c) App id: 435728194 (Rank #9)

Figure 7: The variation of weekly average ratings and weekly number of reviewers. The x-axis is the index of each week ordered by date. The left y-axis represents the number of reviewers and the right y-axis represents the average rating.

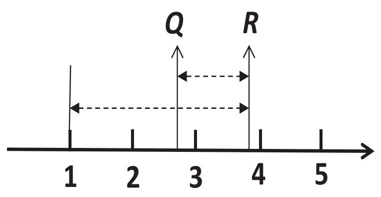


Figure 9: The positions an app's quality Q_k and its final rating R_k if the correlation coefficient $r(V, Z) > 0$.

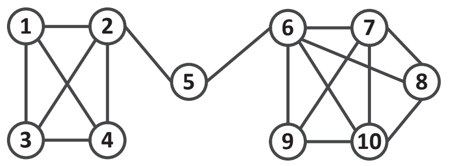


Figure 10: The tie graph of two collusion groups A={1, 2, 3, 4} and B={6, 7, 8, 9, 10}. In this tie graph, all the weak ties and negative ties have been removed.

other hand, when CC is a negative value, R_k is below Q_k. Hence, Q_k is bounded between $[R_k, 5]$ and $5 - R_k$ is the range of Q_k. Here the range $Q_k - R_k$ is mainly caused by demotion attacks. The more severe the attack is, both R^2 and $Q_k - R_k$ will increase, and vice versa. Similarly, we can normalize $Q_k - R_k$ as $\frac{Q_k - R_k}{5 - R_k}$ and approximate it with $R^2 = p * r^2(V, Z)$. Let E_r follow Eq. 8, we can get Eq. 7.

$$p * r^2(V, Z) = \frac{R_k - Q_k}{R_k - E_r} \tag{7}$$

$$E_r = \begin{cases} 1 & \text{if } r(V, Z) \geq 0 \\ 5 & \text{if } r(V, Z) < 0 \end{cases} \tag{8}$$

Finally, to derive app quality Q_k, we can rewrite Eq. 7 as Eq. 9.

$$Q_k = R_k - p * r^2(V, Z)(R_k - E_r) \tag{9}$$

Note that in our system, the system parameter p must not be greater than $\frac{1}{r^2(V,Z)}$ to guarantee the right hand of Formula 7 will not exceed 1. In practice, once we set a threshold CC value (that is, if an app has CC value above the threshold, it will be considered as suspicious), we can set p accordingly. In the simulation and experiments of this work, we set the threshold CC value as 0.25 and $p = 15$. During calculation, whenever $p * r^2(V, Z) > 1$, we will set it to 1. As a result, $Q_k = E_r$. This means, for a clearly promoted app (i.e., $|CC| > 0.25$), we will estimate its score as 1 (according to Formula 9). This may be inaccurate compared to its actual quality (again the purpose of our estimation is not to report actual score for market use). However, it has the advantage of exposing strong ties when two reviewers both gave 5 to this promoted app.

3.4 Collusion Groups Discovery

After estimating apps' quality following Eq. 9, we are able to calculate the pairwise tie strength following Eq. 2 and further build a tie graph to capture all the pairwise relations among reviewers. Like the scenario shown in Fig. 5, the ties between reviewers are different. Specifically, tie strength between collusion group members is accumulated through each group action and their ties will finally become strong positive ties. For honest reviewers, their rating deviations from apps' quality are randomly distributed and thus positive ties are harder to form. Therefore, to detect collusion groups, we only need to care about the strong positive ties and neglect the other types of ties.

Discovery of collusion groups finally becomes partitioning the tie graph which only contains strong positive ties. Here, we exploit two characteristics of collusion groups. First, even though collaboration enhances the relation, group members are not always connected to each other in the tie graph because they do not necessarily all participate in the same set of tasks or always behave the same. For example, as shown in Fig. 10, reviewer8 has never collaborated with reviewer9 even though they both belong to the same group B. It is possible that reviewer8 only promotes apps and reviewer9 only demotes apps. The other three reviewers in group B perform both promotion and demotion.

Second, different collusion groups are not necessarily separated from each other. In reality, they could be connected for some reason. In Fig. 10, group A and group B are connected by reviewer5 because reviewer5 might by chance have rated some common apps with reviewer2 and reviewer6, respectively. As a result, even though there is a strong tie between reviewer5 and reviewer2, they do not belong to the same collusion group. The same reason holds for reviewer5 and reviewer6. Hence, one collusion group does not need to be fully separated from the other groups.

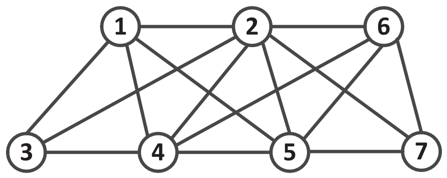


Figure 11: An example of 4-clique-community. There are four 4-cliques which are {1, 2, 3, 4}(c1), {1, 2, 4, 5}(c2), {2, 4, 5, 6}(c3) and {2, 5, 6, 7}(c4). c1 is adjacent to c2 and they share three nodes. Similarly, c2 is adjacent to c3 and c3 is adjacent to c4. Any two 4-cliques can reach each other through such adjacent neighbors.

In this section, considering the above two characteristics, we introduce an algorithm to detect collusion groups. Based on the observation that a typical member in a community is linked to many other members but not all of them, a community can be interpreted as a union of smaller complete (fully connected) subgraphs. Let k refer to the number of nodes in a subgraph; then a subgraph is also called a k-clique. Palla et al. [14] defined a k-clique-community as a union of all k-cliques. All the k-cliques can be reached from each other by way of a series of adjacent k-cliques, where "adjacent" means sharing $k - 1$ nodes. For example, Fig. 11 shows a 4-clique-community where four 4-cliques comprise a community. This definition expresses the essential feature of community: members can be reached through well-connected subsets of nodes. The other parts cannot be reached through k-cliques are probably another k-clique community. Meanwhile, a single node can be in several communities. Thus, the whole graph consists of overlapping communities. Since these features of communities also exist in collusion groups, discovering collusion groups is equivalent to find k-clique-communities on the tie graph with only strong positive ties. It is called k-clique-groups in this work. We can adapt the algorithm introduced in [14] to discover k-clique-groups. The variance of k influences the structure of collusion groups. If $k = 2$, a tie graph is considered to be a collusion group because all nodes are connected. For example, there is only one single collusion group in Fig. 10, which contains all the reviewers. Similarly, a 3-clique-group is given by the union of triangles that can be reached from one to another through a series of shared edges. As k increases, the size of collusion groups shrinks, but on the other hand, it becomes more cohesive since their members have to be part of at least one k-clique. For instance, Fig. 10 is actually a 4-clique-groups which contains group A and group B. Group B contains two 4-cliques and they share a 3-clique.

Since we are looking for k-clique-groups, those groups smaller than k will be discarded because they are unable to complete a successful attack with few group members. These small groups may be formed by biased reviewers who happen to rate the same apps. They can also be some random reviewers even though they rate honestly. Even though we misidentify some honest reviewers as attackers, it is easy to further filter them out with other features discussed in Section. 4.2.1. In practice, we may use GroupTie as the first round of discovery and set a small k to catch as many attackers as possible during the next round of detection.

Table 5: The average value of FPR and FNR under attacks like PoA, DoA and OA.

Attacks	precision	recall
PoA	99.96%	89.61%
DoA	99.38%	92.56%
OA	99.75%	92.34%
Overall	99.70%	91.50%

4. EXPERIMENT AND RESULT ANALYSIS

We have implemented our collusion groups discovery model in JAVA based on Fedora 13 and stored all the data in MYSQL 5.1.56. Two types of experiments are conducted; one is the simulation[5] to evaluate the algorithm and the other one is to detect collusion groups in an actual app store.

4.1 Simulation Study

From the apps that look free of attacks (i.e., without any heuristics mentioned in Section. 4.2.1) in Apple App China Store, we randomly choose 29 apps and consider all of their original ratings honest. We set each period to be one week because it is a general period of our daily life; accordingly, the lifetime of an app is described in terms of number of weeks. We then introduce one collusion group to manipulate N_{apps} apps ($N_{apps} = 4$ by default[6]) by adding positive, negative, or honest ratings. In the simulation, the collusion group controls two parameters: P_{period}, the percentage of weeks in an app's lifetime to introduce attacks; and P_{raters}, the ratio of the number of malicious raters to the population of raters in each week for an app. Once the collusion group chooses an app to attack (i.e., to promote or demote), its members may take one or more attack strategies, as described in Section 2.2: PoA, DoA and OA.

Two standard metrics of classification are used in our evaluation: *precision* and *recall*. In particular, precision measures the percentage of identified attackers which actually do belong to a collusion group and recall represents the proportion of attackers that are truly identified. Under the above attacks, we calculate the precision and recall of GroupTie, as shown in Table. 5. From the table we can see that the overall precision and recall of GroupTie is about 99.70% and 91.50%, respectively. GroupTie has higher precision but lower recall in the PoA case than in the DoA case. Its performance with OA is just between that with PoA and with DoA.

4.2 Discovering Collusion Groups in App Store

4.2.1 Data Description and Detector Settings

We collected and analyzed the data of 200 apps from Apple's China App Store, which consist of 100 top paid apps and 100 top free apps on May 21, 2012. Among them, 111 apps have lifetime shorter than 9 weeks, so they were not included for further study. Finally, we obtained the test set of 89 apps, 818,545 reviewers and 1,042,832 reviews.

Establishing the ground truth on the suspiciousness of these apps and their reviewers is certainly a challenge here.

[5]For the detailed simulation result, please refer to `https://www.dropbox.com/s/jhzhnlalgyeooi0/GroupTie_wisec_full.pdf`

[6]We also tried 8 target apps, which showed similar results.

Manually checking all the reviews to identify possible attacks is not feasible in two aspects: scalability (the number of reviews is too huge) and accuracy (many evidences are hidden). As such, we turn to build an automatic tool that leverages several heuristics: review intensity, skewed ratings, highly similar review history, consecutive reviewer ids to expose suspicious apps and the collusion reviewer groups. Note that we cannot claim the results based on these heuristics are 100% accurate, but our algorithm output can provide a much-narrowed-down list for further investigation by app stores. App stores may use additional evidences which we do not have (e.g., information about reviewer accounts) to further pinpoint the abused apps and the collusion groups. Also note that for the objectiveness of our evaluation, these heuristics were not learned from the 89 apps under test in the next section.

Review Intensity represents the distribution on the number of ratings over each unit time. Basically, if one checks all the reviews of an app and orders them by "helpfulness", one may notice that many same type of reviews are posted in a very short period. For example, among the reviews of the app with id 499814295, most five star ratings were posted on April 28, 2012 and May 22, 2012. Among all the reviews of app 499805269, there were 188 most helpful reviews and 183 of them were five star posted on June 1, 2012. This phenomenon reflects the collaborative rating behavior of collusion groups to make the attacks more effective.

Skewed Ratings means that the distribution of rating scores of an app in a short time period is very skewed compared to its overall score distribution. For example, for the app with id 474429394, its overall rating summary for version 1.5.2 is $(28, 9, 25, 37, 241)$ (rating scores ordered from one star to five star). We also collected its ratings from July 1, 2012 to Aug 3, 2012 and got its rating summary $(1, 0, 0, 3, 136)$ for this time period. It is easy to observe that the ratio of five star to one star ratings increases from $241/28 = 8.61$ to $136/1 = 136.00$. Therefore, the developer might have hired a collusion group to promote its app version 1.5.0. Another example is app 499805269. Its rating summary of version 2.5.6 was $(2, 1, 0, 0, 183)$. If checking all its ratings in Jun 1, 2012, the rating summary was $(0, 0, 0, 183)$. This indicates all its five ratings appeared in one day. However, for normal apps, the ratio does not change a lot in different periods. This phenomenon is due to collaborative promotion which generates massive five star ratings in a short time window to be effective.

Highly Similar Review History means group members have co-rated many common apps in the past. For example, we find a group of 171 reviewers all have rated and nearly only rated apps with ids (499814295, 525948761, 485252012, 496474967, 499805269) in the past. Only a few of them also rated 460351323.

Consecutive Review Id means reviewer IDs of some groups are very close. Even though we do not know the mechanism for id (i.e., Apple ID) generation in iTunes, it is still weird that many reviewers with very close ids reviewed the same set of apps. For example, the following reviewers have commonly reviewed two apps with ids 474429394 and 525378313, respectively.

"212525736, 212525739, 212525752, 212525762, 212525765, 212525781, 212525784, 212525788, 212525795, 212525800, 212525808, 212525811, 212525825, 212525827, 212525834, 212525847, 212525851, 212525857, 212525860, 212525864, 212525903, 212525990, 212525996"[7]

If iTunes generates user ids in an incremental way, this group probably applied these IDs in a very fast way (e.g., through bots). This sign would indicate not only the existence of a collusion group, but also the co-rated apps by these reviewers are probably abused apps.

Removed Apps means some apps have been removed from app stores, due to reasons like rating manipulation as reported by the news.

Table 6: Abuse signs of apps

Sign	# of apps
Reviews Intensity or Skewed Ratings	9
Highly Similar Review History	7
Consecutive Ids	3
Removed Apps	6

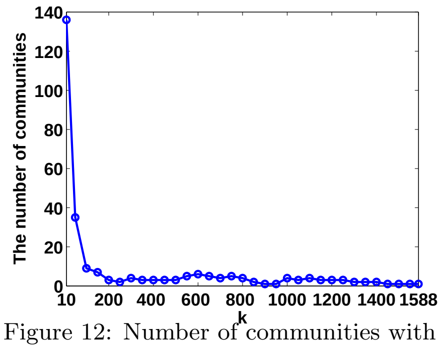


Figure 12: Number of communities with k.

4.2.2 *Findings*

Based on the above heuristics, we implemented an automated tool to check the apps for their signs of being abused. Among the 24 apps (out of the total 89 apps) whose CC values were above 0.2 or below -0.2, we identified 19 of them as suspicious (labeled as abused apps), because they bore one or more heuristics mentioned previously, as shown in Table 6. The relation between CC value intervals and the ratio of suspicious apps in each interval has been shown earlier in Table 4. Furthermore, among $818,545$ reviewers, we found that $8,853$ reviewers (i.e., 1.08%) had strong ties, and they formed $734,848$ strong positive ties. We applied the k-clique-communities algorithm described in Section 3.4. As illustrated in Fig. 12, when k increases from 10 to 100, the number of communities decreases sharply from 136 to 9. When k further increases, the number of communities continues to decrease until to the largest k value (i.e., 1,588). The reason is that when k increases, more and more groups cannot satisfy the k-clique requirement any more and they are not considered as a group.

When k equals to 100, we found $3,677$ reviewers forming 9 groups, which contained one large group with $2,652$ members and 8 smaller groups. We also checked their target apps by counting the commonly rated apps of the group members. As shown in Table 7, the 8 small groups targeted

[7]The review history of a reviewer can be accessed through the link https://itunes .apple.com/WebObjects/MZStore.woa/wa/viewSoftware?id=xxx. Make sure to change your location to China and replace xxx with a reviewer id.

Table 7: Collusion groups discovered by GroupTie.

#	Group size	Target App ID
1	104	363966906, 431194169
2	107	444934666, 370130751
3	145	441216572, 512500671
4	146	324101974, 444934666
5	151	324101974, 483583569
6	180	414478124, 387682726
7	198	324101974, 363966906
8	195	512500671, 387682726
9	2652	324101974, 414478124, 441216572 483583569, 512500671, 370130751 363966906, 444934666

at some apps to do promotion or demotion and they have some specific characteristics. For example, the target apps of group 1 actually belonged to one developer. Group 4 seemed to have two types of tasks: one was to promote 324101974 and the other was to demote 444934666. Group 9 was the largest group and it was almost 13 times larger than the other groups. Similar to group 4, its targets included both demoted (e.g., demoting the products of Tencent Inc. [19]) and promoted apps. Indeed, among all the target apps listed in Table. 7, all were labeled as abused apps previously. This indicates these 9 groups are highly likely collusion groups.

We further examined the behavior of the largest group (group # 9) and made the following observations.

- Attacks concentrated within a few days. For example, when demoting app 444934666, lots of low ratings were posted on 3/30/2012 and 3/31/2012.
- Multiple attacks to the same app were launched by the same group. For example, three separate attacks on app 324101974 were launched at around 12/1/2011, 01/19/212, and 5/14/2012.
- Group members have similar rating behaviors. When some members attacked an app, their rating scores were very close to each other. For promotion, most of them posted the highest rating (e.g., 5 star to app 324101974). For demotion, they often left the lowest rating (e.g., 1 star to app 444934666).
- Attacks often happened at the time of updating a new version. For example, the attack on app 324101974 occurred at the second day (i.e., 12/1/2011) of updating version 6.6. Similar scenarios were also found in other attacks.

Some of these observations conform with the heuristics GroupTie applies, and the others may inspire us to define new features. For example, one may pay special attention to the ratings posted right after the publishing of an app to identify collusion groups.

5. DISCUSSION

Next we discuss several issues related to the limitations and future improvement of GroupTie.

Attacks To GroupTie GroupTie is for detecting the close relation between collusion group members. Clearly, a collusion group, once knowing that GroupTie is in place, may try its best to evade the detection, with different levels of difficulty, complexity, efficiency and cost. For example, they can try strategies like randomizing their ratings or collaborators for different apps to make their attacks more stealthy at the cost of lower attack effectiveness. Clearly, there will be an arms race which could attract more research work in the future. We believe our current work, though as a first step, has greatly raised the bar for attackers. However, if collusion groups can hold enough accounts (e.g., millions), they might use each account only once to evade the discovery of GroupTie. The problem becomes a well-known sybil attack which has been studied a lot [20][21][22].

Computational Complexity and Storage Overhead We note that storage overhead is not an issue here if an app store is going to run our algorithm, because it has the rating/review information already. The computational complexity is a concern if we want to run our algorithm over the entire store, that is, for all raters and all apps. This is because we need to compute tie strength for each pair of users if they have ever co-rated an app and the number of such pairs is huge for an entire store. It will not be possible to compute tie strength for all pairs in physical memory. In practice, one optimization could be to run the algorithm only for suspicious apps and their associated reviewers. Further, we may distribute tie strength computation by leveraging the MapReduce framework to build tie graphs.

Application to Other App Stores GroupTie is designed based on general principles and assumptions, so it should also be applicable to Google Play or BlackBerry App World.

6. RELATED WORK

This section discusses a few most relevant fields.

Clique Detection In the context of cloud computing, collusion attack could break the integrity of the data collected from individual nodes. Du et al. [7] offered a mechanism to pinpoint the malicious service providers by detecting nodes outside maximum cliques. Stab et al. [18] presented a mechanism to detect collusion nodes by exploiting how often they work together in the majority or minority and how often they are in opposite groups. Lee et al. [12] designed an algorithm to find the cliques and furthermore, detect the malicious nodes and decrease their influence on the reputation. These methods were applied in grid computing, and after each task the result about whether a node cheats or not is easy to verify. However, reviewers' ratings to apps are subjective and not binary either. Moreover, the full graph including all the nodes is too large to detect cliques efficiently. GroupTie only builds graph for the suspects who have strong ties, which is much smaller than the full graph.

Maximum Independent Set Araujo et al. [3] proposed a maximum independent set approach for collusion detection in voting pools, aiming at classifying nodes as correct or incorrect. The approach first builds a vote against graph where two node in each edge voted against each other. Since each edge represents the disagreement between the nodes in two ends, they must belong to different groups, collusion group or honest group. Assuming that the largest plurality of nodes that do not vote against each other is correct, the detection of collusion groups is to find the maximum independent set from the votes against graph. Since collusion group members in an app store could rate randomly to hide their roles, the ratings to the non-target app of two members in a group could be totally different like one is "1 star" and the other one is "5 star". Thus, it is not suitable to apply maximum independent set in apps store.

Feature Engineering is a method to extract features of collusion groups and apply them to identify other groups. Mukherjee et al. [13] proposed algorithm which first uses a frequent itemset mining(FIM) [1] method to find candidate groups and employs an eight indicators evaluation system to detect collusion groups. Allahbakhsh et al. [2] offered a similar method employing six indicators to discriminate collusion groups with honest groups and they also built a biclique to represent the relationship between reviewers and products. Beutel et al. [5] proposed a method to catch collusive attackers which have lockstep behavior (i.e., launch attacks in a short time) when generating fraudulent "like" page in Facebook. These methods are based on features of specific collusion groups. Instead, features used in GroupTie are the essential features shared by various collusion groups.

7. CONCLUSIONS

In this paper, we have analyzed collusion attacks by providing falsified ratings in current apps store and proposed a novel method called GroupTie to detect collusion group members. Our simulation results showed that both the precision and recall of GroupTie is about 99.70% and 91.50%, respectively. We also applied our method to discover collusion group members among the reviewers of 200 apps in Apple's China App Store and found that 1.08% of the reviewers were likely collusive attackers belonging to one large collusion group and 8 small groups. The result calls for more attention to the collusion group problem in current app stores.

8. ACKNOWLEDGEMENT

This work was supported in part by NSF grant CCF-1320605 and a Google gift. We also thank the reviewers for helpful comments.

9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.-M.-R. Beheshti, N. Foo, and E. Bertino. Detecting, representing and querying collusion in online rating systems. *CoRR*, abs/1211.0963, 2012.

[3] F. Araujo, J. Farinha, P. Domingues, G. C. Silaghi, and D. Kondo. A maximum independent set approach for collusion detection in voting pools. *J. Parallel Distrib. Comput.*, 71(10):1356–1366, 2011.

[4] T. Basar and G. Olsder. *Dynamic noncooperative game theory*, volume 200. SIAM, 1995.

[5] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, 2013.

[6] Y.-F. Chen. Herd behavior in purchasing books online. *Comput. Hum. Behav.*, 24(5):1977–1992, Sept. 2008.

[7] J. Du, W. Wei, X. Gu, and T. Yu. Runtest: assuring integrity of dataflow processing in cloud computing infrastructures. In *AsiaCCS*. ACM, 2010.

[8] fiksu.com. `http://www.fiksu.com/blog/apple-finally-adding-ratings-ranking-factors-0`.

[9] Geekpark. `http://www.geekpark.net/read/view/158104`.

[10] M. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[11] A. Inc. `https://developer.apple.com/news/index.php?id=02062012a`.

[12] H. Lee, J. Kim, and K. Shin. Simplified clique detection for collusion-resistant reputation management scheme in p2p networks. In *Communications and Information Technologies (ISCIT), 2010 International Symposium on*, pages 273–278, Oct 2010.

[13] A. Mukherjee, B. Liu, and N. Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 191–200, New York, NY, USA, 2012. ACM.

[14] G. Palla, I. Derl̨ényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, June 2005.

[15] K. Pearson. *Mathematical contributions to the theory of evolution*, volume 13. Dulau and co., 1904.

[16] A. Salehi-Abari and T. White. On the impact of witness-based collusion in agent societies. In *Proceedings of the 12th International Conference on Principles of Practice in Multi-Agent Systems*, pages 80–96. Springer-Verlag, 2009.

[17] Sina. `http://tech.sina.com.cn/it/2012-05-10/06197087031.shtml`.

[18] E. Staab and T. Engel. Collusion detection for grid computing. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 412–419. IEEE Computer Society, May 2009.

[19] Tencent. `http://www.tencent.com/en-us/index.shtml`.

[20] G. Wang, M. Mohanlal, C. Wilson, X. Wang, M. J. Metzger, H. Zheng, and B. Y. Zhao. Social turing tests: Crowdsourcing sybil detection. In *NDSS*. The Internet Society, 2013.

[21] G. Wang, C. Wilson, X. Zhao, Y. Zhu, M. Mohanlal, H. Zheng, and B. Y. Zhao. Serf and turf: Crowdturfing for fun and profit. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 679–688, New York, NY, USA, 2012. ACM.

[22] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network sybils in the wild. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 259–268, New York, NY, USA, 2011. ACM.