Exploring the Popularity, Reputation and Certification of User-Generated Software

Hanbing Li

Exploring the Popularity, Reputation and Certification of User-Generated Software

Internship report

Hanbing Li
Institut Telecom - Telecom Bretagne, France
hanbing.li@telecom-bretagne.eu
Guided by: Fabien Dagnat, Gwendal Simon

August 14, 2012

Abstract

User-Generated Content has reshaped the landscape of the Information Marketplace during the last years. Among the content, software is a very impacting class. In comparison to regular resource, the estimation of popularity, reputation and certification of user-generated software in distributed architecture appears to be a challenging task. Furthermore how to use popularity to help user to discover the most suitable software, how to use reputation to allow users to evaluate and be aware of the quality of packages circulating on the Web, and how to use certification to assist user in choosing a safe software, these become very important, but more difficult compared with the traditional software system in a distributed environment.

In this report, basic concepts of popularity, reputation and certification and their state of the art approaches are introduced and analyzed. Then we introduce software deployment system and reveal the impact of popularity, reputation and certification on it. After this, we revisit the current methods and theories of estimation, and propose an adaptive distributed architecture and several algorithms for estimating popularity, reputation and certification. At last, some open issues concerning full distributed popularity and reputation estimation, the establishment of software-related social network and so on are highlighted for future research.

Keywords: Software Deployment System, Package Management, Peer-to-peer Network, Popularity, Reputation, Trust, Certification

Contents

List of Figures

List of Tables

1 Introduction

User-generated content (UGC) refers to content generated by users. Among these content, there is a less popular but very impacting class: *software*. The size of the free and open source software community (two millions sourceforge users[1]) and the number of application store developers (163,405 companies for Apple iOS so far[2]) illustrate the importance of crowdsourced software, i.e. software produced by a large number of loosely coordinated developers. As we all know, software management systems are becoming more and more predominant in our daily lives, because we have to manage a growing number of personal devices offering more and more services. However, the deployment of crowdsourced software is a frustrating and error-prone task.

The current approach adopted by most software management systems is to rely on a single distributor, who collects packages from upstream developers, tests, releases and distributes them through a centralized channel, called *repository*. Figure 1 illustrates the central repository-based architecture. The developers submit the software they developed to the distributor, and the user can only install the software released in the repository.

While with the development of *user-generated software*, the number of packages will be so huge that a traditional software server *e.g. Apple App Store* won't have the capacity to process all the packages. In [1], Zhang et al. pointed out that the repository-based approach can't fulfill the need of user-generated software deployment because of the following reasons:

- managing a large repository for user-generated software is becoming more and more expensive with the increase of capacity. An analysis showed that Apple spend $1.3 billion per year to run its iTunes Store[3].
- approving and releasing third party software through a single channel is time

[1] http://en.wikipedia.org/wiki/SourceForge
[2] http://www.appannie.com/search/
[3] http://www.asymco.com/2011/06/13/itunes-now-costs-1-3-billionyr-to-run/

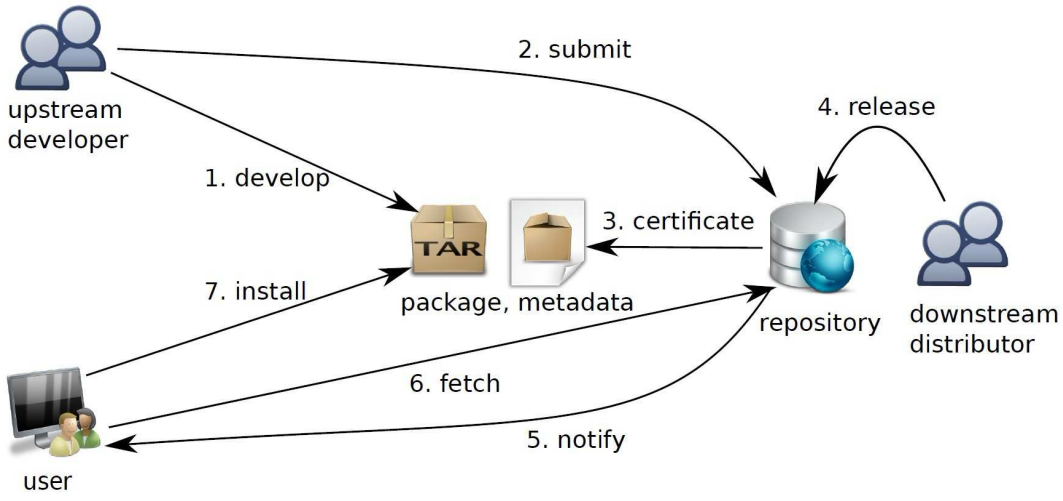

Figure 1: Central repository-based architecture

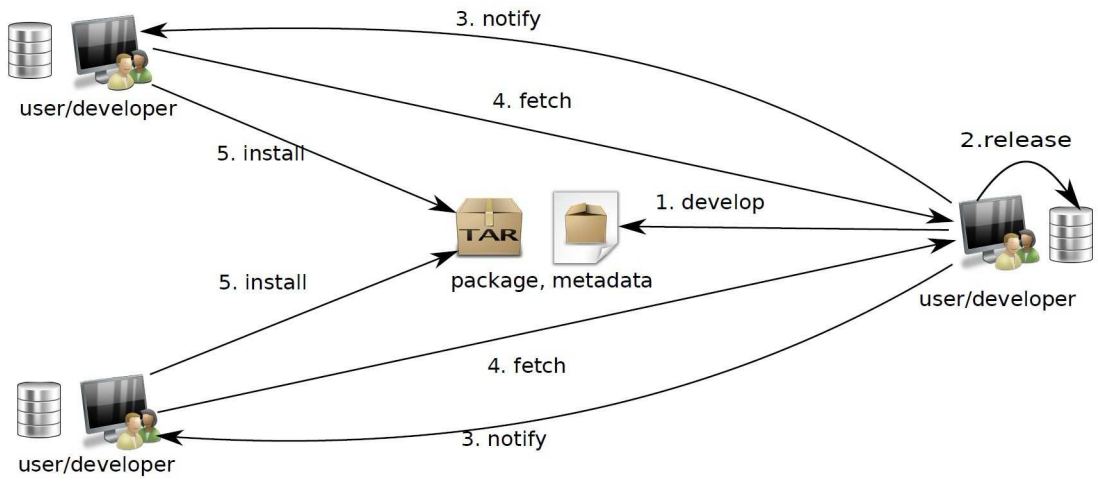

Figure 2: Distributed package management system

consuming. For developers, it increases the time-to-market and degrades the availability of the newest products. And users face security risks due to delayed updates or bug fixes[4].

- relying on a single distributor is a supplier-side approach while software has become a fast moving consumer good. The centralized administration may lead to censorship which is inconsistent with the flexibility of crowdsourced software.

- repository-based approach can not fix the dependency hell. In order to reduce the complexity of dependency resolution, the typical repository-based system, GNU/Linux distribution repositories have to limit their number of packages.

As a conclusion, the central repository has become a limitation in the distribution and management of user-generated packages. So the establishment of distributed software deployment system is essential. We imagine a network consisting of interconnected symmetric peers, atop which all the developers are allowed to publish and distribute new packages and updates at anytime. Both the release and delivery of packages should be handled in a decentralized way. Based on this network, we build up the system *distributed package management network (dPAN)*, and our works all base on it. Figure 2 shows the distributed package management system. Each peer can be user and developer, and can release software. Then each one can download software from his neighbors. The distributed architecture provides compatible and yet more efficient downloading and updating services for packages [2].

In user-generated content, there are two important properties: popularity and reputation. Popularity is the quality of being well-liked or common, or having a high social status. Reputation of a social entity is an opinion about that entity, *i.e.* a result of social evaluation on a set of criteria. Furthermore when the content has to be guaranteed such as for software, certification is needed. Certification refers to the confirmation of certain characteristics of an object, person, or organization. Each of them has their function in software deployment system. For example, popularity can

[4] `http://www.macworld.com/article/1159978/mac_app_store_security_updates.html`

help users to find popular software, especially in a distributed environment where there are millions of user-generated software and these software are stored separately. Reputation can assist the spread of software of high quality. It also can reduce and eventually block the spread of malware, which is software designed to disrupt computer operation, gather sensitive information, or gain unauthorized access to computer systems. This function is particularly important in distributed system because of the complexity of this kind of system; both popularity and reputation can make a contribution towards recommendation system. Certification system can guarantee that the package received by the users has not been altered or corrupted. So we can conclude that popularity, reputation and certification play an important role in software deployment system, and the popularity estimation and reputation and certification system are the essential to software deployment system, especially distributed software deployment system providing a large amount of user-generated software.

However, without the management and control of the central server and with the huge quantity of packages, the mean to ensure security and to judge the quality and popularity of these packages is a very challenging task. Thus finding an efficient and reliable way to help users to safely install useful software is becoming an increasingly important issue. In addition, considering the system itself, we also need to find some approaches to help system store the packages and all kinds of information suitably and reduce the network traffic.

By the way, since the quality of user-generated packages varies drastically from excellent to abuse and spam, to identify high-quality content, an accurate estimation of package popularity and a reputation and certification system are very important to users especially in a open and distributed environment. Moreover, peers play an important role in distributed network, so we also need to consider the estimation of the peer popularity and reputation.

To the best of our knowledge, no previous studies have discussed approach for the estimation of popularity, reputation and certification in distributed software deployment system. But in other domains, there are already some related works. For example, in [3], the authors describe that the popularity of P2P objects can be modeled by a Mandelbrot-Zipf distribution in several autonomous systems domains. [4] proposes PGrid, a reputation system for decentralized networks. A protocol called XRep which is a reputation sharing protocol for Gnutella is proposed in [5]. There are also corresponding security techniques in APT/DPKG and YUM/RPM for the certification [6]. These approaches are all about the estimation of popularity, reputation and certification. However, considering the characteristics of the packages and peers in distributed software deployment system, *e.g.* a package has kinds of different versions, like updated version, certified version, distributed version, etc; a peer not only is a user, but also can be a certifier, a developer, etc, we find that these approaches have their shortcomings and can't be extended to our system. So we need to propose our own approach adapted to distributed software deployment system.

In this report, by analyzing popularity, reputation and certification and software deployment system and revisiting the state of the art theories and algorithms, we put forward the corresponding algorithms of the estimation of popularity, reputation and certification in distributed software deployment system based on the previous approaches. First of all, we design a hybrid distributed network structure which

is composed of a core structured network (*Super-Net*) and many unstructured networks (*Normal-Net*). The structured ring based network forms the backbone, and the unstructured networks are attached to this backbone. This hybrid structure lays the foundations of *dPAN*. The package searching, storage, information collection and communication all base on this structure. Besides, we also propose three main information transmission methods which are used during the procedure of popularity estimation and reputation system. Through our approach, we can estimate the popularity of packages and peers in *dPAN*, and the advantage of our algorithm is that it considers the standoff of accuracy and network traffic. And we can establish a reputation system for the packages and peers in distributed software deployment system with a special reputation rating scheme which is proposed by us. Besides, our reputation system uses the combination methods in reputation collection, aggregation and storage, which has more benefits and avoids more attacks to reputation-based systems. In certification system, we sign both metadata and packages, and combine the code signing and the digital signature of distributors and certifiers; and we can guarantee that the users can download and install the software safely with the aid of our reputation and certification system. Besides, we also discuss what we can do by using the estimation result and some open issues, *e.g.* different levels in certification system, coexistence problem of the original version with the signed versions and so on.

The remainder of the report is organized as follow: in the next section we give the definition of popularity, reputation and certification, and we present software deployment system including its items, functionalities and roles. Section 3 introduces the impact of popularity, reputation and certification on software deployment system. Thereafter, a suitable network architecture and estimation algorithms are proposed in Section 4. At last Section 5 concludes the report and mentions open problems and challenges.

2 Background

In this section, we describe what are popularity, reputation and certification. Then in order to help understand software deployment system, we examine several typical systems, like Apt (for Advanced Package Tool)[5] of Debian[6] and Ubuntu[7], YUM (for Yellowdog Updater, Modified)[8] of Red Hat Enterprise Linux[9] and Fedora[10], App Store (iOS)[11] of Apple[12] and so on. We analyse, synthetize and generalize these current software deployment systems, give the details of them, and introduce the items, the functionalities and the roles in the system.

2.1 Definitions

2.1.1 Popularity

Popularity is defined in the dictionary as the quality or state of being popular, especially the state of being widely admired, accepted, or sought after.

Definition 1. ***Popularity*** *is the fact of being known.*

According to the definition of popularity, we obtain the measure of popularity.

$$The\ popularity\ of\ a\ thing\ (T) = \frac{number\ of\ people\ knowing\ T}{total\ number\ of\ people} \tag{1}$$

This notion is the center of lots of study which are mainly focused on predicting within large set of elements, popularity is of interest to (a) help user to find elements, (b) give confidence on their quality and (c) predict whether a new element may reach success.

2.1.2 Reputation and Trust

Reputation is defined in the dictionary as the estimation in which a person or thing is generally held.

Definition 2. ***Reputation*** *is the fact of being estimated.*

Definition 3. ***Reputation management*** *is the process of tracking an entity's actions and other entities' opinions about those actions; reporting on those actions and opinions; and reacting to that report creating a feedback loop.*

We can get the measure of reputation based on these definition.

$$The\ reputation\ of\ a\ thing\ (T) = \frac{sum(feedback\ of\ the\ estimation)}{number\ of\ people\ estimating\ T} \tag{2}$$

Trust is defined in the dictionary as reliance on and confidence in the truth, worth, reliability, etc., of a person or thing.

[5] http://wiki.debian.org/Apt/
[6] http://www.debian.org/
[7] http://www.ubuntu.com/
[8] http://yum.baseurl.org/
[9] http://www.redhat.com/products/enterprise-linux/
[10] http://fedoraproject.org/
[11] http://www.apple.com/itunes/
[12] http://www.apple.com/

Definition 4. ***Trust*** *is the probability of the event that an individual on whom another individual relies.*

Reputation plays an important role in many domains, like education, business, and online communities. It can give confidence to the entity, and make other entities know whether an entity can be trusted. Reputation is a fundamental instrument of social order.

2.1.3 Certification

Certification is defined in the dictionary as definition by the dictionary) validating the authenticity of something or someone.

Definition 5. ***Certification*** *refers to the confirmation of certain quality of an object, person, or organization.*

The certification generally relies on the reputation of the certification authority.

2.2 Software deployment system

Software deployment is all of the activities that make a software system available for use. Now we introduce software deployment system with three parts: items, functionalities and roles.

2.2.1 Items

The following items are the ones being part of software deployment system.

- *Software* is a collection of computer programs and related data that provides the instructions for telling a computer what to do and how to do it. It is a set of programs, procedures, algorithms and its documentation concerned with the operation of a data processing system.

  System software is computer software designed to operate and control the computer hardware and to provide a platform for running application software.

- *Devices* in software deployment system include personal computing devices, servers, embedded devices, Internet of Things and so on. In one word, any device which needs system patches or software installation will be considered.

- In package management systems, which are commonly used with Linux-based operating systems, usually, a *package* is a binary bundle that contains a component, all the data needed to its correct functioning and some metadata [7]. More generally, it represents the way software are distributed. It is the unit of deployment.

- The *metadata* of a package describes its attributes and its requirements with respect to the environment in which the package will be deployed [8]. Metadata is embeded in package to provide information such as the package's name, version, a description of its functionalities as well as its dependency requirements. Figure 3 shows an excerpt of the metadata taken from the firefox2 DEB package[13].

[13] `http://www.mancoosi.org/edos/formalization/`

Package: firefox
Version: 1.5.dfsg+1.5.0.1-2 ...
Depends: fontconfig, psmisc, libatk1.0-0 (>= 1.9.0), libc6 (>= 2.3.5-1) ...
Suggests: xprint, firefox-gnome-support (= 1.5.dfsg+1.5.0.1-2), latex-xft-fonts
Conflicts: mozilla-firefox ($\ll$ 1.5.dfsg-1)
Replaces: mozilla-firefox
Provides: www-browser...

Figure 3: Example of DEB package metadata

- Software *repository* is a special file server containing the distributed packages, it is used to provide packages and package metadata. Package repositories store packages, package metadata, and the root metadata file. The root metadata provides the location and secure hashes of the tarballs that contain the package metadata. For example, the root metadata file is called different things in APT (*Release*) and YUM (*repomd.xml*) but the content is similar [6]. The users can retrieve, download and install the software they need from the repository. Nowadays, the principal repositories are those centralized servers, *e.g.* Apple's app store, Ubuntu software centre. Actually, in the distributed software deployment system, each peer can be the repository.

  It is common for repositories to consist of multiple servers that host the same data. These additional servers are called mirrors and are used to offload traffic from the main repository.

- Once a piece of software (such as a utility, driver or application) is considered finished, it is released to the public with a *version*. Traditionally, the software is then at version 1.0 - but many software architects, both those working for big corporations and those coding just for fun, choose their own ways. Within a given version number category (major, minor), these numbers are generally assigned in increasing order and correspond to new developments in the software.

- In software deployment system, there is an important part - *participator*. Participators refer to the people who own the devices, who install, update, remove and use the software, who develop, release, distribute, certify the software. They play different roles in the system which will be introduced in next subsection.

The relation among these items can be described in Figure 4. It will be described in more details in section 2.2.3.

2.2.2 Functionalities

Software deployment refers to the process that starts from the moment when software works correctly on the developer's machine; it lasts till the end of the software's life cycle, *i.e.* the moment when the software is uninstalled from the user's machine. It is a process consisting of a number of inter-related activities. The beginning is the *release* of software at the end of the development cycle. Then the software is *distributed* from the developer to the end user. After that the software is *installed*

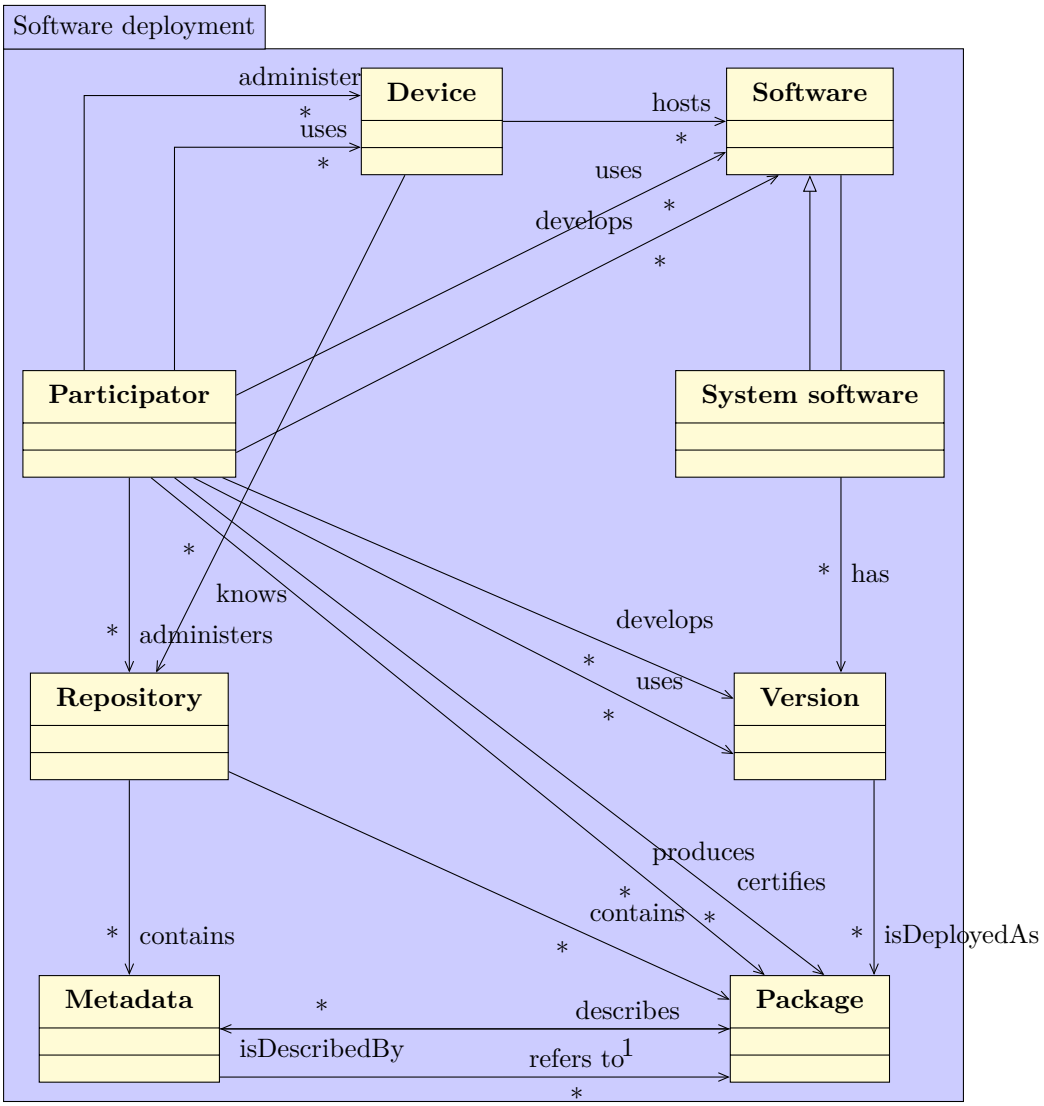

Figure 4: A UML Class diagram of software deployment system

into the execution environment; Finally the process ends with the *configuration*, *upgrade* and *removal* of the software on/from the device.

- Install or update one software: the user queries the software using its name optionally with version constraint, and fetches all the necessary metadata from the network, after resolving the dependencies (optional), it downloads all the missing packages of the software, configures and installs them on its device. If no version constraint is specified, the peer will default to choose the latest version according to its knowledge. If a software with the required name is already install on the peer, then it is an update operation.
- Remove one software: the user gives the name of the software, and the system will analyse the dependencies (optional), then remove it or parts of it from local. It is the opposite of an installation.
- Upgrade the whole system or part of it: the user checks its installation, updates

all the installed software or part of them that have a later version than the currently installed ones. In another word, this is done by executing the update operation for all the installed software or part of them.

- Synchronize with a specified peer: the user selects a specific peer to follow, and the user can choose to synchronize its installation with that peer. It is useful for configuring newly joined peers in a fully decentralized network. This functionality can be extended to follow several peers and to synchronize part of them separately.
- Certificate a new software: test the safety and functionality of a new software when the certifier receives it.
- Storage: after the distribution, each software need to be stored in the network. In decentralized network, the individual distributors must have a suitable mechanism to store all the packages without a centralized repository.
- Search: for the software deployment system, it must ensure that each user can find the software he wants quickly and accurately, especially in decentralized network.
- Notify other peers of newly available packages: after a package is issued, the producer must disseminate the metadata of the package across the network so that other peers can discover and download it.
- Download: after the user find a software he wants, he needs to download it on his device to install it.
- Purchase: not all the software are free, especially in some application stores, *e.g.* paid apps in Apple App Store and Android market, Microsoft Office, and so on. So the users need to be able to purchase these applications.
- Manage users: about the user management, the system administrator has responsibilities that include: adding, removing, or updating user account information, resetting passwords, etc.
- Manage repositories: one of the administrator's responsibilities is to add or remove the repositories that the peer knows (also include expressing the opinion on these repositories, or sending warning messages to neighbors).
- Manage trusted certifiers: one of the administrator's responsibilities, add, remove or update trusted certifiers.
- Manage trusted providers/developers: one of the administrator's responsibilities, add, remove or update trusted package providers/developers. For example, import the public key from a trusted software provider.

2.2.3 Roles

The use case diagram in Figure 5 describes the relation between the roles and the functionalities in software deployment.

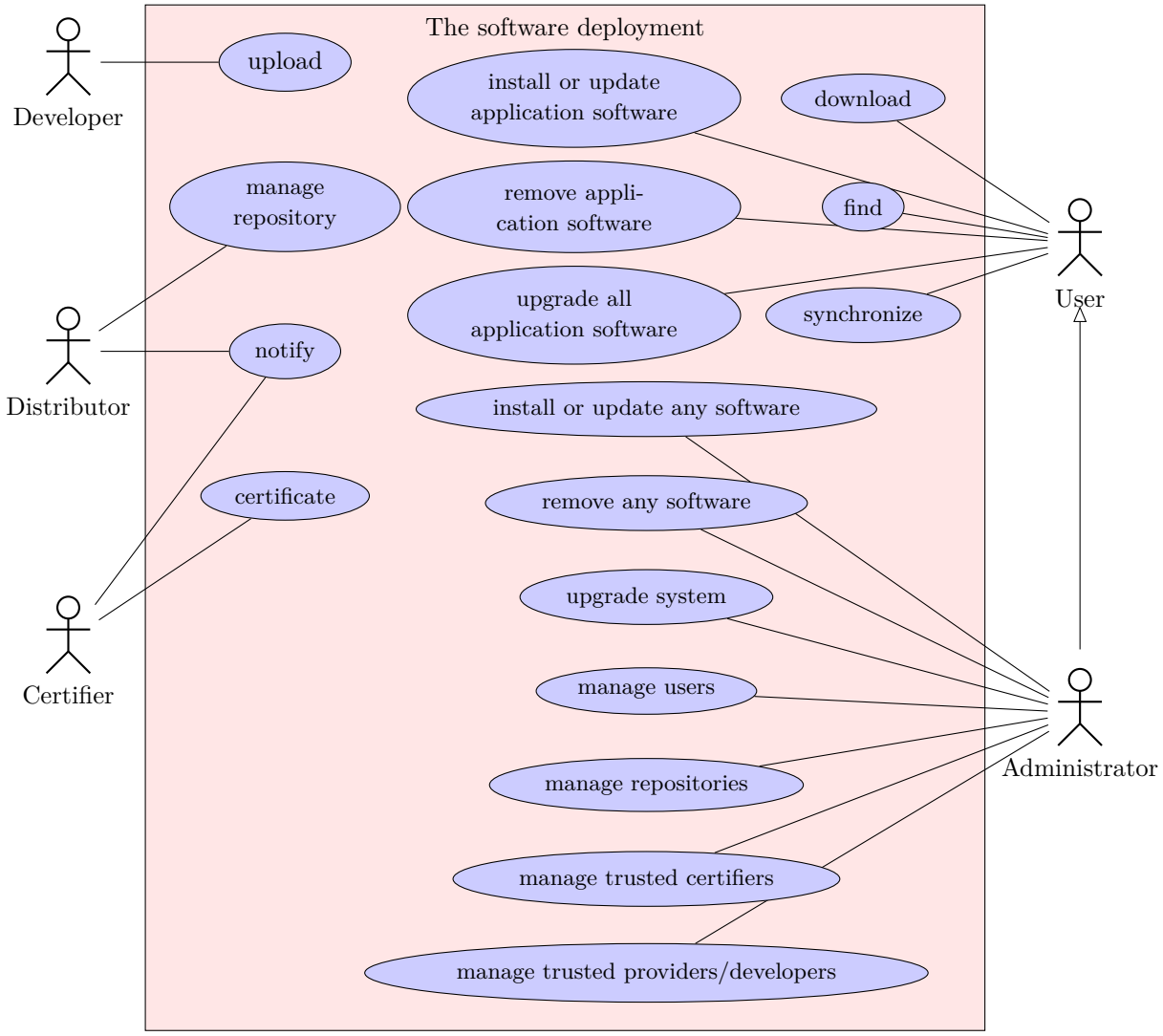

Figure 5: A UML Use case diagram of software deployment system

- Developers

  A software developer is a person concerned with facets of the software development process. Their work includes researching, designing, developing, and testing software.

  As shown in Figure 6, a developers as a participator are in charge of the development of several software and its following versions, and produces the corresponding packages. In fact, software is a set of versions. Each version consists of a set of packages. And the package also can be shared by different software.

  Nowadays, the evolution of development tools, and the growing number of reusable components, frameworks and APIs have made developing software much easier than it has ever been. Today, one can create customized applications by simply doing some drag and drop. The emergence of on-line application stores have attracted the interest of millions of developers that are more or less professional. Some are expecting to earn money, others are just

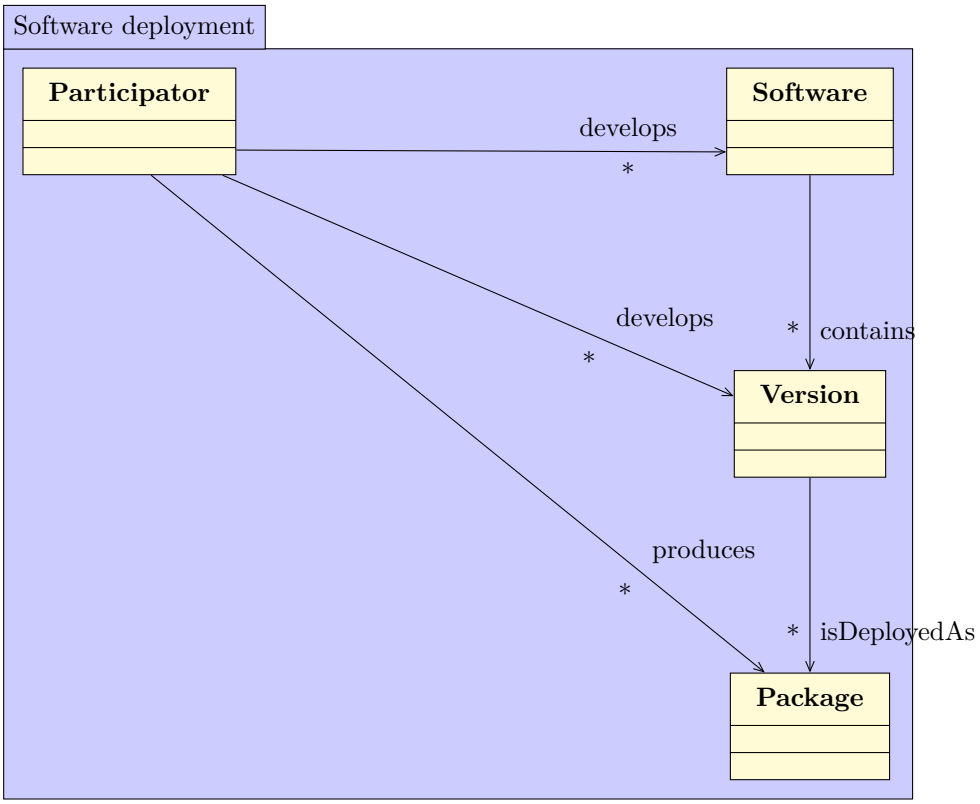

Figure 6: A UML Class diagram of developer

seeking for personal satisfaction from user's recognition. Thus, developers need to pay more attention to the popularity and the feedback of users. One issue that all developers must face is how to get their software diffused. In the past, they did it by posting advertisement on some frequently visited website, now they submit their product to application stores, the price of doing so is to pay a certain amount of distribution fees and their application must go through some opaque approval process which increases the time-to-market. Besides, application stores don't really help in diffusing newly appeared software. On the Apple's Appstore, some new features are listed on the front-page, but this information get refreshed everyday. Without enough reviews and rating, most users prefer to stick with the most popular applications which are published long time ago.

Another problem developers (and also users) care a lot is payment. Application stores do provide a uniform and simple payment service, which save users from typing coordinates and credit card information again and again. It offers also some guarantee to both sides on the payment. But equivalent service can be provided by other e-commerce platforms, say ebay. In addition, delegating too multiple responsibilities on a single actor leads to security vulnerability, like the black card issue[14]. So-called black card, the credit card that is of unknown origin, refers specifically to the action of binding with the iTunes account of illegal credit card.

[14] `http://www.ixwebhosting.mobi/2011/10/26/6463.html`

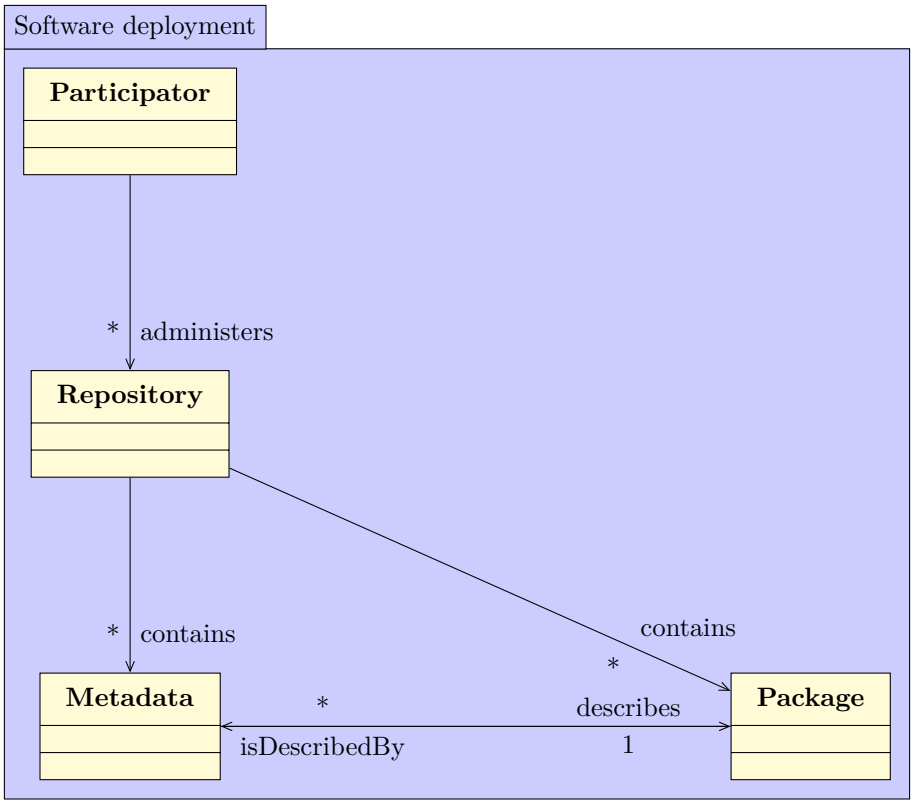

Figure 7: A UML Class diagram of distributor

- Distributors

  Figure 7 shows that distributors administer their repositories which contain packages and their corresponding metadata, and their responsibility is to collect packages and to publish them in their repositories. One package can be in several different repositories.

  Traditional distributor is a well identified entity that provides a centralized channel for diffusing software. Current large systems such as the GNU/Linux distributions rely on a single distributor [9], who collects packages from upstream sources, approves them and publishes them together in a *repository*. Such responsibility is usually taken (even not necessarily) by the OS/platform provider or device manufacture. Their motivation is to help user discover software and attract developers to their platform.

  But the repository-based approach can not fulfill the need of crowdsourced software deployment, because it exhibits some majors drawbacks, for example, managing a large repository for crowdsourced software is expensive; relying on a single distributor is a supplier-side approach while software has become a fast moving consumer good (FMCG); approving and releasing third party software through a single channel is time consuming.

  While, if both the release and delivery of packages are handled in a decentralized way, it still has some shortcoming in version management, package certification and reputation. For example, two developers modify the same version of a package concurrently, without knowing the existence of the oth-

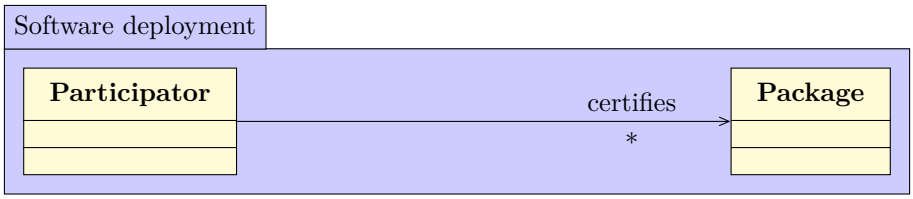

Figure 8: A UML Class diagram of certifier

ers. If the package version is still identified like in traditional repository-based system, then both developers may attribute the same version number to their new release. When the package is released in a decentralized way, there is no one like the repository to certify the package.

- Certifiers

  Certifier is an entity that guarantees the quality of software. Figure 8 shows that the responsibility of certifiers is to certify the packages and their metadata, which is an important functionality of software deployment. A package could be certified by various certifiers.

  Apple's AppStore for iPhone is currently the largest collection of mobile applications in the world. More than 500.000 apps are available. But how to ensure that the users can download these applications safely? The AppStore imposes an approval process to validate third-party applications. To get applications into the AppStore, developers are required to submit their app and wait for approval or rejection by Apple. Rejected apps are given feedback on the reason they were rejected so they could be modified and resubmitted. Here, Apple acts as a certifier. Actually, in Android market, there are also such certification processes, while, maybe the mechanisms are different. But the certifiers are very important to the software deployment system, it brings confidence.

  The basic procedure will be like this. First, the certifier gets the package from the network, and uses its approval method to estimate the quality of the package (Q). If Q is better than the threshold of the certifier, he creates the signature with the private key and encloses the signature with the package. The method of creating signature is to encrypt the package or just the metadata or the value that the package is hashed to (here, we can call them *package digest*). Then the certifier distributes this signed version of the package. To verify the contents of digitally signed package, the user generates a new package digest from the package that was received, decrypts the original package digest with the certifier's public key, and compares the decrypted digest with the newly generated digest. If the two digests match, the integrity of the package is verified[15]. It also means that if the user trusts the certifier, he can trust this package also.

- Administrators

  Administrator is a person who maintains and operates a system. More and more, administrators are usually the owner of the devices, while a device may be administered by several administrators. So they have the largest amount

[15] `http://www.altova.com/digital_signature_technology_primer.html`

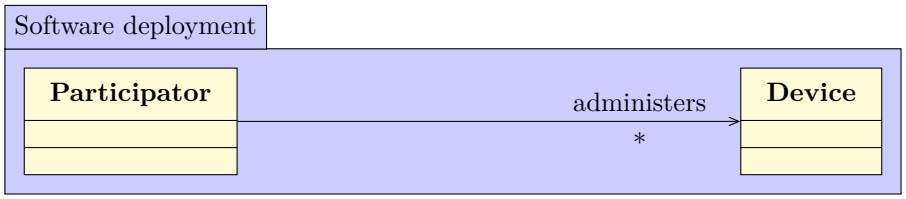

Figure 9: A UML Class diagram of administrator

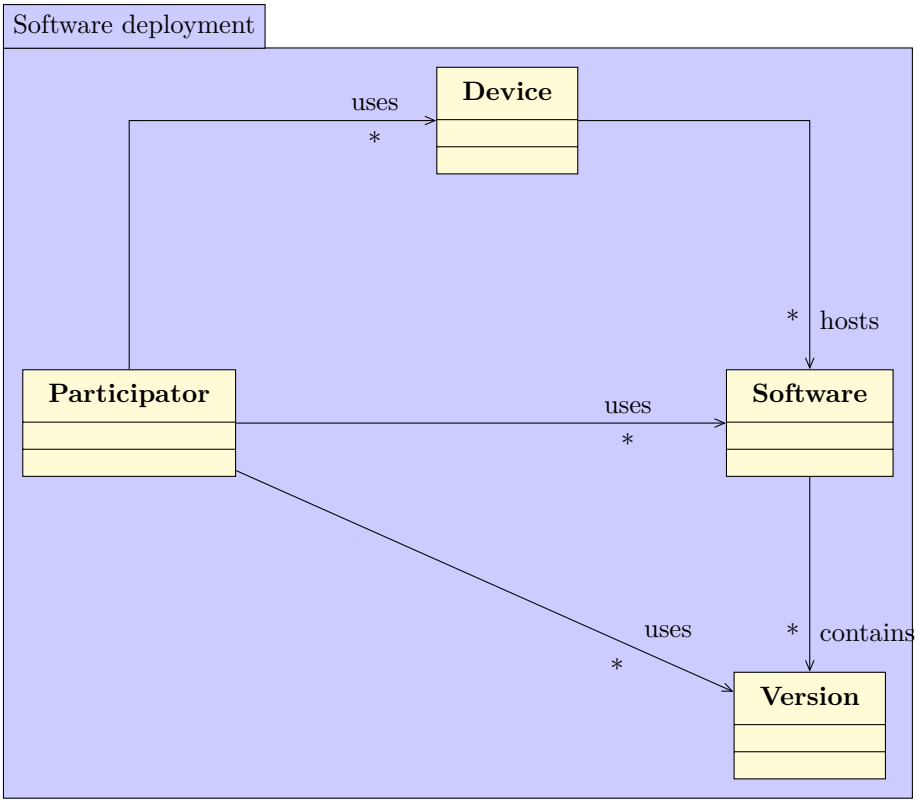

Figure 10: A UML diagram of user

of default permissions and the ability to change their own permissions. They have the ability to install, update and remove the software on their device, especially the system software. While, on the contrary, the users may just have the right to install, update and remove the application software. Figure 9 indicates that administrator is the participator who is in charge of the device. And in Figure 5, we can tell the administrator from the normal user, because their permissions are different.

- Users

  User is a person who perform most common tasks, such as running applications, shutting down and locking the device. Figure 10 and Figure 5 indicate that users can find and download software they want to the device, and install, update or remove and use this software.

  The users want the deployment process to be as simple, fast and secure as possible. The package manager should be totally transparent to users. They don't always care about where and how the software is deployed, but only want

to have a piece of software that runs correctly on their device. Most users don't have much knowledge about system administration, they don't like seeing pop-up messages or clicking on buttons to make any decision. Upon receiving an upgrade notification, a large portion of users react in two manners, upgrade all or upgrade nothing. Therefore, there is an increasing interest in automating the entire deployment process and hiding details from users. But trust is the most crucial obstacle that comes in the way. Users without enough expertise tend to rely on some well trusted authorities like Apple and Google to obtain software. However these centralized distributors don't really guarantee the quality of software appearing on their platform, all they can do is to approve software according to some general content policy.

What's more, how the individual distributors get the trust of the users is another challenge. Actually, the most effective way for users to discover good software has always been learning from friends and other people or finding them from top popular applications.

On the other hand, more and more users need "custom software". So the responsibility for application development is shifting from generic manufacturers of software to users of software, it means that the users are becoming an important part in the development of software. The requirement of users will have a great influence on the developers.

3 Theoretical Contribution

In this section, firstly we introduce our contribution to the link between popularity, reputation, certification and software deployment system, then, we present some related works. Furthermore, we also analyse the arguments of popularity and reputation, and the properties of popularity and reputation system.

3.1 Software popularity system

The software popularity system we propose contains two main parts, popularity of packages or software and popularity of peers.

The popularity of software can be defined as quality or state of being liked, installed or used by the users. High software popularity may imply that 1) many users are "interested" in the software and that 2) more users are likely to install and use the software compared to low popularity software. The software popularity is very useful in the aspect of packages storage, package search and so on.

The definition of popularity of peers is the quality or state of being liked or followed by the other peers. In software-related social network, if a peer A thinks that the installation of another peer B is good, A can follow B, then A is the follower of B. The popularity of peers can be judged by the number of followers. The more followers a peer has, the more it is popular, and therefore may be considered as a good target for following and synchronizing. The popularity of peers is very important in software-related social network.

3.1.1 Related works

In order to estimate the popularity efficiently, there are several approaches. In [10], Breslau et al. find that the distribution of page requests generally follows a Zipf-like distribution. Furthermore, Saleh et al. show that the popularity of P2P objects can be modeled by a Mandelbrot-Zipf distribution in several autonomous systems domains[3]. [11] describes methods for maintaining popularity estimate with the help of feedback from previous searches, and adapting the parameters of the random walk as a function of this variable estimate. In [12] Kamran et al. propose a globally structured hybrid peer-to-peer system, the system collected the number of replicas of the items in hybrid method.

However these approaches have their limitations. For example, the traffic of the approach of [12] is heavy when the scale of the network is large. And the distribution of packages in software deployment system is more complex because of all kinds of versions, with the different versions, the popularities of package and version are totally different. For instance, if one version of a popular package has some terrible bugs, it affects the popularity of this version, while the popularity of the package may still be high.

3.1.2 The function of software popularity system

Here, we present the importance of software popularity system in the following aspects: searching, classification, pricing, update notification adn storage.

Firstly, popularity helps users to discover good software. Nowadays, with the development of all kinds of OS, application store has become an important part

of people's personal digital life. Application store, a generic term for digital distribution platforms, contains Apple App store, Android Market[16], Windows Phone Marketplace[17], HP App Catalog[18], Ubuntu Software Center[19], and so on. There are always thousands of software in the application store, with such huge amount of software, how do the user find the better ones? For example, more than 500.000 apps are available in the Apple App Store in any category: games, lifestyle, social networking, and education, and more. The way that Apple solves the problem is *Top iPhone Apps*, which includes *Top Paid Apps*, *Top Free Apps*, *Top Grossing Apps*. In this way, users can get the most popular applications and choose the ones they want.

What's more, there are also some websites like appstoreapps.com[20], mac.informer[21], Iusethis.com[22], they display the list of the most popular applications. Because the popularity indicates which applications are installed and used by the most users, the popularity is a good way for users to discover the good software.

Another important aspect is that popularity also helps developers to decide the category and price. Because developing software is becoming much easier than it has ever been, more and more developers pay attention to the software development. Most of them purpose to earn money, so they need to make a suitable decision that which kind of application they should develop. In general, they refer the most popular application. For example, Dota is very popular, so many games similar to Dota appear; Infinity Blade sold very well at launch in Apple App Store, selling more than 270,000 copies and making over $1.4 million in its first four days after release, making it the "fastest-grossing app" ever released for iOS up to that point. So many game developers follow this successful example and release analogous games.

On the other hand, Mobile Orchard's Dan Grigsby found some relationship between popularity and price in the iPhone App Store [23]. Their test focused on a single category - the Games category which is the most popular category on the store and the most money is at play there. From the data they analysed, what's noteworthy is that the top games in terms of revenue (projected by multiplying popularity by price) are not the most popular games overall and nearly all of the top ten are by established players in the games market. Their conclusion is that established players make the most money. It helps prove that established brand values are important when it comes to making money, not a low price. Through this result we also get the conclusion that the reputation will also affect the price.

Popularity also may be used for update notification. When the most popular software are updated, the system should push the notification and even the software.

What's more, popularity plays an important role in software storage. There are two facets in this part. For the popular software, we need to prepare enough replicas in the system. In this way, when the users want to search and download the popular software, the network traffic, the load of distributors and the time users spend are all reduced. And for the rare software, they should be treated specifically so that

[16] https://market.android.com/
[17] http://www.windowsphone.com/en-US/marketplace
[18] http://www.hp.com/global/webos/ca/en/apps.html
[19] https://launchpad.net/ubuntu/+source/software-center
[20] http://www.appstoreapps.com/most-popular-apps/
[21] http://mac.informer.com/
[22] http://osx.iusethis.com/top
[23] http://mobileorchard.com/price-and-popularity-the-iphone-app-stores-data-shows-whos-making-the-big-money/

they won't disappear in the system.

3.2 Software reputation system

In our system, reputation is what is generally said or believed about a peer's or software's character or standing. Reputation can be considered as a collective measure of trustworthiness based on the referrals or ratings from members in a community. Peers express their opinions on other peers by assigning positive and negative scores, and gain or loss reputation by aggregating feedbacks from others. Considering that each peer has different roles at the same time, the reputation system will be separated into the corresponding subsystem.

3.2.1 Related works

We investigate several following reputation systems. [4] proposes PGrid, a reputation system for decentralized networks. It assumes that peers of the network are honest in most case. However it can use the *complaints* to express the distrust. The reputation system receives the complaints and uses probabilistic analysis to compute the reputation value with these complaints. If one peer's value exceed the average, it will be judged as a dishonest peer. Damiani et al. proposes the XRep protocol which is a reputation sharing protocol for Gnutella in [5]. Each peer in the network will share the reputation with others. Besides, it uses the combination of peer-based and resource-based reputation scheme and binary votes. EigenTrust proposed by Kamvar et al. in [13], is a reputation system to avoid the spread of inauthentic files in P2P file sharing networks. It uses the global reputation and uses a distributed algorithm to compute the global reputation value by combining these global trust values with local trust values. Buchegger et al. propose CONFIDANT, where each node only cares their neighbors and the firsthand information [14]. It monitors its neighbors' behavior and maintains a reputation for each neighbor. It distributes firsthand information to other nodes, and doesn't accept other firsthand information if this information differs from its own opinion. Srivatsa et al. propose TrustGuard to counter three vulnerabilities *i.e.* oscillatory peer behavior, fake transactions, and unfair rating attacks [15] which are detrimental to decentralized reputation management. It employs a *personalized similarity measure* to compute the reputation dynamically. Similar to XRep, the goal of Credence proposed by Walsh et al is to prevent file pollution in P2P file-sharing Gnutella networks. It only votes the resources by (-1,+1), and employs a *correlation coefficient* to compare voting histories of user pairs [16].

These systems also have their limitations. Firstly, their voting schemes are not suitable to software deployment system, because the voting for software should contains the users' feedback to kinds of software properties, like safety, usability, interestingness and so on. About the reputation collection, aggregation and so on, these systems usually considered one aspect, while the two aspects have both advantages and disadvantages (see section 3.2.2), so we may combine the two methods to establish a robust and reliable reputation system (see section 4.2.2).

3.2.2 Taxonomy on reputation management

We establish a taxonomy on reputation management which includes four processes *i.e.* the reputation collection, aggregation, storage, and communication. They need to be effectively designed in order to build a reliable reputation system, We employ this taxonomy to revisit and organize recent research and as a framework of the implementation of our reputation system.

- Reputation collection

  Peer-to-peer network is combined with peers and the interactions between peers, so reputation ratings are normally associated with peers. Meanwhile, there is also a reputation rating associated with a package.

  Peer-based reputations suffer from the *cold-start* problem for peers *i.e.* newcomers need to struggle initially to build their reputations [17]. While in pure package-based reputation scheme, newcomers can more quickly participate actively by distributing well-reputed packages. At the same time, this also improves load balancing by increasing copies of a well-reputed package. In turn, there is no effective way of linking a bad package to the peer that provided it in the package-based scheme. In a peer-based scheme, the malicious peers can be easily blacklisted.

  Combining peer-based and package-based reputations can relieve the shortcomings of both approaches. However, storage is becoming another disadvantage, because more space is needed for the information of the reputation of both peers and packages.

  The choice of using peer-based reputations or package-based reputations could also depend on the churn rate for each entity. If the turnover of peers in the network is high due to free and easily generated identities, then package-based reputations may turn out to be a better option. On the other hand, if the turnover of packages in the network is high, then it may be better to leverage the more stable nature of peers.

- Reputation aggregation

  Reputation systems aggregate feedback by using two approaches. One approach is to use only firsthand information *i.e.* each peer doesn't use any other peer's feedback or global information. And the second one is to use global information. Global reputation is efficient while firsthand ratings are highly reliable. However firsthand information don't help to blacklist malicious peers for others. So global information provides more information than firsthand information, reputation systems predominantly employ it.

- Reputation storage

  In distributed reputation system, we need to store the reputation data in a decentralized way. Decentralized reputation can be stored by the requester [5], the provider [18], or an anonymous third-party [4]. There are two main approaches in decentralized storage, Distributed Hash Table (DHT) and to have each peer store trust values locally.

  Using DHT means that we store the reputation information in the whole network. Each peer in the network provides some space for the reputation storage.

In this way, anonymity and redundancy mitigate peer collusion and tampering of reputation. Chord, a distributed lookup protocol that addresses this problem is a good example of DHT. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data items pair at the node to which the key maps [19]. In the second method, each peer stores the reputation information about the peers which they have interacted with locally. For example, in PRIDE, peers generate their own identities using self certification and locally store the recommendations received by them [20]. While considering the security, we should store these reputation information locally to protect the reputation data from malicious modification.

- Reputation communication

  The data exchanged, the storage mechanism employed, and the type of distributed network (structured or unstructured) are some factors that determine the type of communication protocol. For instance, PRIDE complements Gnutella with minimal modification of the Gnutella protocol [20]. And XRep, the reputation sharing protocol proposed for Gnutella, communicates via broadcasting on the Gnutella network [5].

3.3 Software certification system

Software certification is to use digital signature to guarantee that package has not been altered or corrupted since it was signed by use of a cryptographic hash. Here, a digital signature mechanism is a mathematical mechanism for demonstrating the authenticity of a digital message or document.

Many certification implementations will provide a way to sign the package using private and public key systems. The certifiers can either generate this key on their own or obtain one from a trusted certificate authority (CA). In the former case, the public key used for certification should be traceable back to a trusted root authority. And in the latter case, the user would normally have to obtain the public key in some fashion directly from the certifier to verify the object is from them for the first time.

There is another security guarantee measure, code signing. Code signing is the process of digitally signing executables and scripts to confirm the software author and assure that the code has not been modified.

The most common use of code signing is to provide security when deploying. Almost every code signing implementation will provide some sort of digital signature mechanism to verify the identity of the author or build system, and a checksum to verify that the object has not been modified.

One of its major usages is to safely provide updates and patches to existing software. Most Linux distributions, as well as both Apple Mac OS X and Microsoft Windows update services use code signing to ensure that it is not possible to maliciously distribute code via the patch system.

It's very important to note that in the software certification and code signing system, it only remains secure as long as the private key remains private and the system does not protect the end user from any malicious activity or unintentional software bugs by the software author or certifier, it merely ensures that the software has not been modified by anyone other than the author or certifiers.

Overall, package management systems provide privileged, central mechanisms for the management of software on computing systems, many packages being installed in the root context of the operating system. Thus package management security is essential to the overall security of the computing system [6].

3.3.1 Related works

Advanced Packaging Tool (APT) and *Yellowdog Updater, Modified (YUM)* are two typical package management system. They use different approaches to provide security. APT focuses on securing the repository metadata rather than signing packages. An APT repository optionally provides a signature for the *Release* file in a file called *Release.gpg*. This allows APT to verify that the *Release* file is signed by the repository key and therefore came from the repository. The Release file contains the secure hashes of the package metadata and APT can verify whether the one in package matches it. While YUM uses signatures on packages to provide security instead of signing the repository metadata. A YUM distribution maintainer signs all of the packages on the repository. YUM verifies package signatures after downloading packages [6].

3.4 The potential impact of popularity, reputation and certification on functionalities of software deployment system

First of all, we need to mention the two usages of popularity: direct usage and indirect usage. Direct usage of popularity means that the popularity result can be seen and used by the users or the other roles directly, *e.g.* function users can know the popularity of the iPhone applications via the *Top iPhone Apps*. While, some other usages of popularity are just for the judgement of the system, *e.g.* when the users search a software, the results shown to the users are sorted by the popularity. These usages are called indirect usage.

Each installation and remove will affect the popularity of a package, while the update and upgrade will influence the popularity of its versions. Meanwhile, if considering the dependency, these action will also affect the popularities of the related packages. The functionalities storage and finding are related with the indirect usage of popularity of the software, and the functionality of finding could be coupled with direct use of popularity. As we discussed in subsection 3.1.2, purchase concerns the popularity.

When the user searches a software and the sources are more than one, he can decide which one to download in terms of the reputation of software and its offerer.

In the system, certification is used in the procedure of certifying and verifying.

3.5 Popularity and reputation arguments

Which value could be used to estimate the popularity and reputation of software? We give a description of all the arguments. *Usage amount* is accurate for the popularity and reputation, but it may need the user to announce it (iusethis.com) or to install a client program to estimate automatically (software.informer.com). *Number of downloads* is easily to be calculated by using website counter. However when using this parameter, there exists a problem that one person can download many copies, inflating the total and many people can install from a single download, deflating

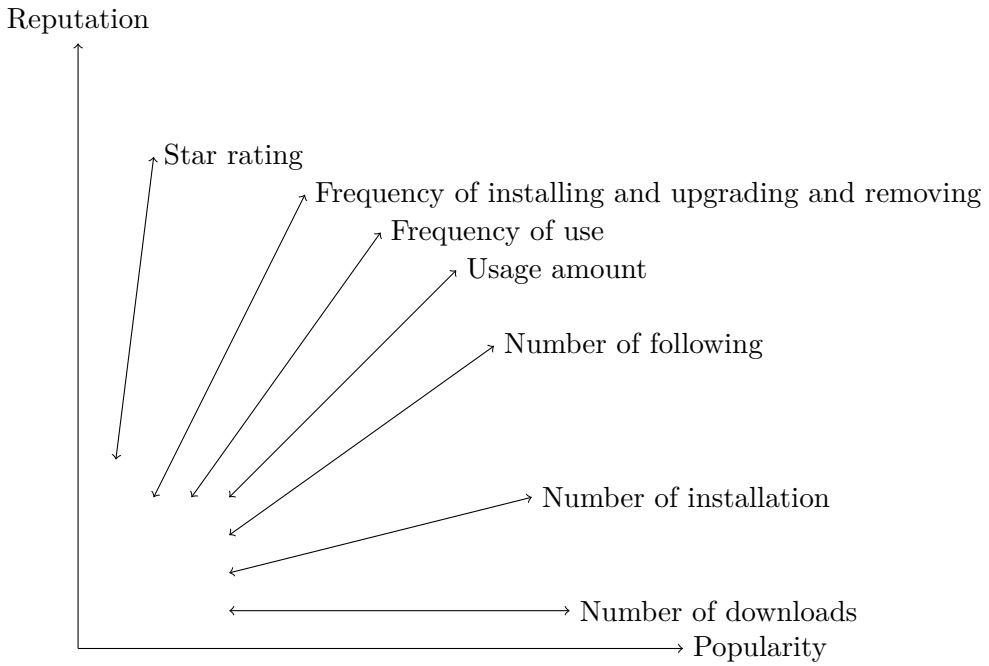

Figure 11: The influence on popularity and reputation

it. *Number of following* is also a good parameter. Maybe some user doesn't use a software, even doesn't download it, but he has an interest on it. Just like twitter, he follows the software and is a potential user of this software. Each software has its follower list and create a social network. This can solve the notifications problem. Besides, *number of installation*, *frequency of use/Number of people who use this software regularly*, *star rating*, *frequency of installing and upgrading* and *frequency of removing* are all the arguments which can affect the popularity and reputation of software. Meanwhile any composition of the previous arguments is also a good way of estimation of popularity and reputation.

Among these arguments, *usage amount, number of installation, frequency of use, frequency of installing, upgrading and removing* can be computed by installing a client program; *number of downloads* can be calculated by using website counter; *number of following and star rating* can be estimated by announcement of user.

These arguments not only are related with popularity, but also reputation. So in Figure 11, we show the influence of these arguments on popularity and reputation. For example, *star rating* is more about reputation. *Number of downloads* is almost only about popularity. While *usage amount* can affect both reputation and popularity.

Another problem need to be discussed is which popularity and reputation we should use, the ones of the package or the ones of each version.

A good argument and algorithm is very important for the popularity estimation, *e.g.* Apple changes popularity algorithm for App Store, because it found a sharp change in the most popular free apps. Now it is not only the number of downloads but also the active user's number that is going to be taken into consideration. Apple is not the first company to adjust its ranking algorithm; its main rival Google

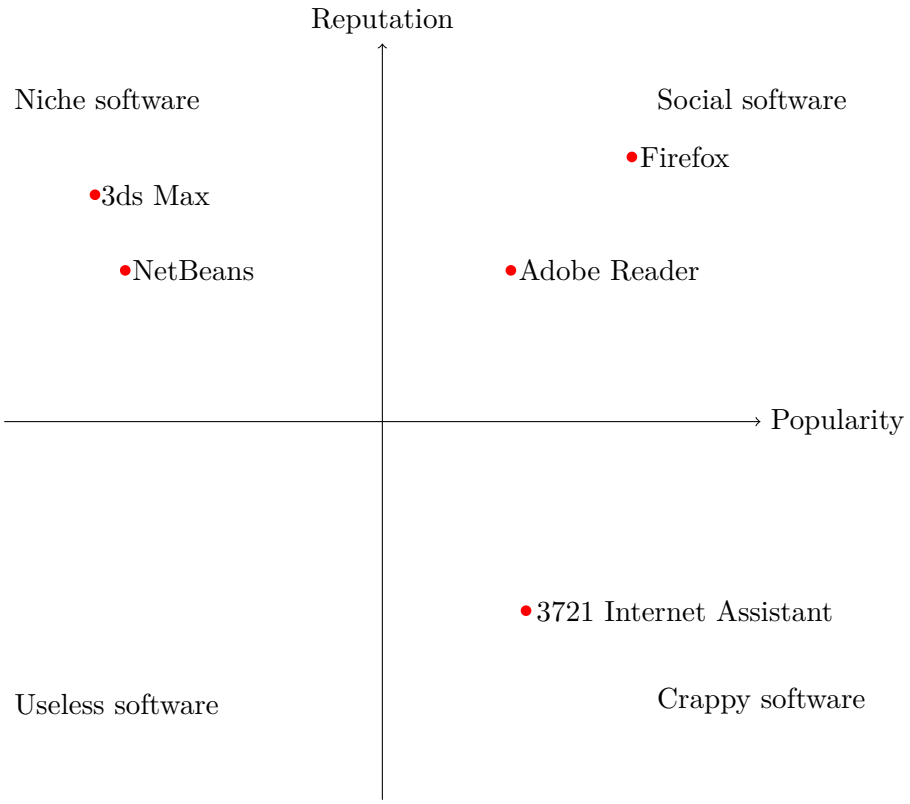

Figure 12: The relationship between popularity and reputation

initiated the practice on the Android marketplace earlier [24].

3.6 Relation between popularity and reputation

According to the weblog of Gavin Heaton[25], there are some relation between popularity and reputation. This relationship in software deployment system are in Figure 12.

Where the software's popularity and reputation are both high, they are the social software - these software have good quality and most peers want to install and use them, *e.g.* Firefox[26], Adobe Reader[27]. However, where the software is a popular one but has a lower level of reputation, the software is likely to fall into the "crappy" category. The most suitable example is malware, which is software designed to disrupt computer operation, gather sensitive information, or gain unauthorized access to computer systems. Because it can reproduce itself, it is "popular" in the network, while it harms the system, so it receives low reputation. Here we can quote an instance: 3721 Internet Assistant[28]. Lower levels of popularity but high levels of reputation indicate influence within niche users, like some professional software,

[24] http://www.itproportal.com/2011/04/19/apple-changes-popularity-algorithm-for-app-store/
[25] http://www.servantofchaos.com/2008/11/influence-and-p.html
[26] http://www.mozilla.org/en-US/firefox/fx/
[27] http://www.adobe.com/products/reader.html?promoid=DJDXD
[28] http://en.wikipedia.org/wiki/Yahoo!_Assistant

NetBeans[29] (for developing code), 3ds Max[30] (for making 3D animations, models, and images). While lower levels of both reputation and popularity indicate that the software are useless.

3.7 The properties of popularity and reputation system

According to Resnick et al. [21], popularity and reputation systems must have the following three properties to operate at all:

1. Entities must be long lived, so that with every interaction there is always an expectation of future interactions. It means that it should be impossible or difficult for an agent to change identity or pseudonym for the purpose of erasing the connection to its past behaviour.

2. Ratings about current interactions are captured and distributed. This is usually not a problem for centralised systems, but is a major challenge for distributed systems.

3. Ratings about past interactions must guide decisions about current interactions.

3.8 conclusion

In this section, we introduce the software popularity, reputation and certification system and some related works. And we can conclude that these systems are important to distributed software deployment system. In the next section, we bring the implementation of these systems base on *dPAN*.

[29] `http://netbeans.org/`

[30] `http://usa.autodesk.com/3ds-max/`

4 Implementation

We have specified the importance of software popularity, reputation and certification system in the previous sections. We also mention the limitations of current practices. So for the distributed software deployment system, we propose our novel approaches to realize the estimation of popularity and reputation, and establish a suitable software reputation and certification system.

4.1 Implementation of popularity

As previously mentioned, our software popularity system must tackle the problems of popularity information collection in a distributed environment. So in this subsection, we propose a new popularity estimation algorithm fitting to the current structure of *dPAN*. Then we show the network traffic of our algorithm and usage of our popularity system.

4.1.1 Estimation of popularity

The first thing of the estimation of popularity we need to solve is how to express the popularity of software? We can consider the following three parameters: ranking, real value and category. Ranking is usually used in the *most popular top list*. The real value (*e.g.* percentage) is used in the calculation. The category is to set threshold and separate the packages into popular ones, normal ones and rare ones, and the storage of software can use this information.

About the popularities of peer, a peer can act as certifier, user, distributor or developer, and their popularities helps user to get packages. While their popularities should not be used alone, combining with reputations are more useful.

4.1.2 The estimation algorithm

Firstly, we introduce the architecture of the system.

Figure 13 shows the architecture of the system which is composed of a core structured network (*Super-Net*) and many unstructured networks (*Normal-Net*). The *Super-Net* is a DHT-based peer-to-peer network. A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service

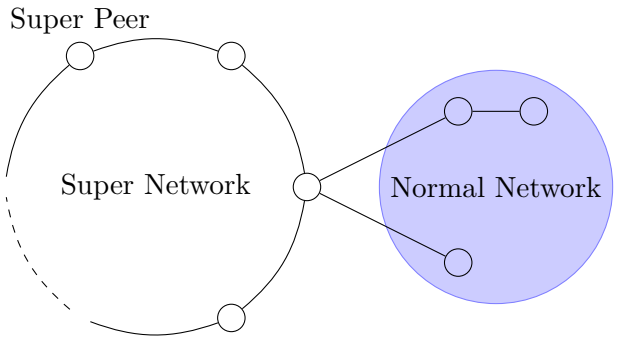

Figure 13: The Architecture of System

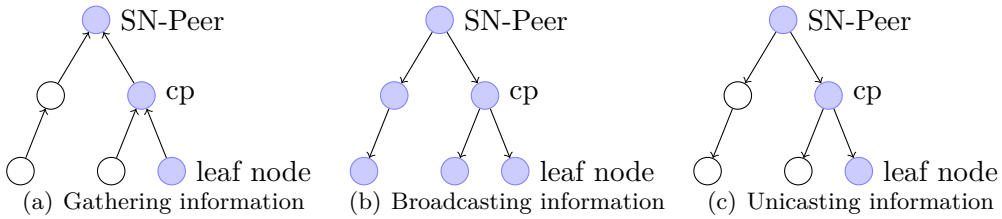

Figure 14: Information Transmission Methods

similar to a hash table [31]. Our Super-Net is a ring based DHT network similar to *Chord* which is one of the original distributed hash table protocols [22]. Each peer in the *Super-Net* namely *SN-peer* joins the ring based on its ID. This ID is a unique key and generated according to the peer's motherboard, processor, BIOS and so on.

Peers in the *Normal-Net* are called *NN-peers*. There is a threshold for load balancing, the load degree of a peer should not exceed it. When a new NN-peer joins the system, its join request is sent by the SN-peer along a random branch until it reaches a peer with the degree less than the threshold to which it will connect. This peer is called the *connect point (cp)* of the new NN-peer. Each NN-peer maintains the address of the SN-peer of its Normal-Net and its *cp* besides the neighbor list. If a *cp* leaves, it will inform its leaf nodes and transfer the load to a neighbor *cp*. And if a *cp* failed, its leaf nodes will resend join requests.

The selection of SN-peer and *cp* relates to the stability and efficiency of the system. So when we select SN-peer, the following points must be considered: performance (Server is better to be a SN-peer than a normal desktop), network (SN-peer should ensure its online ratio), and position (If a LAN can have a SN-peer, this will optimize the communication bewteen SN-peer and Normal-Net). And for *cp*, we need to make sure that its failure ratio is low.

As Figure 14 shown, there are three main information transmission methods: gathering information from leaf nodes to the Super-Net, broadcasting information from SN-peer to each leaf node and unicasting information from SN-peer to a certain node.

- Each leaf peer gathers its information and informs its *connect point (cp)*. The *cp* adds the information of its items, and sends total information to its own *cp*. Finally, all the information of the *Normal-Net* is gathered in the SN-peers as in Figure 14(a).
- When the SN-peer wants to inform the NN-peers in its Normal-Net about the estimation result, it broadcasts the information in its Normal-Net as in Figure 14(b).
- The opposite method of gathering information, the SN-peer sends the result to a certain peer as in Figure 14(c).

In each peer, there are two lists: one stores the local information (table 1) and another stores the global information (table 2). Here, in the global list, the items are ordered by their number.

[31] `http://en.wikipedia.org/wiki/Distributed_hash_table`

When a peer joins the system, it needs to send its local list to the SN-peer by using gathering method at the beginning of next period θ. Meanwhile it receives the global list from its neighbor or its *cp* or SN-peer.

During each period θ, if the peer changes its local list (install or remove some packages, add or remove a repository and etc.), it must save the changes into local changed list (table 3). In this list, -1 indicates an uninstallation, and 1 indicates an installation. Here, we calculate the popularity of package, so when a package is updated, it does not affect the list. While if we want to calculate the popularity of version, the update will affect the list.

Then at the beginning of each next period θ, it sends the local changed list to the SN-peer by using gathering method. Super-Net calculates the new popularities, and send back the result of package popularity by broadcasting. The peer updates

Local Information				
Certifiers	Distributors	Developers	Following users	Packages
certifier a	distributor a	developer a	following user a	package a
certifier b	distributor b	developer b	following user b	package b
certifier c	distributor c	developer c	following user c	package c
...	...	...	...	...

Table 1: Local List of Popularity

Global Information									
Certifiers		Distributors		Developers		Following users		Packages	
certifier a	200	distributor a	300	developer a	240	following user a	150	package a	5000
certifier b	180	distributor b	200	developer b	210	following user b	100	package b	4800
certifier c	120	distributor c	150	developer c	200	following user c	80	package c	4500
...	...	...	...	...	...	...	...	...	...

Table 2: Global List of Popularity

Certifiers		Distributors		Developers		Following users		Packages	
certifier a	1	distributor a	1	developer a	1	following user a	1	package a	1
certifier b	-1	distributor b	-1	developer b	-1	following user b	-1	package b	-1
...	...	...	...	...	...	...	...	...	...

Table 3: Local Changed List of Popularity

its global list. The algorithms are shown in Algorithm 1 and 2.

Algorithm 1: Popularity estimation (peer $NN-peer$)

```
if successors == ∅ then
    sendListToCP();
else
    foreach suc ∈successors do
        tempList ← receiveFromSuccessor(suc);
        addListToLocalList(tempList,localList);
    sendListToCP();
```

Algorithm 2: Popularity estimation (peer $SN-peer$)

```
foreach suc ∈successors do
    tempList ← receiveFromSuccessor(suc);
    addListToLocalList(tempList,localList);
receiveFromOtherSP();
calculateGlobalPop();
broadcastGlobalPop();
```

function	Description
`sendListToCP()`	Send the LocalChangedList of the NN-peer to its connect point (cp).
`receiveFromSuc(`*suc*`)`	Receive the LocalChangedList from its successors.
`addListToLocalList()`	Add the information of the successor's LocalChangedList to its own LocalChangedList.
`receiveFromOtherSP()`	Gather all popularity information in Super Net.
`calculateGlobalPop()`	After gathering, calculate the global popularity.
`broadcastGlobalPop()`	Send back the result of package global popularity by broadcasting.

Table 4: The popularity algorithm functions

In order to eliminate the information of failed or left peers, the system gathers the popularity at each beginning of long time period τ. Every peer in the system sends its local list to the SN-peer.

In the processing, in order to reduce the network traffic, we optimize the algorithm in the following aspects. About the popularity information, the list contains only top 50 or 100, instead of the whole information. When the SN-peer broadcasts the result, the result is only the package popularity. The other information, because they are not used as often as package popularity, are updated only when the peer needs them and sends a request to the SN-peer, and the SN-peer unicasts the result to the peer. In each estimation, the leaf node only sends the changed information, not the whole local list.

4.1.3 Traffic calculation

As we mentioned in the algorithm, at the beginning of each next period θ, for each NN-peer, it just need to send its list to its *cp* and get the global list from its *cp*, and for SN-peers, they need to communicate the popularity information with each other and get the newest global information after the calculation. So for a network which

contains m SN-peers and n NN-peers, the network traffic is n (NN-peer sending to its *cp*) + *2m* (the communication in Super-Net) + n (NN-peer getting local list) = *2n+2m*. So we can conclude that the network traffic won't increase too much with the increasing of peers in the system. This also means that our algorithm can work well in large scale of system.

4.1.4 The usage of our system

With the popularity of software, the system can notify the most popular software (*e.g.* Top 10 or Top 100) to the users. When users search software just with keywords, rather than an accurate software name, the system can help users decide which software to choose among the result according to the popularities of these software. Another usage is that for some space limited repositories, they can choose which software to store according to its popularities. And for the system, it should increase the replicas of the popular packages and store them in Normal-Net, and also pay attention to the rare ones, and ensure that these rare packages will not disappear in the sytem. Besides these rare software should be stored in Super-Net in order to be found easily.

4.2 Implementation of reputation and certification

As we mentioned in the previous sections, the reputation and certification system plays an important role in the safety and security of software deployment system. So the establishment of our reputation and certification system has a great significance. In this subsection, we firstly introduce our reputation rating scheme, and our reputation and certification approaches. Then we present the process of our system. Thereafter, we analyse the attacks in the reputation system and our solution. Finally, we show the usage of our reputation and certification system.

4.2.1 Reputation rating scheme

At the end of each package transmission, the requester furnishes the system with its evaluation of the package provider in the form of a reputation rating. For example, a binary rating scheme (0 indicating bad, 1 indicating good) [23], or a (-1, 1) scheme where -1 indicates a bad transaction, and 1 indicates a good transaction [13].

- Package reputation

  Considering that 1 bit contains not enough information, meanwhile there are almost no differences between transfering one bit and transfering several bits in the network, we propose a novel package reputation rating scheme.

  The feedback contains n 2-tuples (pairs) *i.e.* 2n bits. Each pair indicates one property of the software reputation. For example, the first pair indicates the overall impression of the software; the second pair indicates whether the software is safe and no harm to the device; the third pair indicates whether the behavior of software matches that of its specification...

  There are 2 bits in the pair, the first bit means whether this property is judged by the requester in this feedback, (0 indicating no feedback about this property,

1 indicating having feedback), and the second bit means the quality of the property (0 indicating bad, 1 indicating good).

We can give an example of the reputation sequence: {(overall) (safety) (functionality) (usability) (interestingness)}. So when a user votes "1100111110", it means that he thinks that the functions of the software are like it describes; the software is easy to use; the software isn't interesting; but in summary the software is good. Here, the order is important. The order in the example is default and it can be changed corresponding with the requirement of the user when a user searches software. For instance, when the user wants to find a game, the interestingness and functionality may be considered more.

The novel package reputation rating scheme provides not only a detailed description of kinds of properties of the software, but also a method of searching. When a user wants to search a safe and interesting software, the order will be changed to (overall) (safety) (interestingness) (functionality) (usability). The user sends the search request and then he receives several results. Then he can use the new order to change the reputation of the received software, and the software with the biggest reputation is the most satisfying one.

- Peer reputation

  The format of peer reputation is similar to the package one, but each pair indicates the reputation of the peer's different role. For example, when a user gives a vote "00110000" to a peer based on the sequence {(certifier) (distributor) (developer) (following user)}, it means that the user regards the peer as a good distributor.

4.2.2 Decision on each aspect of reputation management

About reputation estimation, we need consider the following aspects:

- Reputation collection

  In the subsection 3.2.2, we have mentioned that both Peer-based reputations and Package-based reputations exhibit some shortcomings and advantages, so combining peer-based and package-based reputations looks promising.

- Reputation aggregation

  We choose both global reputation and firsthand information, this takes advantage of the reliability of firsthand reputation and the efficiency of global reputation.

- Reputation storage The storage scheme we choose is based on the aggregation, that is to use a Distributed Hash Table (DHT) to store the global reputation and to store the firsthand information locally.

- Reputation communication

  Because the calculation of global reputation should be based on the hybrid distributed structure which is used in popularity estimation, so we choose the similar communication methods to popularity estimation. Besides in SuperNet which uses a DHT-based storage approach, when the peer receives a search

message with a data key that it is not responsible for, it forwards the request according to its routing table. And in Normal-Net, broadcast and unicast are used more usually.

4.2.3 Certification approaches

Considering the security rules and attacks on secure package manager, we propose our own secure package manager. The principles in our system contain the following two parts: one, we sign both metadata and packages, because signing both metadata and packages makes it more difficult for an attacker to launch most types of attacks [24]; the other, we combine the code signing and the digital signature of distributors and certifiers, *i.e.* we utilize the reputation of developers, distributors and certifiers to strengthen the certification.

Now we describe the process of certification in *dPAN*. When a package is deployed, the developer first hashes the metadata and the package to *metadata digest* and *package digest*. Then uses his private key to sign these two digests. After that, the package is uploaded to the distributor, in the same way, the distributor also need to sign the digests of metadata and package with his private key. When a certifier get the package from the network, he uses his approval method to estimate the quality of the package (Q). If Q is better than the threshold of the certifier, he uses his private key to sign the digests of metadata and package.

When the user receives this package, he needs to use the corresponding public key to decrypt the metadata and package, and gets *expected metadata digest* and *expected package digest*. Meanwhil, he also needs to use the same function to hashes the metadata and the package to *actual metadata digest* and *actual package digest* If the expected digests and the actual digests match, the integrity of the package is verified.

4.2.4 The reputation and certification system

Firstly, we present the procedure of reputation aggregation. Each peer stores the local reputation separately (table 5). Similar to the estimation of popularity, the aggregation of the global information also bases on the hybrid structure of the system.

Reputation Information			
Packages Reputation		Peers Reputation	
package a	vote	peer a	vote
package b	vote	peer b	vote
package c	vote	peer c	vote
...	...	...	...

Table 5: Reputation List

When a peer joins the system, it needs to send its local reputation list (if it has) to the SN-peer by using gathering method at the beginning of next period θ. During each period θ, if the peer has a package transmission, it adds its new vote about this transmission into local list. Then at the beginning of each period θ, each peer

sends the local reputation list to the SN-peer by using gathering method. Peer *cp* calculates the average value of all the information from its successors, and sends the average information to its own *cp*. In this way, the accuracy of the estimation result is not great enough, but we can reduce the influence of the collusion. Then Super-Net calculates the new reputation and store the global reputation information. The algorithms are shown in Algorithm 3 and 4.

Algorithm 3: Reputation estimation (peer $NN-peer$)

```
if successors == ∅ then
    sendRepToCP();
else
    n ← 0;
    foreach suc ∈successors do
        tempList [n] ← receiveFromSuccessor(suc);
        n++;
    calculateAverageValue(tempList,localRepList);
    sendRepToCP();
```

Algorithm 4: Reputation estimation (peer $SN-peer$)

```
n ← 0;
foreach suc ∈successors do
    tempList ← receiveFromSuccessor(suc);
    n++;
calculateAverageValue(tempList,localRepList);
receiveFromOtherSP();
calculateGlobalRep();
```

function	Description
`sendRepToCP()`	Send the reputation list of the NN-peer to its connect point (cp).
`receiveFromSuc(`*suc*`)`	Receive the reputation list from its successors.
`calculateAverageValue` (*tempList*,*localRepList*)	Calculate the average value of all the reputation information from its successors.
`receiveFromOtherSP()`	Gather all reputation information in Super Net.
`calculateGlobalPop()`	After gathering, calculate the global reputation.

Table 6: The reputation algorithm functions

Then, we introduce our system which consists of the following phases: package searching, package reputation requirement and package selection, peer reputation requirement and calculation, best peer selection, package downloading, certification and voting.

1. Package searching. First, the initiator i broadcasts to all its neighbors a *Request* message containing the search keywords. When a distributor receives a *Request* message for which it has a match, it responds with a *Response* message. The *Response* message includes the packages's metadata (names, version, package_id and related information), the speed of the responder, the peer_id of the responder.

2. Package reputation requirement and package selection Upon recep-

tion of the *Response* message, the initiator i selects, among the possibly different packages offered, the package p that best seems to satisfy its request. Such selection can be guided by the design of our package reputation rating scheme and the user's preferences. Because i never has these packages, it needs to get the global reputation of these packages from the *Super-Net*. Then i uses the reputation to select a most suitable package.

3. Peer reputation requirement and calculation. As a result of the previous phase, i gets a targeted package and a set of its offerers. So in this phase, i needs to judge these distributors by using peer reputation. Firstly, i sends the global distributor reputation request to the *Super-Net*, and gets the set of these distributors' global reputation (dgr). If i has downloaded package from these distributors, it should have these distributors' local reputation (dlr). Then the peer reputation can be calculated like:

$$peer\_reputation = dlr \times \alpha + dgr \times (1 - \alpha) \tag{3}$$

Here the range of α is from 0 to 1, and it represents the degree that i trusts its local information. The value of α is decided according to the user and system. While, if i never has communicated with the distributor, the peer_reputation just equals dgr.

4. Best peer selection. Previous phase presents the set of these offerers' reputation, and i should select the best offerer from which to execute the download. A straightforward way is for i to download the package from the offerer who has the best reputation (according to the peer_reputation received and calculated). The drawback of such a solution is that it could easily create a performance bottleneck on reliable distributor. To avoid this, our system selects a set of reliable distributors (*e.g.* the top 5 in peer_reputation list), whereafter, we set a threshold λ, if the size of package is smaller than λ, we choose one distributor from the set as the best offerer; else, we choose all the distributors in the set as best offerers for parallel downloads, and in this case, we can make full use of decentralized network, increase the speed of package downloading and reduce the occupation of some most reliable distributors.

5. Package downloading. Now i has decided from which distributor to download the package. Then, it contacts the chosen distributor directly and requests the package. Meanwhile if i choose parallel downloads, it can ask different distributors for different fragments of the package to be downloaded.

6. Certification and voting. After the download, i already has the metadata and package, so it uses our certification approaches to check the package to ensure its integrity. If there is no correspondence, it will discard the package. Also, i updates its opinion on the downloaded package and its distributor (maybe also its developer and certifier). The opinion is stored as firsthand information and it also affects the global reputation.

4.2.5 Attacks to reputation-based systems and our solution

Managing reputation is not a easy task in distributed system. We list several attacks in reputation-based system, and present our solution to these attacks.

- *Collusion.* Collusion occurs when two or more peers collectively boost one another's reputation or conspire against one or more peers in the network. For

example, a colluding group inflates each other's reputations which then allows them to use the good reputation to attack other system peers.

- *Unfair ratings.* Unfair ratings mean that a malicious peer tries to subvert a system by falsely rating a bad transaction as good, and vice versa. This can affect the reputation of distributors unfairly [25].

- *Sybil attack.* In a sybil attack, a malicious user obtains multiple fake identities and pretends to be multiple, distinct nodes in the system [26] [27].

  In our system, the peer uses both global reputation and firsthand information, while these three attacks can not affect the firsthand information, and they almost have no effect on the global reputation because of our global reputation collection algorithm (we have mentioned in section 4.2.4), unless the number of fake identities is too huge in sybil attack. So in general, these three attacks have no effect on our system.

- *Dynamic peer personalities.* Some peers can exhibit a dynamic personality, that is, switching between an honest and dishonest behavior. Reputation milkers, or oscillating peers, are one type of peer personality that builds a good reputation and then takes advantage of it to do harm [28]. While in our system, the reputation of peers is not the only criterion, it needs also consider the reputation of packages. So when a peer wants to do harm, the receiver can reject this transaction because of the bad reputation of the package.

4.2.6 Problems solved

Here, we describe some important problems to be aware of when designing a practical reputation system, as well as our corresponding solutions.

- *Cold-start.* Cold-start is a big problem in reputation-based systems. While thanks to peer-based reputation, new packages offered by a well established peer will be regarded as reliable, this has the advantage of avoiding cold-start problems for new package. On the other hand, peer-based reputations suffer from a cold-start problem in the sense that new peers offering a limited number of packages will struggle to actively participate in the distribution. Our package-based reputations make new peers can immediately participate in distribution of well known packages.

- *Change of identities.* A peer may rejoin the system under a new identity if he can benefit from doing so. While in section 4.1.2, we have mentioned that the ID of each peer is unique and related to its motherboard, processor and BIOS. So in our system, the peer can not change its identity.

- *Performance bottlenecks.* A peer-based reputation approach may foster bottleneck creation directing all downloads to the most reputable peers. Our algorithm allows peers to choose one distributor from a set of best offerers or choose several best offerers for parallel downloads.

- *Blacklisting.* Package-based reputations do not support blacklisting as no linking is made between a bad package and the peer that provided it. On the

```
Lancement du peerHT sur la machine pc-df-691 : java Init -ht -n 19 -p 1 1 -s 1.txt
Lancement de la simulation sur la machine pc-df-692 : java Init -pht pc-df-691 -p 2 2 -s 2.txt
Lancement de la simulation sur la machine pc-df-693 : java Init -pht pc-df-691 -p 3 3 -s 3.txt
Lancement de la simulation sur la machine pc-df-694 : java Init -pht pc-df-691 -p 4 4 -s 4.txt
Lancement de la simulation sur la machine pc-df-695 : java Init -pht pc-df-691 -p 5 5 -s 5.txt
Lancement de la simulation sur la machine pc-df-696 : java Init -pht pc-df-691 -p 6 6 -s 6.txt
Lancement de la simulation sur la machine pc-df-697 : java Init -pht pc-df-691 -p 7 7 -s 7.txt
Lancement de la simulation sur la machine pc-df-698 : java Init -pht pc-df-691 -p 8 8 -s 8.txt
Lancement de la simulation sur la machine pc-df-699 : java Init -pht pc-df-691 -p 9 9 -s 9.txt
Lancement de la simulation sur la machine pc-info-186 : java Init -pht pc-df-691 -p 10 1 -s 1.txt
Lancement de la simulation sur la machine pc-info-187 : java Init -pht pc-df-691 -p 11 2 -s 2.txt
Lancement de la simulation sur la machine pc-info-188 : java Init -pht pc-df-691 -p 12 3 -s 3.txt
Lancement de la simulation sur la machine pc-info-189 : java Init -pht pc-df-691 -p 13 4 -s 4.txt
Lancement de la simulation sur la machine pc-info-190 : java Init -pht pc-df-691 -p 14 5 -s 5.txt
Lancement de la simulation sur la machine pc-info-191 : java Init -pht pc-df-691 -p 15 6 -s 6.txt
Lancement de la simulation sur la machine pc-info-192 : java Init -pht pc-df-691 -p 16 7 -s 7.txt
Lancement de la simulation sur la machine pc-info-193 : java Init -pht pc-df-691 -p 17 8 -s 8.txt
Lancement de la simulation sur la machine pc-info-194 : java Init -pht pc-df-691 -p 18 9 -s 9.txt
Lancement de la simulation sur la machine pc-info-195 : java Init -pht pc-df-691 -p 19 1 -s 1.txt
```

Figure 15: Simulation starting

```
## -> Local Popularity : {
Certifiers ()
Distributors (dis1 1 )
Developers ()
Following users ()
Packages (apt 1 diff 1 cron 1 ftp 1 )}

## -> Global Popularity : {
Certifiers (cer3 2 cer2 2 cer1 2 )
Distributors (dis1 8 dis3 2 dis2 2 )
Developers (dev3 4 dev1 2 dev2 2 )
Following users (fu2 4 fu1 2 fu3 2 )
Packages (libc6 10 apt 8 diff 6 cron 6 libattr1 4 libacl1 4 ftp 4 apt-utils 4 base-passwd 4 hal 2 gimp 2 eject 2 binutils 2 coreutils 2 )}

## -> Installation : {ftp (5cf80bb1bbe377b3029902072922c27e), apt (f030f433144035c7c0b0ef4084fd812e), cron (6045adc14bc4e7f45275bdb67952669e)
, diff (7de44fda7cf2a80cd3bdbe771b5c5b3f)}.
## -> Stored Packages : {ftp (5cf80bb1bbe377b3029902072922c27e), apt (f030f433144035c7c0b0ef4084fd812e), apt (9223b72087b45f7fad4513a2691b98c
5), binutils (3e38e5cdeba154ec8605bae8abc1a258), cron (6045adc14bc4e7f45275bdb67952669e), diff (7de44fda7cf2a80cd3bdbe771b5c5b3f)}.
```

Figure 16: Peer local information

other hand, peer-based reputations can effectively support blacklisting only giving up on anonymity. In our context, since users can not change its ID, blacklisting of malicious peers is effective under the combination of peer-based and package-based reputation.

4.3 Simulation and conclusion

We simulate our popularity estimation algorithm and reputation and certification system based on *dPAN*. The simulation is realized in Java, and we use Java RMI (Java remote method invocation) technology to realize the distributed environment. Java RMI is a Java API that performs the object-oriented equivalent of remote procedure calls[32]. The total code line number of *dPAN* is about 6000, and the part of popularity and reputation is about 1500 lines. Then we use shell script to launch these peers on different machines in Telecom Bretagne, so the capacity of the system is based on the number of machines in our school.

4.3.1 Simulation result

We start the simulation as shown in Figure 15. There are 19 peers in this simulation. These peers install the required packages, judge these packages and other peers during the simulation.

After the simulation, the information of each peer is stored in the *log* file. Figure 16 shows the information of one peer.

[32] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

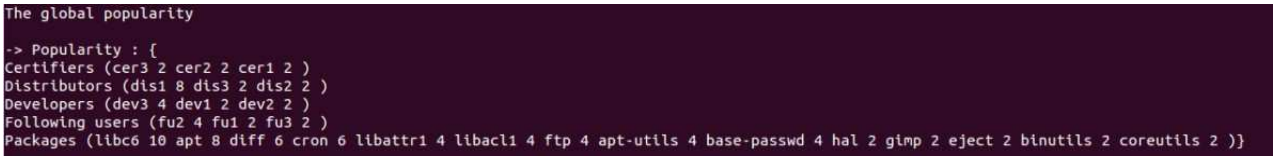

Figure 17: Popularity result

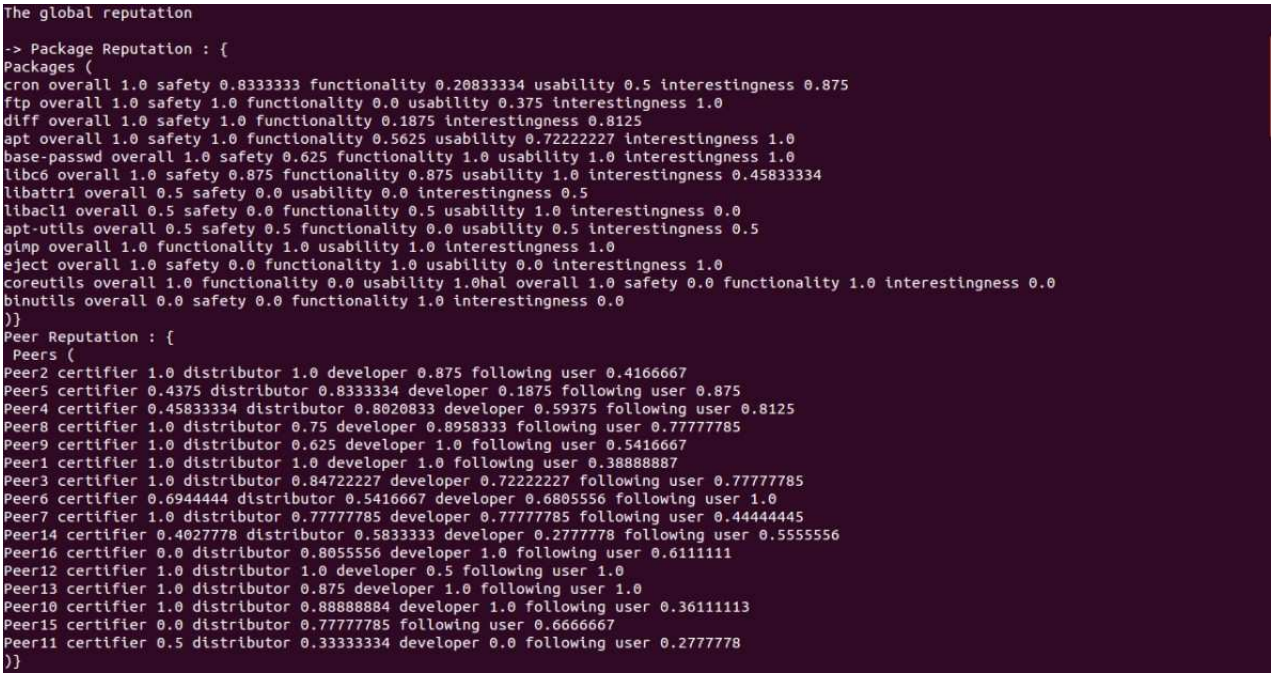

Figure 18: Reputation result

Figure 17 is the global popularity. And in the parentheses, each 2-tuple means the name of the item and the number, and these 2-tuples are sorted from high to low according to the number.

The reputation of peer and package is shown in Figure 18. In the part of package reputation, one line expresses the reputation of one package. The 2-tuples following the package name represent the reputation result of the packages' properties. The result of property only is shown when it has been judged. For example, no one judge the usability of package *diff*, so in the line of *diff*, there is no information of usability. While there is a 2-tuple (usability 0.0) in line of *libattr1* which means the users dislike the usability of package *libattr1*. The peer reputation is similar to the package one. Those 2-tuples represent the reputation result of each peer as different roles.

The simulation result shows that our algorithm and system can work well in a hybird distributed software deployment system. However because the capacity of our system is not huge enough (the maximum number of peers is about 100), so we can't evaluate the performance of our algorithm in the large scale environment.

5 Conclusion and remaining challenges

With the popularity of user-generated content and the development of crowdsourced software, the current approach adopted by most software package management systems which rely on a single point of distribution can not satisfy the demand of modern users and developers. A new distributed approach for software deployment is becoming mandatory, that is in a network consisting of interconnected symmetric peers, all the developers are allowed to release and distribute software asynchronously and all the users can download software from their neighbors and certify them. In this distributed software deployment system, the estimation of popularity, reputation and certification, which is simple in centralized architecture, becomes more difficult and more important. The primary goal of this report has been to investigate and address the calculation of popularity and the establishment of the system of reputation and certification, and to use our system to help user to search and download a safe and useful software.

In this report, concepts and principles related with popularity, reputation, certification and software deployment system have been introduced. We analyse the components of software deployment system, for instance, items, functionalities and roles. Moreover, we present the link between distributed software deployment system and popularity, reputation and certification. We also represent several related works of popularity, reputation and certification. They all exhibit certain limitations with respect to the distributed environment and the special characteristics of the packages and peers in distributed software deployment system. So we propose our algorithm and architecture for estimating popularity, and proposed a decentralized reputation and certification system by combining the advantages of several previous systems and our own ideas.

The architecture we designed for *dPAN* is the foundation of our system. The package searching, storage, information collection and communication all base on this hybrid distributed network structure. The popularity estimation algorithm took both reduction of network traffic and appropriate degree of accuracy into consideration. It means that we can calculate the popularity of packages and peers with light network traffic in distributed software deployment system. The reputation system can aggregate the reputation of packages and peers in a reliable, accurate and safe way. The certification system can guarantee that the package received by the users has not been altered or corrupted. What's more, the reputation and certification system can help user to find a most suitable and safest software, and help the spread of software of high quality. Our system also tried to address the leading reasons for erroneous and misleading values produced by reputation systems today, like user collusion. Furthermore, we designed a special reputation rating scheme for the package and peer, and this made that the reputation votes contain more information.

Though we have proposed a suitable solution, finding a high efficient and full-decentralized popularity estimation algorithm and designing a robust and reliable distributed reputation and certification system are still largely open challenges.

Our work isn't a final and perfect solution, on the contrary, there are still many places that can be improved. Our work just represents a step towards a future software deployment system. Meanwhile, many open issues are left for future research:

- The full distributed popularity estimation

Our algorithm of popularity estimation is based on a hybrid structure. While the network of *dPAN* consists of interconnected symmetric peers, so we need to find a more efficient and full-decentralized popularity estimation algorithm.

- Different levels in certification system

  In certification system, there are a lot of different certifiers, and their standards to certify are also different. Someone just give basic verification like correctness of metadata, respect of law and so on; others guarantee correctness (package will not break the device); some Unix/linux repositories need to make sure the dependences. The users need to know the different certification in order to choose a suitable software. So to establish a certification level system which will anounce the different certification content is important. The certifiers can use the different levels in the certification level system to announce which kind of certification they make.

- Different certifiers and coexistence problem of the original version with the signed versions

  In *dPAN*, each normal peer can be a certifier. But also Apple, Debian, Microsoft and so on, they are special and more authoritative certifiers. Usually, they pay more attention to those applications which they developed by themselves or are designed for their own devices or operation system. While the ordinary certifier may certify every package which he receives or he has interest in or he has installed and used. These individual certifiers accelerate the certification process and reduce the interval between the distribution of the software and the usage by the users. But it also brings a problem. After the certifiers distribute their signed versions of the package, there will be several different versions in the network at the same time, the original version, the distinct version signed by different certifiers. How can we solve the coexistence problem of all these kinds of package versions? We propose several thoughts: first, all kinds of package versions are allowed to exist in the network, and the users can choose the version according to the certifier they trust or just the original one. However, this measure will increase the difficulty of storage and search of the package; or we can keep only one version on the basis of the feedback of users, or according to the reputation of the certifiers; or we can keep all kinds of metadata, but just one package.

- The method of dealing with mischievous certification and venomous package

  When the users or other certifiers find the mischievous certification, how can they notify the other users and handle the venomous package? Our suggestion is that the users or certifiers firstly try to confer with the certifier who made this mischievous certification, when this certification is really mischievous, if the certifier is conscious of his mistake, he needs to cancel this certification version, else the users or certifiers need to broadcast this mischievous certification and the related certifier.

- The establishment of software-related social network

  The peer in *dPAN* is similar to the person in social network (fackbook[33], twit-

[33] `https://www.facebook.com/`

ter[34] and so on), where person can follow others and acquaint himself with the others' information and preferences. So we also want to build up a software-related social network, and in the network, each peer can follow other peers, and once a peer install or update a new software, it can recommend this software to its followers. This social network should be based on the popularity and reputation of packages and peers.

Through the internship in the Computer Science Department of Telecom Bretagne and under the supervise of Prof. Fabien Dagnat and Prof. Gwendal Simon, I have learned how to work in a research team, how to investigate a topic, and also got trained of divergent thinking and convergent thinking. Besides, this internship gave me a good experience in software design and development, and a strong understanding of software deployment and software architecture in a distributed context. I think all these assets will be extremely valuable for my following research studies.

[34] `https://twitter.com`

References

[1] X. Zhang and F. Dagnat and G. Simon. Toward Decentralized Package Management. In *Lococo - Workshop on Logics for Component Configuration*, 2011.

[2] C. Dale and J. Liu. apt-p2p: A Peer-to-Peer Distribution System for Software Package Releases and Updates. In *Proc. of IEEE INFOCOM*, pages 864–872, 2009.

[3] O. Saleh and M. Hefeeda. Modeling and Caching of Peer-to-Peer Traffic. In *ICNP*, pages 249–258. IEEE Computer Society, 2006.

[4] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the tenth international Conference on Information and Knowledge Management (CIKM01)*, pages 310–317, 2001.

[5] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 207–216. ACM, 2002.

[6] J. Cappos, J. Samuel, S. Baker, and J. Hartman. Package management security. *University of Arizona Technical Report*, pages 08–02, 2008.

[7] F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen. Managing the Complexity of Large Free and Open Source Package-Based Software Distributions. In *ASE*, pages 199–208. IEEE Computer Society, 2006.

[8] J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, and F. Mancinelli. Improving the Quality of GNU/Linux Distributions. In *COMPSAC*, pages 1240–1246. IEEE Computer Society, 2008.

[9] M. Wu and Y. Lin. Open Source Software Development: An Overview. *IEEE Computer*, 34(6):33–38, 2001.

[10] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 126–134 vol.1, 1999.

[11] N. Bisnik and A. Abouzeid. Modeling and analysis of random walk search algorithms in p2p networks. In *Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on*, pages 95–103. IEEE, 2005.

[12] M.M. Kamran and S. Khorsandi. Popularity estimation in a popularity-based hybrid peer-to-peer network. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 399–404, feb. 2011.

[13] S. D. Kamvar, M. T. Schlosser, and H. Garcia-molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the Twelfth International World Wide Web Conference*, pages 640–651. ACM Press, 2003.

[14] S. Buchegger and J. Le Boudec. A Robust Reputation System for P2P and Mobile Ad-hoc Networks. 2004.

[15] M. Srivatsa, L. Xiong, and L. Liu. TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks. In *Proceedings of the 14th international conference on World Wide Web*, pages 422–431. ACM, 2005.

[16] K. Walsh and E.G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation-Volume 3*, pages 1–1. USENIX Association, 2006.

[17] Y. Zhang and Y. Fang. A Fine-Grained Reputation System for Reliable Service Selection in Peer-to-Peer Networks. *IEEE Trans. Parallel Distrib. Syst.*, 18(8):1134–1145, 2007.

[18] B. Ooi, C. Liau, and K. Tan. Managing Trust in Peer-to-Peer Systems Using Reputation-Based Techniques. In Guozhu Dong, Changjie Tang, and Wei Wang 0010, editors, *WAIM*, volume 2762 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2003.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[20] P. Dewan and P. Dasgupta. Pride: peer-to-peer reputation infrastructure for decentralized environments. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 480–481, New York, NY, USA, 2004. ACM.

[21] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, December 2000.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[23] L. Xiong and L. Liu. PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. *Knowledge and Data Engineering, IEEE Transactions on*, 16(7):843–857, 2004.

[24] J. Cappos, J. Samuel, S. M. Baker, and J. H. Hartman. A look in the mirror: attacks on package managers. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 565–574. ACM, 2008.

[25] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM conference on Electronic commerce*, EC '00, pages 150–157, New York, NY, USA, 2000. ACM.

[26] J. Douceur and J. S. Donath. The Sybil Attack. pages 251–260, 2002.

[27] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 267–278, New York, NY, USA, 2006. ACM.

[28] G. Swamynathan, B. Zhao, K. C. Almeroth, and H. Zheng. Globally decoupled reputations for large distributed networks. *Adv. MultiMedia*, 2007(1):12–12, January 2007.