Finding Android Malware Trace From Highly Imbalanced Network Traffic

Ying Pang†, Zhenxiang Chen†*, Xiaomei Li†, Shanshan Wang†, Chuan Zhao†, Lin Wang†, Ke Ji†, and Zicong Li†
†School of Information Science and Engineering, University of Jinan
Jinan, Shandong, China, 250022
‡Department of Political Theory, Shandong Sport University
Jinan, Shandong, China, 250102
*Corresponding author, Email: czx@ujn.edu.cn

***Abstract*—With the yearly increase of the amount of Android users, malicious applications for mobile terminals are emerging in endlessly. Many researchers have started to explore how malicious apps are detected from the perspective of network traffic. We design and implement a control and management system of Android traffic collection, which contains the functions of downloading APKs, malware static detection, network traffic collection and resources management. It can collect network traffic efficiently and manage the dataset easily. Furthermore, we address the machine learning based malware detection which using network traffic is an imbalanced learning problem. In addition, four imbalanced algorithms are applied to Android malware detection using the highly imbalanced network traffic dataset. The result of the experiments show that the combination of SMOTE and SVM are the best performer in the all combinations.**

I. Introduction

In recent years, smart mobile devices are in full swing. According to the world's largest IT industry analysis and consulting services company Gartner's data, the number of worldwide device shipments by Android operating system is 1.455 billion in 2015, and it will reach 1.619 billion in 2016.[1] With the yearly increase of the amount of Android users, malicious applications for mobile terminals are emerging in endlessly, which brings great threat to users of their privacy and property safety.

The current malware detection methods can be roughly classified into three categories: static analysis, dynamic system-level behavior analysis, and network-level behavior detection [3]. The static analysis is using reverse-engineering technology to analyze source codes without running any application. The dynamic system-level observes various events and dynamic run-time characteristics of mobile devices, such as CPU usage, number of running processes, and power, to determine whether malicious behavior exists. There are many forms of the network-level behavior detection, for example, Perdisci et al [4] focused on analyzing the structural similarities among malicious HTTP traffic traces generated by executing HTTP-based malware. There are also many researchers that use the network traffic to train machine learning model for detecting malicious behavior. Arora et.al [5] has built a rule-based classifier for detection of Android malwares using the network traffic features. Most of the existing studies have been detecting the Android malwares using the balanced training dataset. However, the number of malicious flow is much less than the number of benign flow in the wild. And this situation will significantly compromise the performance of most the standard learning algorithms.

We design and implement a control and management system of Android network traffic collection. It can improve the efficiency of the network traffic collection and make the operation of dataset management easier. Furthermore, we analyze the network traffic dataset which we have collected through our system and use it to train four kinds of machine learning models. The results of the experiments show that we will face an imbalanced learning problem if we will want to use the network traffic to detecting Android malwares. Whats more, four kinds of imbalanced algorithms are applied to Android malware detection. The experimental results show the combination of the synthetic minority oversampling technique (SMOTE) and support vector machine(SVM) are the best performer in the all combinations.

The remainder of the paper is organized as follows. The related work is discussed in Section II. In Section III, we present the control and management system of Android traffic collection. We analyze the Android traffic dataset we have collected in the section IV. Section V discusses the malware behavior detection using highly imbalanced network traffic. Section VI presents the limitations and future work of this paper. Brief conclusions are outlined in the last section.

II. Related Work

A number of studies have used machine learning methods to detect malicious apps. They extract features form Network traffic, which are applied to train machine learning models and identify malicious apps. For example, TrafficAV [6] performs a multi-level network traffic analysis, gathering as many features of network traffic as necessary. It proposed a method that combines network traffic analysis with machine learning algorithm (C4.5 decision tree) which is capable of identifying Android malware with high accuracy. Wang et al[7] has proposed a

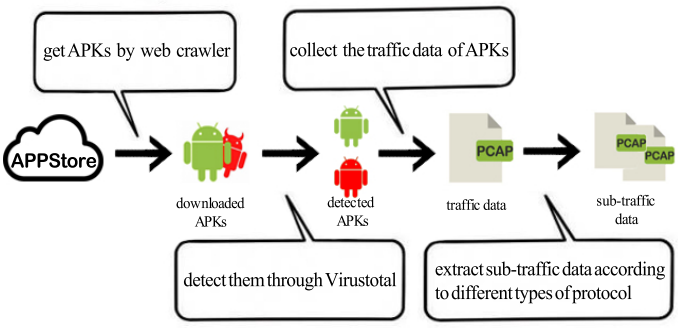

Fig. 1. The relationship between APK and traffic.

method that based on neural network and deep learning to on applications of feature learning, protocol identification and anomalous protocol detection.

However, the result of our network traffic collection shows that using network traffic to detect malware behavior is an imbalanced learning problem. The methods mentioned above used the balanced network traffic dataset to train detection models and didn't take the imbalanced learning problem into consideration. Peng et al[8] developed an IDGC-based model to resolve imbalanced Internet traffic identification problems. However, it is only concerned about the issue of network traffic identification, it didn't mention the Android malware behavior detection.

In this paper, we design and implement a control and management system of Android network traffic collection. We find that using the network traffic to train machine learning models is an imbalanced learning problem by analyzing the network traffic we have collected. In addition, four imbalanced algorithms are applied to Android malware detection using the highly imbalanced network traffic dataset.

III. The Android Traffic Collection and Management System

Different from the current traffic data collection system with a single function and relied on a lot of manual operation, our system is methodical and is able to collect a large-scale network traffic data. Our system includes not only the pre-processing functions of traffic collection, such as downloading APKs, static detecting APKs and APKs' data records management, but also the post-processing functions of traffic data collection, such as stream file extraction and resources sharing. More importantly, we can control and manage the number of threads for traffic collection and the reboot time of Android virtual device, which can obtain purer traffic data efficiently. The main goal of this system is to let the complex work of collecting mobile applications network traffic data easier. We can get a complete relationship between APK and traffic data through using this system as shown in Figure 1. We can clearly know an APK whether is malicious, the functional classification of it and the information of its network traffic.

TABLE I
The APP stores we can download applications at present

APPStore Name	URL
anzhi	apk.hiapk.com
gfan	apk.gfan.com
mi	app.mi.com
sogou	app.sogou.com
huawei	appstore.huawei.com
163	m.163.com
10086	mm.10086.com
91	play.91.com
baidu	shouji.baidu.com
appgionee	www.appgionee.com
360	zhushou.360.cn

A. Architecture

The system consists of four modules, these nodules are pre-processing module, traffic collection module, post-processing module and system management module. The architecture of the system is shown in Figure 2.

B. Core Modules

1) Pre-processing: The main work of this module is processing APK files which used to collect traffic data. It contains APK downloading function and APK static detecting function.

APK downloading function supports two methods of downloading. One is downloading APKs according to APPs name,another is downloading APKs according to APP Store. The first downloading method is present as follows. First, the system conducts a fuzzy search in the main App Stores based on the APK name which user input. Then, the system displays APKs' information which is qualified to the user's input in the form of a list. Finally, users select APKs which they want to download based on this information. The second method is downloading APK according to APP Store in bulk. First, users need to select an APP Store and some related parameters such as the categories of APKs, the number of downloads for each category, the earliest and the latest on-line time of the applications. Then, the system downloads APKs based on these parameters in bulk. The APP stores where we can download applications in the present are listed in Table I.

The other function is static detecting APKs through uploading APK files to the third-party Website(VirusTotal)[2]. Then, according to the results of the static test, the system adds the APK to a malicious application set or a benign application set.

2) Traffic collection: This module is to manage and control the work of network traffic collection. Users can select the applications to get network traffic data. In order to improve the efficiency of collection, the system can set the number of threads for traffic collection work depending on the number of applications and the required time of program execution. The system can also set the reboot time of Android virtual machine, in order to collect a more pure network traffic. The system can observe the Android virtual devices and the status of each thread in real time and obtain the final traffic data set after all the traffic collection threads concluded.

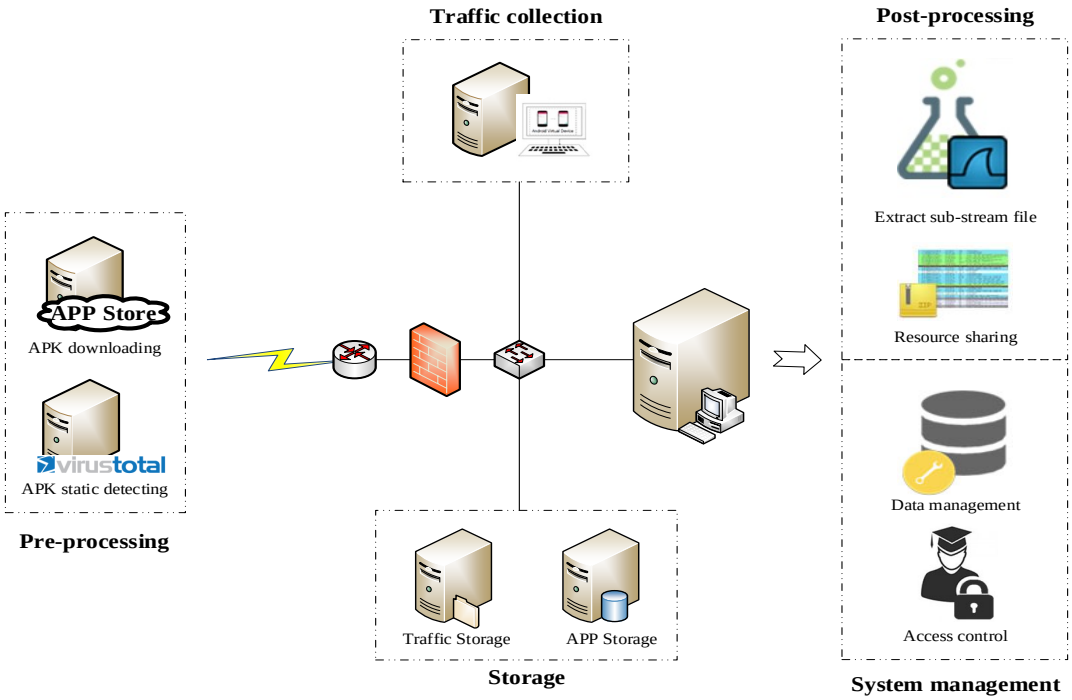

Fig. 2. The architecture of Android traffic collection control and management system.

3) Post-processing: This module is to analyze traffic data we have collected. It also can be divided into two functions, extract stream file function and resources sharing function.

Extract stream file function can extract sub-data stream from the collected traffic data and save it as a stream file with the ".PCAP" extension according to the different types of protocol(HTTP,DNS,TCP,HTTP&TCP) for subsequent analysis.

The system can compress and package sub traffic data files based on user's selected through resources sharing function.

4) System management: This module is divided into data management function and access control function.

Data management function can manage all the records and their relationship in our system, such as the APP Store data records, the APKs' data records, the network traffic data records and so on.

As the system includes not only the traffic data collection function which oriented to researchers, but also the resources sharing function oriented to the general users, the system needs closely access control. The system has three user roles in the present. Those are super administrators, researchers and visitors. Each role is assigned different permissions according to their different requirements.

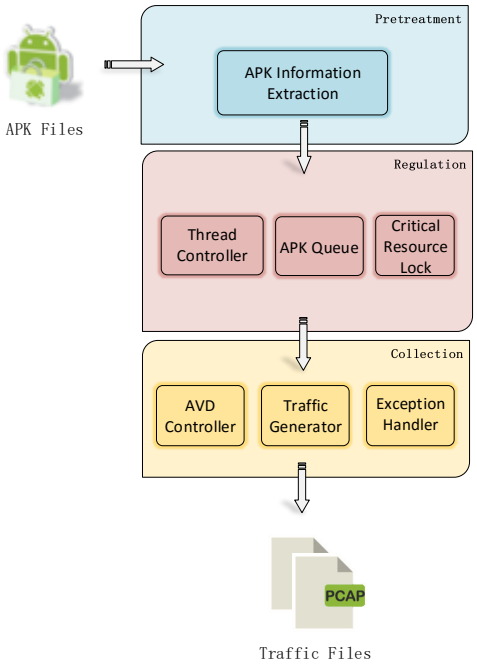

Fig. 3. The work flow of traffic collection

C. Implementation Details

1) Multi-thread: Traffic data collection is the core module of the system. In order to improve collection efficiency, we use multi-threading technology. The system can set the number of threads for traffic collection depending on the number of applications and the required time of collection.The work of each thread contains three parts, pretreatment, regulation, and collection.

The work of each thread contains three parts, pretreatment, regulation, and collection, as shown in Figure 3.

The pretreatment part is to preprocess those APK files to

install these apps automatically. The traffic collection module can parse the AndroidManifest.xml through modifying the command of *aadp dump badging*. From the AndroidManifest.xml files, we can obtain some essential information, such as package name, MainActivity, and storage path which are required by an installer.

The regulation part consists of three components, the critical resource lock, thread controller, and APK queue. The critical resource lock controls critical resources to ensure the correctness of the corresponding programs. Critical resources include the rights to use adb command and to control the APK queue. In addition, the right to read and write thread information resources is also included. The thread controller is responsible for controlling thread starting, thread input, thread output and terminating one thread. When a thread starts, the thread controller needs to receive an APK message from the APK queue and relay this message as a set of parameters to the specific thread. The APK queue is used to store APK information whose network traffic data we wish to obtain.

The collection part includes the AVD(Android Virtual Device) controller, traffic generator, and exception handler. When a thread begins to execute, the collection machine uses the command of the adb and emulator to control the AVDs creation and launch. After starting AVD, the APK is installed on the emulator by the adb shell command. If the APK is installed successfully, then the AVD is shut down. The next step is to restart the emulator and use the tcpdump tool to collect traffic.

2) Python interface: Some functional modules like APK downloading, third-party detection, data stream file extraction mentioned in Section 3 are realized by Python at first. Then, the Python scripts are embedded in the J2EE platform.

For general JAVA program, you can use jython to achieve seamless combination with Python scripts.[9] When JAVA projects is running, all classes and Meta information will be stored in the JVM PermGen space (permanent generation space). As the system already contains a lot of Classes, then if running Python scripts in the JAVA web project through importing jython.jar files, it can easily lead to the exception of java.lang.OutOfMemoryError: PermGen space. To avoid this problem, we use the Runtime Class of JAVA.

First, we encapsulate the input parameters as a string with JSON format and concatenate it behind a string command. Then, the Python scripts parses the string into a JSON object to obtain the information of the parameters. Correspondingly, the Python script also transforms the output result into a JSON format string and print it out. Then, the JAVA program obtains the string by reading the output stream. Finally, the JAVA program parses the string to a JSON object and continue the next operation.

3) Concurrency control: The system uses *synchronized* (a JAVA language keyword) to prevent the current execution of APK web crawlers, because the web crawlers's execution time is very long, and it will take up a lot of bandwidth during the execution. The specific algorithm for APK web crawlers is shown in Algorithm 1.

Algorithm 1: The algorithm of APK web crawlers' concurrency control

```
Input: the parameters of APKs to crawl
Output: APKs' information crawled
if the downloadingFlag is false then
    Set the downloadingFlag is true;
    Downloading APKs;
    if the downloading task succeeds then
        Alert the success information to user;
        Set the downloadingFlag is false;
    else
        Alert the error information to user;
        Set the downloadingFlag is false;
    end
else
    Alert the error information to user;
end
```

IV. Android Traffic Analysis

So far, we have downloaded 5191 APPs, detected 4992 APPs, and collected 1479.68MB traffic data through our system. The details of downloaded and collected APKs as shown in Table II. Further, we analyzed the basic traffic breakdown by the application layer protocols which is shown in Table III, and found that the major traffic type is HTTP(91.84%). As we can see, there are some differences between the number of downloaded APKs, the number of detected APKs, and the number of APKs to collecting traffic. I would like to explain the reasons for these differences. There are two reasons for the difference between the number of APKs for static detection and the number of downloaded APKs. One is that we skipped the APKs failed to upload for detecting because we had unstable connections with the foreign websites(Virustotal). The other is that VirusTotal limited the size of APK to detect less than 30MB, so we also skipped the file larger than this size. The reason caused the differences in the number of APKs for collecting network traffic is that there are some failures about using the Android virtual device to automatically install and running APKs. These failures lead to the result that relevant APKs doesn't successfully collect traffic. To solve these problems is our future work.

Based on our analysis, we find that the number is highly imbalanced between the malicious traffic and benign traffic as we can see in the Figure 4. There are about 1303.371MB legitimate traffic we collected. However the number of malicious traffic we collected is just about 1.689MB. Thus, it will be an imbalanced learning problem if we want to use network traffic to build classifier for malware detection. We will talk about it in the next section.

V. Malware Behavior Detection Using Highly Imbalanced Network Traffic

As we all know, the imbalanced learning problem significantly compromised the performance of most standard

TABLE II
THE DETAILS OF DOWNLOADED AND DETECTED APKS;THE "N" REPRESENTS THE NUMBER OF DETECTION MECHANISMS THAT DETERMINE IT IS A MALWARE, IT PROVIDED BY VIRUSTOTAL(THE TOTAL NUMBER IS ABOUT 55-56.)

Category	crawled APKs	static detect						
		detected APKs	0	$1 <= n < 10$	$11 <= 0 < 20$	$21 <= n < 30$	$31 <= n < 40$	$n >= 40$
AntiVirus	379	376	96	57	127	94	2	0
Browser	57	57	39	9	9	0	0	0
chess	296	285	94	60	48	77	6	0
Communication	141	141	55	35	25	24	2	0
DailyLife	372	369	175	123	49	22	0	0
Education	498	445	173	97	102	73	0	0
Finance	489	486	223	188	52	23	0	0
HealthAndFitness	427	419	163	129	61	66	0	0
Input	25	23	15	6	2	0	0	0
MediaAndVideo	397	390	151	79	94	66	0	0
NewsAndMagazines	485	485	218	143	85	40	0	0
Personalization	81	81	23	9	36	13	0	0
Photography	499	467	131	108	133	93	0	0
Productivity	363	363	237	88	28	10	0	0
Reading	473	453	80	114	120	132	0	0
Sport	107	79	50	15	13	1	0	0
Tools	88	66	37	11	10	8	0	0
TravelAndLocal	14	7	1	2	1	0	0	0
Total	5191	4992	1961	1273	995	742	10	0

TABLE III
THE DETAILS OF TRAFFIC COLLECTION AND DNS/HTTP TRAFFIC PROPORTION IN THE APKS

Category	traffic collection			
	collected APKs	traffic data (MB)	DNS	HTTP
AntiVirus	371	44.3	2.06%	86.86%
Browser	49	4.54	1.81%	87.67%
chess	110	15	1.24%	77.33%
Communication	140	15.4	1.37%	83.77%
DailyLife	140	15.4	1.22%	89.67%
Education	253	35.6	1.31%	89.32%
Finance	351	115	0.54%	88.70%
HealthAnd Fitness	313	126	0.49%	89.44%
Input	21	8.81	0.72%	86.95%
MediaAnd Video	333	108	7.17%	86.94%
NewsAnd Magazines	374	188	0.49%	93.09%
Personali- zation	145	47	0.67%	92.55%
Photography	352	128	0.58%	94.53%
Productivity	253	186	0.22%	95.70%
Reading	498	175	0.94%	91.43%
Sport	19	1.01	6.57%	80.83%
Tools	83	17.2	1.66%	89.54%
TravelAnd Local	14	53.4	0.05%	96.26%
Total	3897	1305.06	17.21%	91.84%

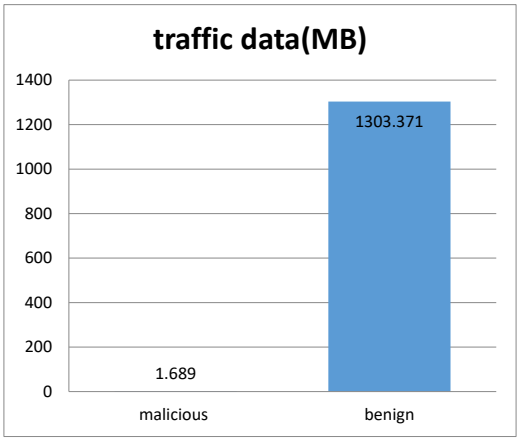

Fig. 4. The statistic results of malicious network traffic and benign network traffic.

learning algorithms [10]. In this section, we apply four classic machine learning algorithms and four imbalanced algorithms to identify malicious network behaviors using the imbalanced network traffic dataset we collected and analyze the results of those experiments.

A. Feature Selection

The current study only focuses on the statistical features of the amount of downlink traffic, the amount of uplink traffic and the duration of a traffic flow.

TABLE IV
CLASSIC CLASSIFIERS WITH THEIR PARAMETERS

Algorithm	Parameter
KNN	k = 5, weights = 'distance'
DecisionTree	criterion = 'entropy'
SVM	kernel = 'rbf', probability = True
AdaBoost	DecisionTreeClassifier(max_depth = 1), algorithm = 'SAMME', n_estimators = 200

TABLE V
THE RESULTS OF THE CLASSIC CLASSIFIERS;'P' REPRESENTS THE PRECISION RATE,'R' REPRESENTS THE RECALL RATE, 'F1' REPRESENTS THE F1-SCORE,'A/T' REPRESENTS THE AVG/TOTAL,'ACC' REPRESENTS THE ACCURACY RATE,'C0' REPRESENTS THE BENIGN CLASS,'C1' REPRESENTS THE MALICIOUS CLASS.

			KNN	Decision Tree	SVM	Ada-Boost
Train	**P**	**c0**	1	1	1	0.99
		c1	1	1	1	0
		A/T	1	1	1	0.99
	R	**c0**	1	1	1	1
		c1	1	1	0.85	0
		A/T	1	1	1	0.99
	F1	**c0**	1	1	1	1
		c1	1	1	0.92	0
		A/T	1	1	1	0.99
	Acc		1	1	0.99	0.99
Test	**P**	**c0**	1	1	0.99	0.99
		c1	0.4	0.12	0	0
		A/T	0.99	0.99	0.99	0.99
	R	**c0**	1	0.99	1	1
		c1	0.25	0.25	0	0
		A/T	0.99	0.99	0.99	0.99
	F1	**c0**	1	1	1	1
		c1	0.31	0.17	0	0
		A/T	0.99	0.99	0.99	0.99
	Acc		0.99	0.99	0.99	0.99

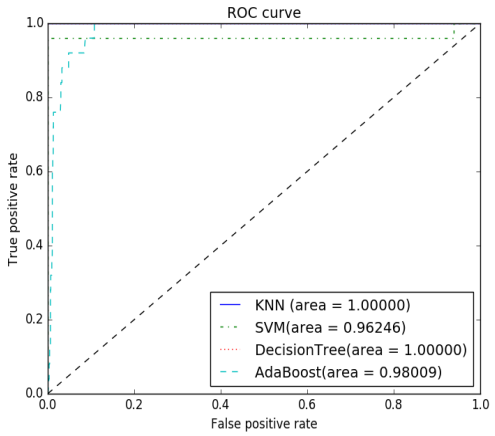

Fig. 5. The ROC curves of the classic classifiers on training set.

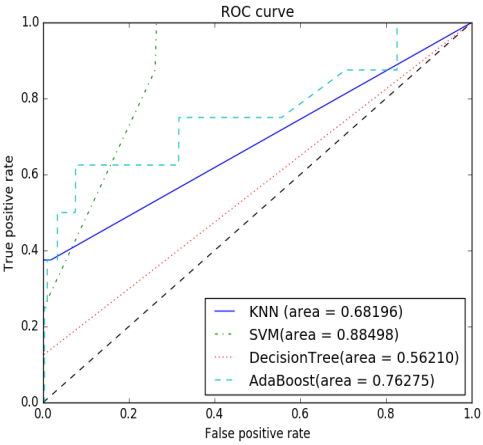

Fig. 6. The ROC curves of the classic classifiers on testing set.

B. Evaluation on Classic Classifiers

1) Experiments: We apply four standard machine learning classification algorithms to detect malicious network behaviors using the imbalanced dataset. The four classic classifiers are KNN [11], decision tree [12], SVM [13], and Adaboost [14]. Those are all embedded into scikit-learn which is an open source machine learning library for the Python programming language [16]. The parameters of the four classic classifiers are listed in Table IV. To make sure the reliability of detection model, 75% of feature vectors are applied to train detection model while 25% of the feature vectors are applied to test the detection model. The evaluation outcomes of accuracy, precision, recall and f1-score are presented in Table V and the ROC curves [17],[18] of training set and testing set are presented in Figure 5 and Figure 6.

2) Result Analysis: As we can see in the Table V, the accuracy rate of those classifiers can reach over 99% both on the training set and testing set. The ROC curves show that the performances of most classifiers are desirable on training set, while are undesirable on the testing set. On the testing set, all of the machine learning classification models have excellent accuracy rate, but have the poor scores of precision, recall and f1 of the malicious class (minority class). The reason for this phenomenon is that the classifiers recognize malicious traffic as benign traffic which is much more than malicious traffic. In our research domain, the consequence of recognize malicious traffic as benign traffic can be overwhelming costly, more so than classifying a benign traffic as malicious. The result of experiments illustrates that the common basic machine learning classification algorithms is not effectively on the imbalanced network traffic dataset. Therefore, it is evident that we require a classifier that will provide high accuracy for the minority class without severely jeopardizing the accuracy of the majority class.

C. Evaluation on Imbalanced Methods

1) Imbalanced learning Methods: There are two main methods for imbalanced learning, the sampling methods and the cost-sensitive methods. In this paper, we selected four algorithms in sampling methods. There are the EasyEnsemble [19] , BalanceCascade [20], SMOTE [21] and ADAYSN [22].

TABLE VI
IMBALANCED ALGORITHMS WITH THEIR PARAMETERS

Algorithm	Parameter
EasyEnsemble	n_subsets = 100
BalanceCascade	default
ADAYSN	default
SMOTE	kind = 'regular'

EasyEnsemble and BalanceCascade are common algorithms in ensemble. SMOTE and ADAYSN are used in oversampling.

EasyEnsemble is a very straightforward algorithm that can iteratively select a random subset and make an ensemble of the different sets. It is an unsupervised learning algorithm to explore the majority class data by using independent and random sampling with replacement.

Balancecascade algorithm uses a specific classifier to develop an ensemble of classifiers to undersample. Specifically, BalanceCascade trains the learners sequentially, where in each step, the majority class examples that are correctly classified by the current trained learners are removed from further consideration.

The synthetic minority oversampling technique (SMOTE) is a powerful method that can create artificial data based on the feature space. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors[21].

The adaptive synthetic (ADASYN) sampling approach uses a systematic method to adaptively create different amounts of synthetic data according to their distributions. It is to use a weighted distribution for different minority class examples according to their level of difficulty in learning, where more synthetic data are generated for minority class examples that are harder to learn compared to those minority examples that are easier to learn[22].

2) Experiments: The above algorithms are implemented through the imbalanced-learn module of scikit-learn [23]. The parameters of the four imbalanced algorithms are listed in Table VI. We split the network traffic dataset in 75% training set and 25% testing set like the preceding experiments. After training and testing, the ROC curve results are shown in Figure 7 and Figure 8 on the training and testing sets.

3) Result Analysis: Figure 8 shows that the combination of imbalanced methods and classifiers is better than the individual classic classifiers. For the ensemble method, the best performer is the combination of BalanceCascade and SVM. As for the oversampling method, the combination of SMOTE and SVM is better than the others. It is also the best performer in the all combinations. The AUC evaluation proves that the oversampling methods and ensemble method are both effective on the imbalanced network traffic dataset. This is particularly obvious on the testing set.

However, the result of the experiments is not ideal. The best AUC [18] evaluation result is only 0.92006 on the testing set. It is because that the count of minority samples is very small (we only have 33 malicious network traffic samples). Therefore, we will collect more malicious network traffic in our future work.

VI. LIMITATIONS AND FUTURE WORK

Although the system has achieved many functions,such as downloading APKs, collecting network traffic, detecting malwares and so on, there are still some imperfections in the system. We use the activation method based on emulator restarting. However, other types of activation methods, such as receiving a call, accessing a website, are not implemented. Furthermore, some advanced malware can detect the current operating environments before it runs its malicious code. To solve the above problems, we will add activation methods and enable the collection of traffic data on real smartphones in our future work.

What's more, the effects of the experiments on imbalanced learning are not agreeable, one of the reasons is that we only use a small amount of statistical features to train the machine learning model. There is another reason is that the number of malicious samples we have collected is very small. In our future work, we will extract and analyze some structural features and collect more malicious network traffic. In addition, we will apply some cost-sensitive methods to detect malicious network behavior using highly imbalanced network traffic.

VII. CONCLUSION

We design and implement a control and management system of Android traffic collection, which can collect network traffic efficiently and manage the dataset easily. The system is divided into four modules. The pre-processing module can achieve downloading and detecting APK files through Python scripts embedded in the J2EE platform. Traffic collection module uses multi-threading technology to improve the efficiency of traffic collection. Post-processing module contains extracting stream file function and resources sharing function. The last module is a management system, it has a data management function and an access control function. We analyze the network traffic dataset which we have collected through our system and address that using network traffic to train machine learning model is an imbalanced learning problem. What's more, we apply four imbalanced algorithms to Android malware detection using highly imbalanced network traffic dataset and compare the experimental results. The result of experiments indicates the combination of SMOTE and SVM are the best performer in the all combinations.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grants No.61672262, No.61573166, No.61472164 and No.61572230, the Natural Science Foundation of Shandong Province under Grants No.ZR2014JL042 and No.ZR2012FM010, the Shandong Provincial Key R&D Program under Grants No.2016GGX101001.

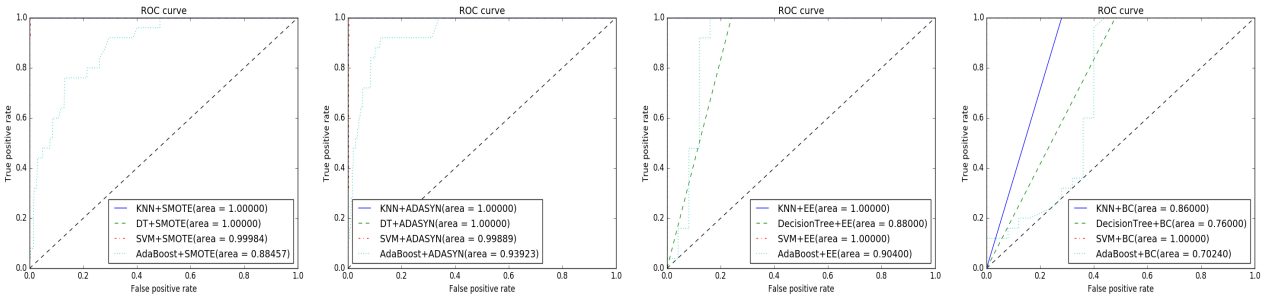

Fig. 7. The ROC curves of the imbalanced methods on training set.

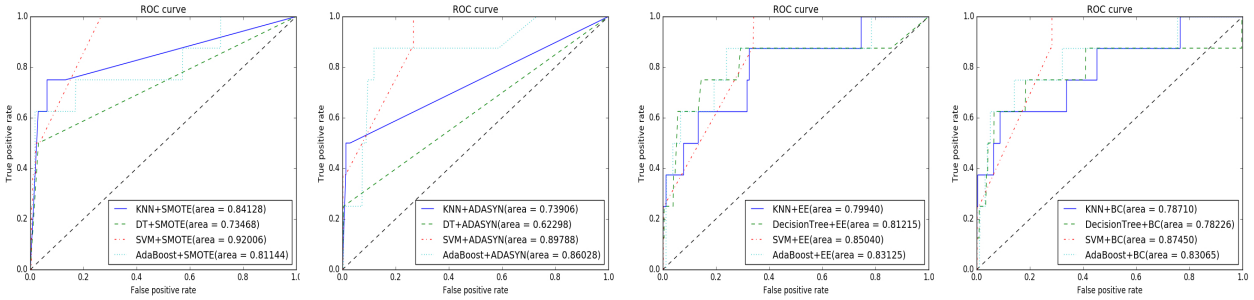

Fig. 8. The ROC curves of the imbalanced methods on testing set.

REFERENCES

[1] Gartner Says Tablet Sales Continue to Be Slow in 2015.2015;Available from: http://www.gartner.com/newsroom/id/2954317.

[2] Virustotal:Available from: https://www.virustotal.com/

[3] Arp, Daniel, et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket." Network and Distributed System Security Symposium 2014.

[4] Perdisci, Roberto, W. Lee, and N. Feamster. "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces." Usenix Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, Ca, Usa DBLP, 2010:391-404.

[5] Arora, Abhishek, S. Garg, and S. K. Peddoju. "Malware Detection Using Network Traffic Analysis in Android Based Mobile Devices." Eighth International Conference on Next Generation Mobile Apps, Services and Technologies 2014:66-71.

[6] Wang, Shanshan, et al. "TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic." Ieee/acm, International Symposium on Quality of Service ACM, 2016:1-6.

[7] Wang, Zhanyi. "The Applications of Deep Learning on Traffic Identification." BlackHat USA (2015).

[8] Peng, Lizhi, et al. "Imbalanced Traffic Identification Using an Imbalanced Data Gravitation-based Classification Model."Computer Communications (2016).

[9] jython Juneau J, Baker J, Wierzbicki F, et al. The definitive guide to Jython : Python for the Java platform[M]. Apress, Springer-Verlag [distributor], 2009.

[10] He, Haibo, and E. A. Garcia. "Learning from Imbalanced Data." Knowledge & Data Engineering IEEE Transactions on 21.9(2009):1263-1284.

[11] Cover, T., and P. Hart. "Nearest neighbor pattern classification." IEEE Transactions on Information Theory 13.1(1967):21-27.

[12] Breiman, L., et al. "Classification and Regression Trees." Biometrics 40.3(1984):358.

[13] Cortes, Corinna, and V. Vapnik. "Support-vector networks." Machine Learning 20.3(1995):273-297.

[14] Freund, Y. "Experiments with a new boosting algorithm." Thirteenth International Conference on Machine Learning 1996:148–156.

[15] Kononenko, Igor. "Semi-Naive Bayesian Classifier." European Working Session on Machine Learning Springer-Verlag, 1991:206-219.

[16] Pedregosa, Fabian, et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research 12.10(2012):2825-2830.

[17] Fawcett, Tom. "ROC Graphs: Notes and Practical Considerations for Researchers." Machine Learning (2004):1-38.

[18] Fawcett, Tom. "An introduction to ROC analysis." Pattern Recognition Letters 27.8(2006):861-874.

[19] Liu, Tian Yu. "EasyEnsemble and Feature Selection for Imbalance Data Sets." International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing IEEE, 2009:517-520.

[20] Liu, X. Y., J. Wu, and Z. H. Zhou. "Exploratory undersampling for class-imbalance learning." IEEE Transactions on Systems Man & Cybernetics Part B Cybernetics A Publication of the IEEE Systems Man & Cybernetics Society 39.2(2009):539.

[21] Chawla, Nitesh V, et al. "SMOTE: synthetic minority over-sampling technique." Journal of Artificial Intelligence Research 16.1(2011):321-357.

[22] He, Haibo, et al. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning." IEEE International Joint Conference on Neural Networks IEEE, 2008:1322-1328.

[23] imbalanced-learn: http://contrib.scikit-learn.org/imbalanced-learn/