Android Fragmentation

Parul Bansal
Department of Computer Engineering
Poornima University, Jaipur, Rajasthan, India

Swati Paliwal
Faculty Coordinator in Computer Engineering Department
Poornima University, Jaipur, Rajasthan, India

Abstract: A mobile operating system becomes fragmented when there are several different OS versions in use at a same time. Android Fragmentation is usually associated with android because wireless carriers and device manufacturers, when OS updates are transfer to different devices which are not only controlled by OS developers. Android bugs report submitted by android users span across operating system versions and hardware platforms. There are two popular vendors that is HTC and Motorola. Here, we use two terms i.e. Labeled LDA(Latent Dirichlet Allocation) on the labeled data and LDA on the original data. The threat or concern that a proliferation of diverging variants of the android platform will result in the in ability of some devices to properly run apps written with the Android SDK. Mobile device fragmentation is the problem which occurs when users run older android on their devices while other users are running newer versions. The peril of fragmentation is basically related to security issues in android device driver customizations. The study is based on ADDICTED a new tool build for automatically detecting some types of flaws in customized driver protection. In order to increase the portability of application, the development of an android application required an efficient porting process. For extracting an ideal behavior of an application, we use behaviour- based portability. The word "fragmented" implies an idealized whole that has been broken into pieces. Hardware developer say that fragmentation is caused by a wrong idiology instructions for the software developers. Software developers say that fragmentation is caused by variety of versions of a particular operating system. Mosaic is a new technology which solve the problem of fragmentation in mobile through novel virtual abstraction. It allows user interaction traces to be recorder on emulators. Using Mosaic we were able to replay 45 different Google Play applications across multiple devices.

Keywords:-Android Fragmentation Problem; portability; behavioural analysis; ADDICTED tool; LDA; Labeled LDA.

I. INTRODUCTION

Android in the latest mobile operating system in the context of openness of source code. Based upon the open source code ,developers need less efforts for application software development[1]. Fragmentation is when a combination of software and hardware do not contain logical contradiction, top-level experience for the vast majority of its user-base. The cause of fragmentation is when a perfect combination of both software and hardware is made available to consumers. According to this, various different various of mobile operating systems, inconsistent APIs among different platforms as well as existence of different UI skins make this problem more serious. The structural definition is shown in Fig.1.

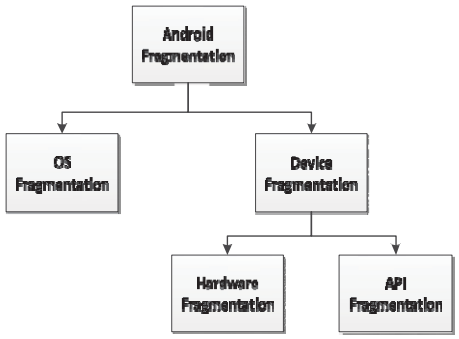

Fig.1.Android Fragmentation Problem

The cause of OS fragmentation is diverse versions of a particular operating system. According to recent study,

OS fragmentation problem has been resolved result of reduction of old version devices. Device fragmentation is a very big issue. Hardware fragmentation causes due to instability among various hardware specifications[2] in mobile devices. API fragmentation occurs due to change in the underlying API according to different thoughts and techniques of various manufacturers and service providers. Many of the researchers test android applications automatically with different testing approaches such as Record playback, Random test. In this, there are two types of analysis i.e. portability, behavioral analysis. Behavior-analysis is based on forensic investigation methods for addressing security issues. It is effective to detect the abnormal behavior of applications. In behavior based portability analysis, we define the expected ideal behavior by inspecting its platform and illuminate the comparison of each application operation flow. Android devices has been influenced global smart phone shipments with more than 70% market share[3].Our study is based upon an automatic tool ,ADDICTED. Which we designed to detect customization hazards like Security risks and Flaw detection. The high-level idea is to automatically identify the Linux files related to the operations on the devices. To implement the idea, we set up into ADDICTED a component called Device Miner that dynamically plans permission-protected device operations on the Android framework layer to their related files on the Linux layer. Device Miner can further fingerprint a device node with a set of system calls involving the file and their parameters. We evaluated ADDICTED on a Google Nexus 4 and 4.2 and 4.3 and Samsung personalized Android on Galaxy SII,ACE 3 and GRAND.

Our objective in this study is to search for the evidence of Android Fragmentation within the bug reports submitted by the users of android devices. A number of topic analysis methods have been used by the researchers in software engineering including Latent Dirichlet Allocation (LDA), Latent Semantic Indexing (LSI)[4] and Labeled Latent Dirichlet Allocation(Labeled-LDA).We applied both Labeled-LDA and LDA topic analysis to the two sets of vendor-specific bugs and then we compared the two sets of topics unique to each vendor are concrete evidence of fragmentation.

As we demonstrate, inspite of the fact that existing replay tools work out of the box for recording and replaying activity on a single device, they do not provide cross platform portability. User actions from one devices cannot be replayed on other devices. To get the better of these issues, we present Mosaic, a cross-platform user input record and replay tool for Android-based mobile devices and applications. Mosaic envisages user intercommunication in a way that allows the application use cases to replayed across a variety of mobile devices running on android platform, each with different hardware and software attributes pertaining to fragmentation.

II. RELATED WORK

A. Code Level Check Method

Code Level Check Method is conceived in order to pick up the fragmentation problem on the code level. The fragmentation pattern under consideration can be frequently detected among android devices. This category of android fragmentation problem happens when a developer does not identify the difference among various mobile devices so that the developer does not optimize the code according to device characteristic. In the method, the source code is examined and the locations that cause fragmentation problem in devices are detected.

For this method, the source code needs to be modified into an appropriate form that contains important itemized values. Moreover, the correlating input values are contrasted to the composed conditions in the prior-defined rule collection. If the contrasting results are true, the source code would be corrected with proper intension process for addressing the android fragmentation problem. For this, we first explain the important terms:

- C is a source code for an application under analysis.
- *Set* c_i $(1 \quad i \quad n)$ is a list extracted from C .
- R is a collection of rules. Each composite condition P and corresponding solution S is included in R such that $R_j = P_{jk} \cup S_j (1 \le j \le m, 1 \le k \le P_j)$

B. Latent Dirichlet Allocation (LDA)

The Latent Dirichlet Allocation (LDA) is a famous prospected unsupervised algorithm that models each document as a mixture of topics[5].LDA automatically learns a set of terms for each topic from a corpus without any constraints. Given a set of documents and the number of topics n, LDA produces the probability distribution of word-topic and the individually distribution of the topic document. It often produces some topics that are hard to interpret, and it is difficult to generate topics that suit a specific purpose. In addition, it needs the number of topics n as an input, but the excellent number of topics can be subjective.

C. Labeled-LDA

Labeled-LDA is an extension of LDA. Labeled-LDA discovers a set of topics by restricting the topic model

to use only those data that is related to a document's label set .Like LDA, Labeled-LDA models each document as a mixture of bundled topics and produces each word from one topic. Unlike LDA, Labeled-LDA is a supervised algorithm that generate topics using the handmade-assigned labels. Therefore, Labeled- LDA can obtain meaningful topics, with words that map well to the labels applied[6]. While these studies used LDA to extract topics, we applied both Labeled-LDA and LDA to get the topics. In our process of research, we take help of the Stanford Topic Modeling Toolbox's (STMT)[6] implementation of Labeled-LDA. We first operate labeled the bug reports with multiple labels and then employed Labeled-LDA to associate topics and documents with the labels we provided.

D. Behaviour-based portability analysis

The aim of this paper is to analyze the quality and efficiently with each platform-specific test result. Figure 2 show the process involved in behavior-based analysis. Behavior-based portability analysis defines normal states of application as ideal method and surely will of execution of any abnormal behavior. Behavior-based portability analysis defines normal states of application as ideal method and surely will of execution of any abnormal behavior. Analysis processes includes of method calculation, test execution, and log analysis. Behavior calculation step necessarily is a process that records logs from the application that is executed on a main platform. The main platform is created on development time.

With the verification and validation processes, application execution flows on the main should be declared to be on the normal state. The ideal behaviour has information generated from applications that are executed at run-time.

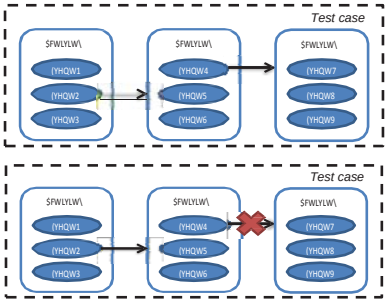

Fig 2. Comparison of ideal and abnormal behaviour

E. Touch Input Architecture of Android

A high-level overview of how Android bridges user interactivity to the underlying application code in shown in Fig2. The user communicates with application user interface (UI) elements located at various display coordinates. The touch screen reports user interaction state updates directly to Android's Linux kernel through events. In addition, the user may also exercise multi-finger gestures, such as a pinching or rotation, where multiple fingers concurrently execute these primitives. All the while, the touch screen tracks finger state atop the display throughout the interaction.

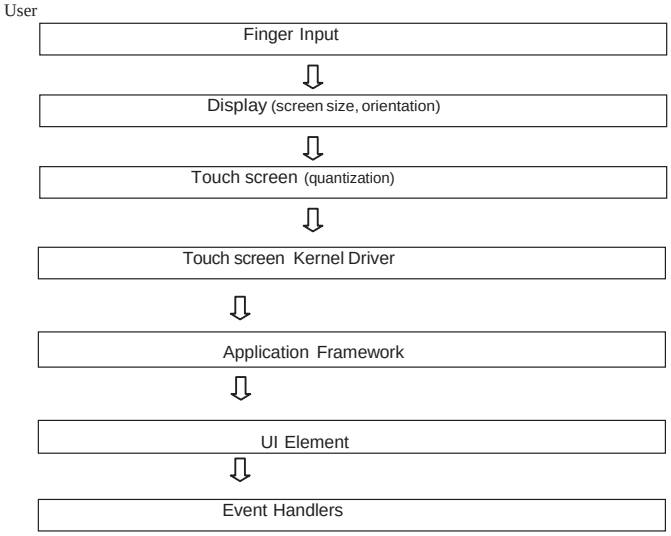

Fig. 3: Android touch screen input architecture overview

III. METHODOLOGY

Our method for carry out research or study into Android fragmentation using topic analysis involved the following steps:

1). First, sets of vendor-specific bug reports are calculated from the Android bug repository.
2).Next, each bug report is manually labeled using feature- based terms used by Android developers.
3). Third, we apply LDA to the basic bug-report sets and Labeled-LDA to the declared sets, as developed in step 2.
4). Next, we calculate and visualize the average relevance of each bug report to each topic over time.
5).We then match the above results between the two venders as in order to look for how fragmentation is manifested through an analysis of common and unique topics.
6). Finally, we also compare the performance of LDA topics versus Labeled-LDA topics by matching the equality of each pair of topics from LDA and Labeled-LDA.

A. Comparison of LDA and Labeled-LDA

Its compulsory because Labeled-LDA took around 60 times the total time it took to identify the topics calculated by plain LDA.[7]. We inspected the way in which the things are shared of bug reports are connected with topics and labels more near and provided summaries in Figures 4 and 5. Figure 4 shows the amount of defect reports that are associated to the same labels in the bug reports of HTC and Figure 5 illustrates the number of bug reports that associated to the similar labels in the bug reports of Motorola. The number of bug reports associated to same labels in LDA and Labeled-LDA are different, this is confirmed by the X2 tests ($p < 0.01$). From these two Jacquard equality plots of topics and labeled-topics between LDA and Labeled- LDA, we can observe that most of the Jacquard comparability values are quite small except a few diagonal ones, especially in HTC.

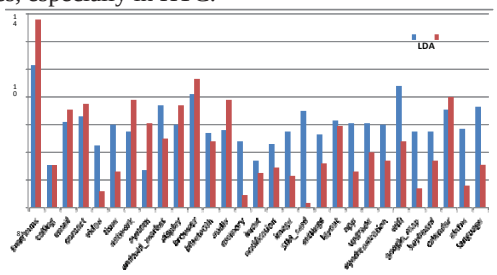

Fig. 4. Comparison of number of bug reports related to the same labels from LDA and Labeled-LDA in HTC

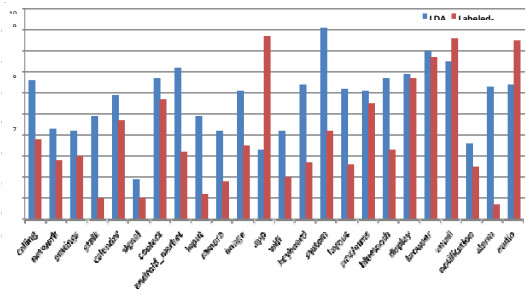

Fig. 5 Comparison of number of bug reports related to the same labels from LDA and Labeled-LDA in Motorola.

For LDA and Labeled-LDA content including similar label, we found that LDA predicted fewer of the relevant bug reports and the connection between topics and bugs for LDA and Labeled-LDA was often very dissimilar. We discover the Labeled- LDA topics are of good quality and seems equal for better to our understanding; but we found that Labeled-LDA need up to 60 times the extra energy that labeling LDA extracted topics needed.

B. Design of ADDICTED

Figure 6 illustrates the design discussed above. ADDICTED includes Device Miner, that performs the afore-mentioned dynamic analysis on the test cases running on a customized Android phone, and further analyzes its outputs (including the system-call traces from multiple executions of individual cases) to identify a set of files related to each device.

Device Miner is designed to trace operation on an Android device to identify its related Linux device files.

However, to handle the complicated inter-process communication (IPC) and message passing model within Android, it requires intensive instrumentations of the OS. As a result, it becomes less portable and unsuitable for analyzing a large number of customized phones. Alternatively, Taint Droid[8] Scope performs the analysis within a virtual machine, however, it cannot conveniently simulate different types of hardware on customized phones or tablets.

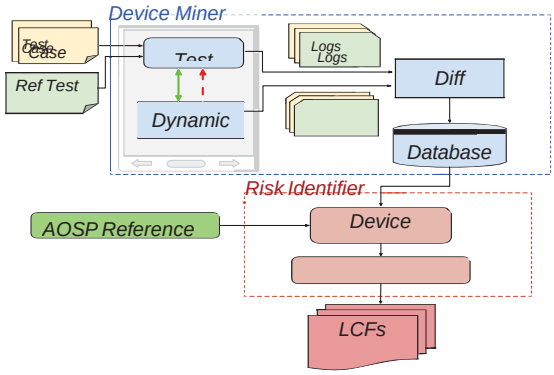

Fig. 6. Design of ADDICTED. Test cases are first executed by Test Runner and analyzed by the Dynamic Analyzer.

We built into Device Miner a dynamic analyzer that works on the system-call level. This makes coarse-grained in tracking device operations, but much more lightweight and portable than prior approaches. More specifically, Device Miner uses a collection of test cases to trigger device operations such as taking pictures, requesting geolocations, etc., and join the set of traces to the app that runs those cases.
Dynamic analysis As soon as Device Miner starts the Test Runner, it join set of traces to the application step.. The app needs to make API calls to access its target devices. In Android, such an API call goes through the binder driver in the kernel, which passes the request to a system service. This interaction is called a transaction as shown in Figure 6. Device Miner includes an instrumented binder that monitors the processes communicating with the test app and attaches tracers to them. During its runtime, this modified binder checks the trans- action parameters of an IPC call to extract the source. Process Identifier (PID) of the transaction and its target PID, together with the transaction data.

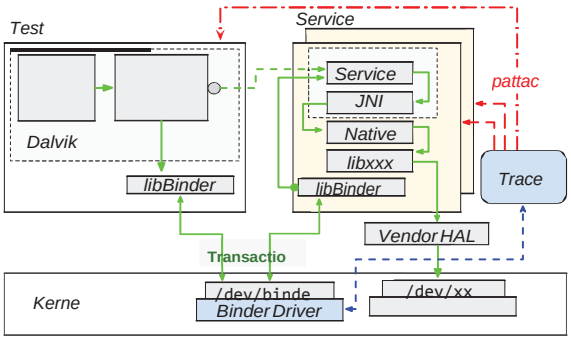

Fig.7 Dynamic Analyzer

Specifically, our approach instruments binder.c with the code for inspecting individual transactions. This can be done automatically, given the binder's source code has not been changed significantly across different Android versions. Later we will discuss an alternative to avoid even this mostly automated instrumentation. Mosaic Design , a cross-platform user input record and replay tool for Android. To overcome the fragmentation issues discussed in the previous section, Mosaic provides portability through virtualization. User inputs are captures on a host device which is then virtualized into a platform- agnostic intermediate representation which can then be retargeted for specific mobile devices.
Formalizing Touchscreen Interactions

Mosaic abstracts touch screen-specific input events into a set of user interactions on a virtualized touch screen. To perform this conversion systematically we develop a formalization for touch screen input events. Interactive mobile applications rely on user input (i.e. interactions) to drive program behavior. More formally,

we denote this set of interactions as {I1, I2, ..., IN }, where an arbitrary Ij corresponds to an individual interaction primitive – to be defined later. While the touch screen driver propagates user input information as individual events, we make the observation that the touch screen implicitly packetizes these events into user interaction primitives. We identify three user interaction primitives that serve as Mosaic's platform-agnostic user interaction intermediate representation. The primitives consist of a finger press, release or movement.

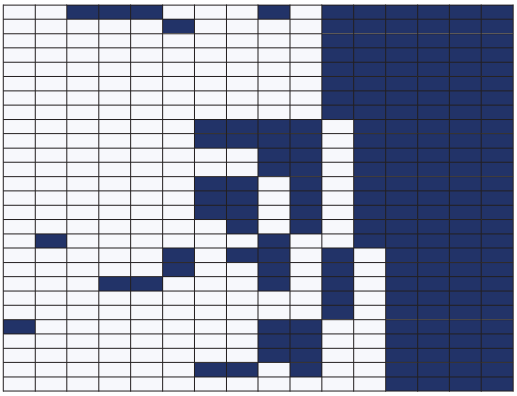

Fig.8 Touch Screen Driver Event

While the touch screen driver propagates user input information as individual events, we make the observation that the touch screen implicitly packetizes these events into user interaction primitives.The Linux/dev/input/event/ interface terminates a sequence of events with a SYN_REPORT event delimits different user inputs. Therefore, we define any arbitrary interaction primitive Ij as a set of events {e1, e2, ..., eN}, where EN is the only SYN_REPORTevent in the set. There- fore, Mosaic relies on SYN_REPORT to identify interaction primitives throughout the sequential touch screen event stream.

IV. CONCLUSION

In this study we found how fragmentation is manifested within Android by comparing and contrasting the bugs topics, extracted from Android bug reports, of two Android smart- phone vendors: HTC and Motorola. . We used our labeled bug reports to compare Labeled- LDA and LDA, as LDA is unsupervised and requires far less effort to run than Labeled-LDA. We found that LDA (with manually labeled topics) and Labeled-LDA produced some similar topics. The drawback of existing research Android portability analysis is that test results analysis consume a significant amount of time and hardly captures any potential errors. This study is used to present a faster result analysis of a number of test results. Thus, developers can effectively do the porting process. As a result, the methodology could be based for the quality assurance of Android applications. In our research, we made the first step toward better understanding of this issue, leveraging a new technique, ADDICTED, designed for automatic detection of some types of security-critical customization flaws. ADDICTED dynamically analyzes the operations on a sensitive Android device to connect it to a set of Linux device files. Our research just scratches the surface of the grand security challenges that come with Android customizations. Even on the Linux layer, still there are many device files we cannot interpret, not to mention detection of their security flaws. Mosaic is a cross- platform, timing accurate user interaction record and replay tool for android. Mosaic can enable researchers and developers to perform cross-device performance evaluations and analyze different type of user interactivity. As a mobile device ecosystem becomes increasingly fragmented to meet the diverse needs of users, the need for tools like Mosaic and will become incresingly important.

REFERENCES

[1] Google Android, http://www.android.com/, Acceesed on Sep. 14, 2011.

[2] ScreenSizesandDensities,http://developer.android.com/resources/dashd/screens.html, Acceesed on Sep. 19, 2011.

[3] Android tops 81 percent of smartphone market share in q3. http://www.engadget.com/2013/10/31/ strategy-analytics-q3- 2013-phone-share/,2013. Accessed: 10/31/2013.

[4] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An Information Retrieval Approach to Concept Location in Source Code," in *In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)*. IEEE Computer Society, 2004, pp. 214–223.

[5] D. Ramage, D. Hall, R. Nallapati, and C. D. Manning, "Labeled LDA: A Supervised Topic Model for Credit Attribution in Multi-labeled Corpora," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, August 2009, pp. 248– 256

[6] "Stanford Topic Modeling Toolbox," [Accessed 11-July-2012]. [Online].Available: http://nlp.stanford.edu/software/tmt/tmt-0.4

[7] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Annotated Topic Data Used in this Study," http://softwareprocess.es/ static/Fragmentation.html, 2012.

[8] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jayson Jung, Patrick McDaniel, and An- mol N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.