Vol.2018-GN-103 No.31
Vol.2018-CDS-21 No.31
Vol.2018-DCC-18 No.31
2018/1/27

加速 Android 環境におけるシステム安定性の改善

福田翔貴[†1]　栗原駿[†1]　神山剛[†2]　福田 晃[†2]　小口正人[†3]　山口実靖[†1]

Android 端末の普及に伴い，膨大な数の Android OS 向けアプリケーションが開発，配布されている．これに伴いアプリケーション動作解析の重要性が増しているが，動的動作解析には長い時間を要することが問題となっている．この問題に対して我々は過去の研究において Linux カーネルを改変しシステム内部の時間の流れを加速することにより動的動作解析時間を短縮する手法の提案と低加速倍率における評価を行った．本稿においては高加速倍率環境における安定性の評価と改善手法の提案を行う．

1. はじめに

近年，スマートフォンやタブレット PC などのモバイル端末向け OS として Android OS が世界的に広く普及し，膨大な数の Android OS 向けアプリケーションが開発, 配布されている．これに伴って Android OS 向けアプリケーションの動作観察も重要性を増している．アプリケーションの動作観察はアプリケーション配布サイト運営者やアプリケーション開発者などにとって非常に重要な事項である．

アプリケーションの動作観察には大きく分けて静的解析と動的解析がある．静的解析はアプリケーションを実際に動作させることなく，アプリケーションの実行ファイルやそのメタファイルを解析して観察対象アプリケーションの振る舞いを予測する手法である[1][2]. 一方で動的解析は実際にアプリケーションを実行し，観察対象アプリケーションの動作を観察する手法である[3][4][5][6]．静的解析には観察対象アプリケーションを実行することなく解析を行うことができるというメリットがあるが，より現実に近い振る舞いを観察し精度の高い解析結果を得るためには動的解析を行うことが好ましいと考えられる．しかしながら，動的解析は非常に長い時間を要することが問題となっている．例えば，観察対象アプリケーションを無操作状態で 24 時間放置したときの振る舞いを観察する場合は観察時間として 24 時間が必要となる. 膨大な数のアプリケーションを解析する必要がある場合や繰り返し観察する必要がある場合は，この観察時間の長さが深刻な課題となると予想できる．

この課題に対して我々は過去に，アプリケーション動的動作解析時間の短縮手法を提案した．具体的には Android OS の Linux カーネルを改変し, 端末内部の時間の流れを現実の時間よりも速くすることによってアプリケーションが認識する時間を加速し，結果としてアプリケーション動作観察時間の短縮を可能とする手法を提案した[7]．この手法を適用した環境にて，単一クライアント環境アプリケーションを用いた評価[7]や，クライアント・サーバ型アプリケーションを用いた評価[8]を行い，特殊なハードウェアや仮想環境などを用意することなく実機上にて動作観察時間を短縮できることを示してきた．また，加速倍率と観察精度の関係や, 加速倍率とシステム安定性の関係の調査を行い，高い加速倍率においてはシステムの安定性が低下することを確認した[9].

本稿では，高倍率加速環境の安定性を改善することを目的とし，Android OS の安定性を向上させる手法の提案と実装を行い，評価によりその有効性を示す．具体的には，高倍率仮想環境において生じるシステム障害の原因がシステムプロセスの強制終了処理にあることを示し，それを回避する手法を提示し, その実装を用いて安定性の向上を示す.

本論文の構成は以下の通りである．2 章にて，カーネルの改変による加速観察環境の実現などの Android アプリケーションの動作観察に関する既存の研究を紹介する．3 章にて，高倍率加速環境のシステム安定性に関する既存の研究の紹介を行う．4 章にて，本稿の研究にて新たに行った高倍率加速加速環境におけるシステム不安定の原因の調査の結果を示す．5 章にて前章の原因に対応する手法を提案し，6 章にてその評価結果を示す．7 章にて考察を述べ，8 章にて関連する研究を紹介し，9 章にて本稿をまとめる．

2. Android アプリケーション観察

2.1 Android アプリケーション

Android はスマートフォンやタブレット PC などのモバイル端末向け OS の一つであり，2017 年第 1 四半期におけるスマートフォン向け OS としての全世界でのマーケットシェアは 85.0%となっており[10]，重要なプラットフォームの一つとなっている．

Android アプリケーションは Google Play Store[11]などで配布され，2017 年 9 月時点で Google Play Store には 330 万件のアプリケーションが登録されている[12]．この様に膨大な数のアプリケーションが提供されており，個人がアプリケーションの動作を検証することは困難となっている．また，サービスの一部としてアプリケーションの動作検証を行っているアプリケーション配布サイトも存在しており[13]，これら配布サイトにとってアプリケーションの動作

†1 工学院大学
Kogakuin University
†2 九州大学
Kyushu University
†3 お茶の水女子大学
Ochanomizu University

ⓒ 2018 Information Processing Society of Japan

Vol.2018-GN-103 No.31
Vol.2018-CDS-21 No.31
Vol.2018-DCC-18 No.31
2018/1/27

解析は重要な事項であると考えられる．しかし，アプリケーションを実際に動作させての挙動調査（動作解析）を行うには膨大な時間を要することがわかっており[3][4][5][6]，すべてのアプリケーションに対して動的手法を用いたアプリケーション挙動調査を実行することは困難であると考えられる．また，アプリケーション開発者にとっても自分が開発したアプリケーションの動作の確認は重要な事項であり，この時間の長さは大きな課題であると考えられる．

2.2 Android における時刻管理

Android OS を含む Linux カーネルを用いる OS では，カーネルにおいて時間と時刻が管理され，システム内プロセスはシステムコールなどにより時間や時刻の情報をカーネルより得ている．

Linux カーネルでは，ハードウェアから供給されるクロックソースをもとに時間と時刻を管理している．クロックソースには複数の種類があり，本稿の実験で使用した環境では gp_timer もしくは dg_timer が用いられている．

クロックソースから得られた時刻情報は変数 `cycle_now` に格納され，この変数の増分が時間や時刻に加算される．Tickless でないカーネルにおいては，時間や時刻の更新は tick (Linux カーネルの周期的なタイマ割り込み間隔)ごとに行われ，`cycle_now` の増分が時間や時刻に加算される．tick 単位は Linux カーネルのパラメータの 1 つであり，コンパイル時に指定することができる．Android OS では多くの場合 tick 単位は 10m 秒(100Hz)とされている．Tickless のカーネルにおいては更新が tick ごとにならないが，同様にクロックソースの増分が時間や時刻に加算される．

2.3 動的解析による消費電力の大きいアプリケーションの推定

文献[3][4]において，アプリケーションの動作解析による消費電力の多いアプリケーションの発見手法が提案されている．

同文献では，無操作状態のスマートフォンでもアプリケーションが動作し，バッテリを消費することに着目し，無操作状態にて多くのバッテリを消費するアプリケーションを発見する手法について考察している．具体的には，端末がスリープ状態に移行するのを妨げる WakeLock や，指定時刻に処理を呼び出す仕組みである Alarm セットの発行回数の観察などにより無操作状態消費電力を大きく増加させるアプリケーションの発見が可能であることと，実際にアプリケーションを動作させる動的解析には多くの時間がかかることが示されている．

2.4 アプリケーション観察時間の短縮

アプリケーションの動的動作解析の解析時間の短縮手法として，我々は Android OS のカーネルの時刻管理実装を改変しシステム内のアプリケーションが認識する時間の流れを速くすることにより観察時間を短くする手法を提案した[7]．当該手法では，前述の Linux カーネルにおけるクロックソース管理変数 cycle_now の増加速度がクロックソースの増加速度よりも高くなるように修正し，システム内の時間の流れる速度を高めている．また，提案手法を実際に Android OS に実装し，実スマートフォン端末にインストールし，加速機能付き Android 端末上でベンチマークアプリケーションの動作の検証を行い，対象アプリケーションについて正しく加速観察が実現できることを確認している．

また，本手法のネットワークを用いるクライアント・サーバ型ベンチマークアプリケーションによる評価[8][14]や，ネットワークを用いる実アプリケーションを用いての評価[17]を行い，対象としたアプリケーションの加速が本手法により正しく行われたことを確認している．

3. 高倍率加速環境でのシステム安定性評価

2 章で紹介した評価はいずれも実時間比で 2 倍などの低い倍率における評価である．我々はより高い倍率における評価も必要であると考え，高倍率加速環境における評価や考察を行った[9]．本章にて，文献[9]における高倍率加速環境におけるシステム安定性の評価を紹介する．

3.1 高倍率加速環境における観察精度

まず我々は，高倍率加速環境における加速倍率と観察精度の関係性について評価を行った[9]．Android 端末内時間加速手法の加速倍率を 2, 6, 24, 48, 60 倍と変化させ，高倍率加速が測定結果に与える影響を調査した．実験は，実験用アプリケーションをインストールし無操作状態で端末を 1 時間放置し，その間に実行される Alarm セットと WakeLock を観察することにより行った．実験用アプリケーションには，2015 年 10 月 24 日付 GooglePlay ニュース&雑誌カテゴリ上位 10 件を使用した．計測環境は表 4 のとおりである．Alarm セットと WakeLock は，その観察がバッテリ消費を増やすアプリケーションの特定に重要であることが確認されている[3][4]ため，これらに着目してアプリケーションの観察を行った．計測環境は表 1 のとおりである．

観察結果は図 1～図 3 の通りである．図 1 より，加速倍率 6 倍以下においては Alarm セット回数や WakeLock 回数の通常状態との差異が小さくなっていることがわかる．そして，24 倍以上においては差異が大きくなっていることも分かる．確認された最大の差異は Alarm セット回数において 14.9%，WakeLock 回数において 26.6%（いずれも加速率 60 倍時）であり，要求される精度がこれより低い状況では 60 倍の加速による観察も可能であることが分かる．

文献[9]の計測においては，頻繁に Alarm や WakeLock を用いるアプリケーションの検出という目的は達成しており，上記誤差は許容できる範囲となっている．

表 1　計測環境

Device Name	Nexus7 (2013)
OS	Android 5.0.1 (AOSP)

ⓒ 2018 Information Processing Society of Japan

Vol.2018-GN-103 No.31
Vol.2018-CDS-21 No.31
Vol.2018-DCC-18 No.31
2018/1/27

	with modified Linux kernel 3.4.0

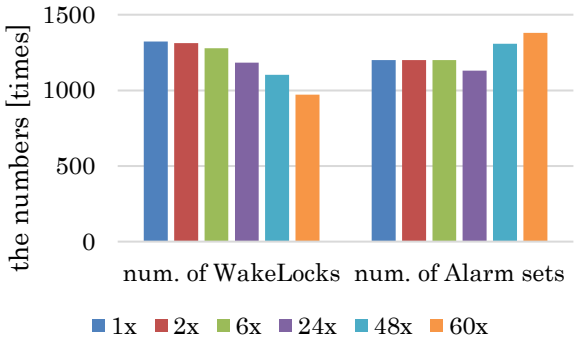

図 1 Alarm セット回数と WakeLock 回数 [9]

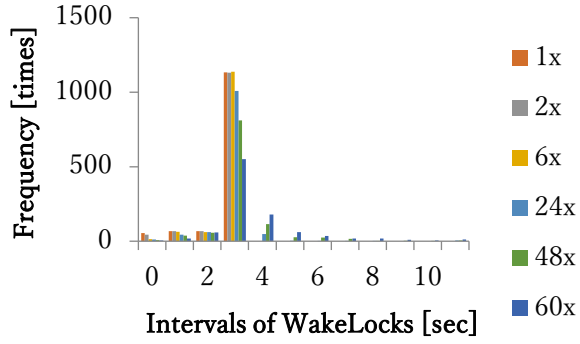

図 2 加速倍率による WakeLock 間隔の変化 [9]

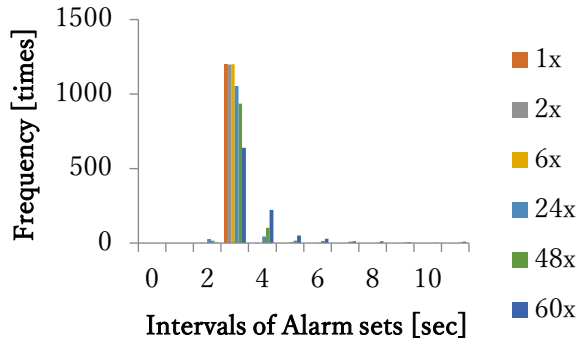

図 3 加速倍率による Alarm セット間隔の変化[9]

3.2 高倍率加速環境における安定性

次に我々は，高い加速倍率とシステムの安定性の関係を調査した[9]．加速倍率を 60, 120,180, 240 倍と変化させ，無操作状態で端末内時間にて 1 時間 Launcher 画面表示を維持できた回数を計測することにより安定性を評価した．加速は OS(カーネル)の起動後 30 秒前後たってから有効化し，測定ごとに OS の再起動を行った．画面表示の確認は目視にて行った．測定環境は表 2 のとおりである．

観察結果は図 4 の通りである．図の横軸は加速倍率を，縦軸は端末内時間で 1 時間経過後も Launcher 画面表示を維持できた割合を示している．図より，およそ 60 倍までの加速倍率においては高倍率加速環境においても Launcher 画面表示を端末内時間で 1 時間維持できることがわかる．しかしながら，120 倍以上の倍率においては Launcher 画面表示を維持できた割合が下がっており，システムの安定性の低下が確認できる．

この評価において発生した障害はすべてユーザ空間で動作する Android フレームワークプロセスの再起動である．Linux カーネルの停止は確認されず，PC からの ADB 接続が途切れることはなかった．

表 2 計測環境

Device Name	Nexus7 (2013)
OS	Android 5.1.1 (AOSP) with modified Linux kernel 3.4.0

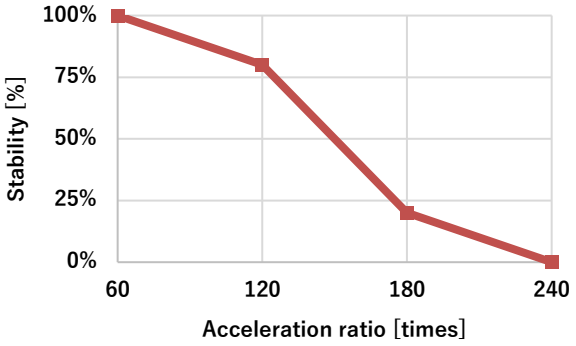

図 4 加速倍率とシステム安定性 [9]

3.3 高倍率加速環境の安定性の改善

次に，システムの安定性に障害をもたらした要因についての考察をおこなった[9].

logcat を用いて Android フレームワークが再起動したときの Android OS のログを取得した結果，Activity Manager やウォッチドッグタイマによるシステムプロセスのキル(強制終了)メッセージが確認された．これらにより，Android フレームワークが強制終了され再起動を行っていると考えられた. 加速環境では CPU などの処理能力が上昇していない一方で，端末内部の時間のみ速く進むことになり，通常はソフトウェアのバグ等以外では検出されないタイムアウト等が高倍率加速の環境においては検出されたと考えられた. この問題を回避するために Android OS を改変し，加速倍率 n 倍のときにウォッチドッグタイマによるタイムアウト時間を n 倍に拡大するという対策と，Activity Manager によるプロセスキルを無効化するといった対策を行った.

対策後のシステム安定性の観察結果は図 5 の通りである. 観察環境は前節と同様である．図より対策によって安定性が高まっていることが確認できる．しかし 240 倍以上では安定性が低くなっていることも確認できる.

ⓒ 2018 Information Processing Society of Japan

Vol.2018-GN-103 No.31
Vol.2018-CDS-21 No.31
Vol.2018-DCC-18 No.31
2018/1/27

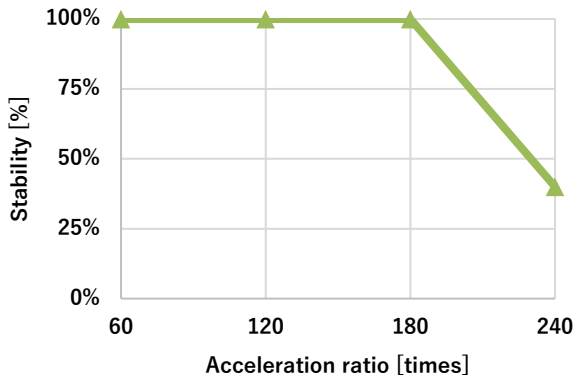

図 5 加速倍率とシステム安定性 [9]

4. 安定性改善状態におけるシステムの再起動

既存手法[9]適用後のシステム安定性の改善を行うため，本稿ではまず同手法適用状態におけるシステムの安定性の評価と，同環境において送信されるシグナルの調査を行う．

実時間比 300 倍の高倍率加速環境にて Android 端末を放置し，Android フレームワークが再起動するまでに発行された全てのシグナルを Linux カーネルにて記録した．計測環境を表 3 に，計測結果を表 4 に示す．

表 4 より，キルシグナル(signal 9)が発行されていることが分かり，これによりシステムの安定性が低下している可能性が考えられる．また，既存研究における観察と同様に，発生した障害は全て Android フレームワークプロセスの再起動であり，これに対応することにより安定性のさらなる向上が可能であると考えられる．

5. システムプロセスキルの無効化

前章より，Activity Manager やウォッチドッグタイマによる Android フレームワークプロセスのキル(強制終了)の無効化を行っても，同プロセスに対するキルが行われていることがわかる．本章では，Android フレームワークプロセスに対して行われるキルの頻度をさらに低下させるために，Android OS 実装に対して以下の改変行う手法を提案する．

図 6 のように Android OS のシグナル発行部を改変し，発行されたシグナルが 9(キル)であり，対象が Android フレームワークである場合は，それを無効化する．

改変したファイルは Android ソースコード内にある `frameworks/base/core/jni/android_util_Process.cpp` である．このファイル内の `android_os_Process_sendSignal()` 関数におけるシグナル送信処理を，シグナルが 9 かつ対象が Android フレームワークという条件が成り立たないときに実行するように改変した．

表 3 計測環境

Device Name	Nexus7 (2013)
OS	Android 5.1.1 (AOSP) with modified Linux kernel 3.4.0

表 4 シグナル発行回数

Signal number	概要	回数 [回]
14	ALRM Alarm clock	358
11	SEGV Segmentation fault	30
17	CHLD Child exited	12
19	STOP Stopped (signal)	3
18	CONT Continue	2
4	ILL Illegal instruction	1
9	KILL Killed	1

6. 評価

本章にて，3.3 節の既存手法[9]に加えて 5 章の本稿の提案手法を実装した環境の安定性の評価を行う．

```
863 void android_os_Process_sendSignal(JNIEnv* env, jobject clazz, jint pid, jint sig)
864 {
865     if (pid > 0) {
866         ALOGI("Sending signal. PID: %" PRId32 " SIG: %" PRId32, pid, sig);
867         //kill(pid, sig);
868         if (pid == getpid() && sig == 9) {
869             ALOGI("NOT Sending signal. PID: %" PRId32 " SIG: %" PRId32, pid, sig);
870         }else{
871             ALOGI("Sending signal. PID: %" PRId32 " SIG: %" PRId32, pid, sig);
872             kill(pid, sig);
873         }
874     }
875 }
```

図 6 改変後のソースコード(867 行目をコメントアウト，868～873 行目を追加)

ⓒ 2018 Information Processing Society of Japan

Vol.2018-GN-103 No.31
Vol.2018-CDS-21 No.31
Vol.2018-DCC-18 No.31
2018/1/27

評価は加速環境にて端末内時間で 1 時間端末を放置し，システムプロセス（Android フレームワーク）の再起動の発生の観察を行った．具体的には，加速開始時と加速終了時に起動中のプロセスを ps コマンドにて取得し，Android フレームワークプロセスのプロセス ID が変化しているか否かを調査した．本評価は Launcher プロセスをフォアグラウンドに表示して行った．計測は加速倍率 64 倍，256 倍，1024 倍, 4096 倍にて行い, 各倍率で 30 回の計測を行った．計測ごとに OS（カーネルを含む）の再起動を実施し，Launcher プロセスが起動してから 60 秒後に加速を有効化した．計測環境は表 5 の通りである．

計測結果を図 7 に示す．Normal は 3.3 節および 5 章の安定化向上対策を行っていない場合，AM+WD は 3.3 節の手法を適用した場合，AM+WD+SIG は 3.3 節の手法に加えて 5 章の手法を適用した場合である．図の横軸は実時間比での加速倍率，縦軸は安定性であり加速状態で端末内時間 1 時間放置した後も Launcher プロセスが存在していた割合を示している．

図より，3.3 節の手法と 5 章の手法をともに適用した環境が最も安定性が高いことがわかる．

表 5　計測環境

Device Name	Nexus7 (2013)
OS	Android 5.1.1 (AOSP) with modified Linux kernel 3.4.0

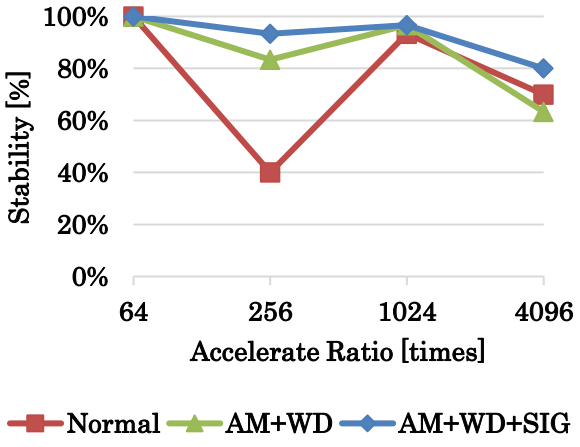

図 7 高倍率加速環境における安定性

7. 考察

本章において，Android OS におけるプロセスキルを無効化したことの影響，およびこれに起因して本手法適用に要求される前提についての考察を行う．

本稿で提案した手法では，Android フレームワークに対するキルを無効化している．これにより Android フレームワークが再起動する回数が減少すると期待できる．ただし，同プロセスの再起動が必要な場合に再起動ができない問題が生じる可能性が考えられる．例としては，Android フレームワーク実装の不具合により再起動が必要となる様な場合が考えられる．ただし，我々が確認できている範囲では現在の Android OS にて Android フレームワークが不具合により再起動する事象は発生しておらず，これは大きな制限にはならないと予想している．同様に，本手法は Android フレームワークの動作の検証(同プロセスが再起動なく安定して稼働し続けることが可能であるか否かの検証)には用いることができないこととなる．また，同プログラムの強制終了を行う悪意のあるアプリケーションの検出なども行うこともできなくなる．一般に作成，配布されているアプリケーションの多くは上記条件と関連がなく，多くのアプリケーションの評価において提案手法は有効であると予想できる．

8. 関連研究

本章にて, 本研究と関連する研究を紹介する. Enck らは，スマートフォンアプリケーションのセキュリティに関する考察のために，1100 個の著名な Android のアプリケーションの調査を行った[16]．彼らはデコンパイラを用いてアプリケーションの解析を行い, フィールド値の調査を行った．しかし，当該研究は静的解析に基づくものであり，動的解析やその時間負荷の軽減に関する考察はなされていない．

アプリケーションの動的解析に基づく研究として，Android におけるアプリケーション動作の動的観察によるバッテリ消費の大きいアプリケーションの推定手法の提案[3][4]がある．当該手法は，無操作状態 Android 端末においてアプリケーションを実際に実行させ，各アプリケーションのアラームの設定などの回数を観察することによりバッテリ消費の大きいアプリケーションの特定を行っている．しかし，当該研究では動的解析には長い時間がかかることが課題となっており，それに関する改善手法の提案はなされていない．Petsas は，セキュリティ用に動的観察を含むアプリケーション解析手法を提案している[17]．同文献では提案手法の実マルウェアへの適用が示され，公開されている実アプリケーションの解析が示されている．当該研究では観察手法の提案のみがあり，観察時間の短縮手法は考察されていない．Yan らは Android 用の仮想化ベースマルウェア解析プラットフォームを提案している[18]．本手法も解析手法を提案するのみであり，解析時間の短縮手法は提案していない．

カーネルにおける時刻管理に関する研究としては，Ferrari らがソフトウェアベースの IEEE1588 実装の評価を行っている[19]．同研究において，クロックソースを含む時刻管理手法にも言及されているが，システム内のプロセスが認識する時間の流れを速くするする手法については言

ⓒ 2018 Information Processing Society of Japan

及されていない.

長時間の動作観察を目的として Linux システムの時間を加速させる研究がある[20]．同文献はカーネルの時刻管理実装を修正することにより加速度テスト環境を構築する開拓的な研究であると言える．しかし，同文献では時間の加速に jiffies を増加させる手法をとっている．現在の Linux kernel においては jiffies を書き換えても端末内時間に影響を及ぼさないことが確認されており，本手法を現在もそのまま採用することはできなくなっている．Linux カーネルを搭載した Android システムの時間を加速した研究として，文献[7][8][9][15][21]がある．しかし，高倍率加速環境において Android フレームワークの強制終了の OS レベルでの無効化については言及されていない.

9. おわりに

本稿では，Android アプリケーション観察時間の短縮を目的としたシステム内時間を加速する手法に着目し，高倍率加速環境下における安定性の向上手法について考察し，改善手法の提案を行った．具体的には，従来行っていた Activity Manager や Watchdog タイマによるプロセスのキルの制御に加え，Linux カーネルに送信されるシグナルを制御することにより安定性の向上が可能であることを示した.

今後は，より多くのアプリケーションによる評価などを行う予定である.

謝辞

本研究は JSPS 科研費 26730040, 15H02696, 17K00109 の助成を受けたものである.

本研究は、JST、CREST JPMJCR1503 の支援を受けたものである.

参考文献

[1] 松崎 悠太，千石 靖 “Android アプリの危険性を見分ける支援ツールの提案”，研究報告マルチメディア通信と分散処理（DPS），Vol.2015-DPS-162，No. 46，pp.1-6，2015 年 2 月 26 日

[2]坂下 卓弥，小形 真平，海谷 治彦 海尻 賢二 “静的解析による Android パーミッションの利用目的の可視化方法”，情報処理学会論文誌，Vol.56，no.1，pp.391-400，2015 年 1 月 15 日

[3]Shun Kurihara, Shoki Fukuda, Ayano Koyanagi, Ayumu Kubota, Akihiro Nakarai, Masato Oguchi and Saneyasu Yamaguchi: “A Study on Identifying Battery-Draining Android Applications in Screen-Off State”, 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE 2015), 2015.

[4]栗原 駿, 福田翔貴, 小柳文乃, 小口正人, 山口実靖 “Alarm の観察による無操作状態携帯端末の消費電力の増加の原因となるアプリケーションの推定”，信学技報, vol. 115, no. 230, DE2015-23, pp. 17-22, 2015 年 9 月.

[5]中村優太，早川愛，竹森敬祐，半井明大，小口正人．山口実靖 “Android 端末におけるインストールアプリケーションとブロードキャストインテント発行による電力消費に関する一考察”，研究報告データベースシステム（DBS），Vol. 2014-DBS-159，No.7，pp.1-6，2014 年 7 月 25 日

[6]橋田 啓佑，金井 文宏．吉岡 克成，松本 勉 “Android の実機を利用した動的解析環境の提案”, コンピュータセキュリティシンポジウム 2014 論文集，Vol.2014，No.2，pp.1000-1006，2014 年 10 月 15 日

[7]Shoki Fukuda, Shun Kurihara, Shintaro Hamanaka, Masato Oguchi and Saneyasu Yamaguchi: “Accelerated Application Monitoring Environment of Android”, 2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), 2016

[8]Shoki Fukuda, Shun Kurihara, Shintaro Hamanaka, Masato Oguchi and Saneyasu Yamaguchi: “An Accelerated Application Monitoring Environment with Accelerated Servers”, 2016 IEEE The 5th IEEE Global Conference on Consumer Electronics (GCCE 2016), 2016

[9]福田翔貴，栗原駿，濱中真太郎，小口正人，山口実靖，“時間加速 Android 環境におけるシステム安定性に関する一考察”，16 回情報科学技術フォーラム(FIT2017)，B-019

[10] IDC: Smartphone OS Market Share, https://www.idc.com/promo/smartphone-market-share/os

[11]Google Play Store, https://play.google.com/store/apps

[12]Google Play Store: number of apps 2009-2017 | Statistic https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[13]Android and Security - Official Google Mobile Blog, http://googlemobile.blogspot.jp/2012/02/android-and-security.html

[14]福田翔貴，栗原駿，濱中真太郎，小口正人，山口実靖, "Android アプリケーション観察を目的としたクライアント・サーバ加速環境”，信学技報, vol. 116, no. 214, DE2016-16, pp. 25-30, 2016 年 9 月.

[15]Shoki Fukuda, Shun Kurihara, Shintaro Hamanaka, Masato Oguchi and Saneyasu Yamaguchi. “Accelerated Test for Applications with Client Application and Server Software”， 2017 The International Conference on Ubiquitous Information Management and Communication (IMCOM2017), 2017

[16]William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. 2011. A study of android application security. In Proceedings of the 20th USENIX conference on Security (SEC'11). USENIX Association, Berkeley, CA, USA, 21-21.

[17]Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos,Michalis Polychronakis, Sotiris Ioannidis , “Rage against the virtual machine: hindering dynamic analysis of Android malware” EuroSec '14 Proceedings of the Seventh European Workshop on System Security, Article No. 5

[18]Lok Kwong Yan, Heng Yin, “DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis” 21st USENIX Security Symposium, 2012

[19]P. Ferrari, A. Flammini, S. Rinaldi, A. Bondavalli and F. Brancati, "Evaluation of timestamping uncertainty in a software-based IEEE1588 implementation," Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE, Binjiang, 2011, pp. 1-6.

[20]Yoshitake Kobayashi, “Linux Kernel Acceleration for Long-term Testing”, CELF Embedded Linux Conference Europe 2010.

[21]Shoki Fukuda, Shun Kurihara, Masato Oguchi and Saneyasu Yamaguchi, “Stability Improvement of an Accelerated Android Operating System for Application Observation,” 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018.

ⓒ 2018 Information Processing Society of Japan