Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

# Android アプリケーション観察の加速環境の構築

福田翔貴[†1]　栗原駿[†1]　濱中真太郎[†2]
小口正人[†3]　山口実靖[†1]

スマートフォン上では多種多様なアプリケーションが動作し，その動作の観察は重要な事項の一つとなっている．しかし，アプリケーションを実際に動作させての動作観察には長い時間がかかり，この時間の短縮は重要な課題の一つとなっている．本研究では，著名なスマートフォン OS の一つである Android OS に着目し，アプリケーションの動作観察を短時間で実現できる加速環境の構築を行う．Android OS はカーネルに Linux カーネルを用いており，同カーネルにより OS 内のプロセスに時刻情報が供給される．よって，同カーネルの改変により OS 内のアプリケーションが認識する時間の流れを実時間の流れよりも速くすることが可能であると考えられる．本稿では本研究の初期段階として，指定した前提を満たすアプリケーションが認識する時間の流れを速くできる環境の構築方法を提案する．そして，評価によりその効果を示す．

**キーワード**：スマートフォン，Android，アプリケーション，アプリケーション動作観察，動的解析

## 1. はじめに

近年，スマートフォンやタブレット PC といった高機能な携帯端末が広く普及し，それらの端末の上で動くソフトウェアプラットフォームである Android OS が注目されている．Android OS の 2015 年第 2 四半期におけるスマートフォン OS の全世界マーケットシェアは 82.8%であり[1]，Android OS が非常に重要なプラットフォームとなっていることが分かる．そのため Android OS 向けアプリケーションも多数開発されており，2015 年 11 月にいて Google Play ストアでダウンロードできる Android アプリケーションの数は 180 万に上っている[2].

このように膨大な数のアプリケーションが入手可能な状況となり，一般のスマートフォンユーザが個別のアプリケーションの安全性の確認などの検証を行うことは容易ではなくなっており，サービスの一部としてアプリケーションの動作の検証を行っているアプリケーション配布サイト(マーケットストア)も存在している[3]．このようなサービスを提供する場合，アプリケーション配布サイト運営者は該当アプリケーションがどのような挙動をするかを把握してからストアに掲載する必要がある．しかし，アプリケーションを実際に動作させてアプリケーションの挙動を観察する[4][5][6]には膨大な時間がかかり，大量のアプリケーションが公開される中，すべてのアプリケーションの挙動を長時間かけて観察することは困難であると考えられる．よって，アプリケーションの挙動観察にかかる時間的負担を軽減することが重要であると考えられる．

Android OS は OS の中心部に Linux カーネルを採用しており，この Linux カーネルの改変によりシステム内の時刻管理などの挙動を修正することが可能である．よって，特殊なハードウェアなどを用いることなく端末内の時間が実時間よりも速く進む加速環境を構築可能であると期待できる．

本研究では， Android OS およびその一部である Linux カーネルを改変し，アプリケーションの動作を実時間よりも短い時間で観察できる加速実験を行える環境の構築を目的とする．本稿ではその研究の初期段階として，観測対象となるアプリケーションが時間加速環境に対する対抗措置をとらないこと，アプリケーションが端末外の情報により行動を決めることがないことを前提とし，本前提下における Android OS の時間加速環境の構築手法の提案, 実装の紹介，観察結果による評価を行う．

## 2. Android

### 2.1 アプリケーションの配布と動作の観察

Android OS が普及し多数のアプリケーションが開発されている．そして，OS 開発元によるアプリケーション配布サイト[7]や端末を販売しているキャリアのアプリケーション配布サイト[8][9]などで多数のアプリケーションが配布されている．前述の様にアプリケーションの検証は重要なサービスの一つとなっており，アプリケーションの審査はこれらアプリケーション配布サイトなどで広く行われている．

しかし，アプリケーション配布サイトにおける審査の実施にもかかわらず不適切なアプリケーションが配布されてしまった事例[11]や，近年のスマートフォン向けマルウェアの増加[12]をうけ，アプリケーションの動作検証の研究も活発に行われている[13][14][15].

アプリケーションの振る舞いの検証方法としては，アプリケーションの実行を伴わない静的な手法[13][14]と，実行を伴う動的な手法[4][5][6][15]がある．前者は計算資源が十分にある状況においては短い時間で大量の検証を行うことができると期待できるが，実際の動作を見ていないため，

†1 工学院大学
Kogakuin University
†2 工学院大学大学院
Kogakuin University Graduate School
†3 お茶の水女子大学
Ochanomizu University

Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

検証の正確さには限界があると予想できる．後者は，実際の動作を観察するためより高い程度の検証が可能であると期待できるが，検証には長い時間を要すると予想できる．

また，アプリケーションによる端末消費電力の増加の調査に関する既存の研究[4][5][6]においても，アプリケーションを動作させて検証し WakeLock や Alarm のセットを観察することにより消費電力増加の大きいアプリケーションを低くない精度で推定できることが確認されているが，実際にアプリケーションを動作させての検証には長い時間を要し，その高速化は重大な課題になっている．

### 2.2 Android OS における時刻管理

Android OS は，カーネルに Linux カーネルを採用しており，システム内の時刻の管理は同カーネルによって行われている．このため，Linux カーネルを改変することにより Android OS およびそのシステム内で動作するアプリケーションに対する時刻管理の動作を修正できると考えられる．

Linux カーネルでは，xtime と呼ばれる変数に時刻が格納されている．xtime は OS 起動時にハードウェア時刻が読み込まれることにより初期化され，OS が起動している間は原則として tick(Linux カーネルの周期的な処理間隔)ごとに更新される．tick 単位は Linux カーネルのパラメータの一つであり，コンパイル時に指定することができる．多くの Android OS では tick 単位は 10m 秒(100Hz)とされており，この場合は毎秒 100 回の割り込みが発生し，そのたびに xtime が更新される．

xtime 更新時にはハードウェアから供給されるクロックソースが参照され，これをもとに xtime が更新される．クロックソースには複数の種類があり，今回使用した環境では gp_timer もしくは dg_timer が用いられる．

## 3. 提案手法

### 3.1 システム内時間の加速

本章にて，Android OS のカーネルの時間管理実装を改変し，アプリケーション観察時間を短縮する手法を提案する．

Android OS を含む Linux カーネルを用いる OS では，カーネルがアプリケーションプロセスなどに与える時刻情報を制御することにより，OS 上で動作するアプリケーションやシステムプロセスが認識する時間の流れを高速化し，アプリケーションの観察を実時間上よりも短い時間で実現できると期待できる．

本提案手法では Linux カーネルが与える時刻情報が進む速度を高速化し，アプリケーションが認識する時間が進む速度を高速化する．

### 3.2 実装

Linux カーネル 3.4.0 を用いて実装を行った．Linux カーネルでは，時刻は複数の変数を用いて管理されている．本稿の実装では xtime に格納されている時刻を制御し，Android OS のユーザ空間で動作するアプリケーションの時間の認識を制御する．xtime はカーネルソースファイル time/timekeeping.c の関数 timekeeping_get_ns にて更新が行われ，本稿の実装では同関数を修正することにより加速環境を実装した．ソースコードの改変の詳細は付録 A に示す．

同関数の中では cycle_t 型の変数 cycle_now と cycle_delta が宣言されている．cycle_now にはクロックソースから取得した値が格納されており，この値を元にして cycle_delta にクロックソースの増分が格納される．時刻の変化を表す cycle_delta の値を拡大し，端末内の時間の流れを加速する．

## 4. 評価

提案手法を Android OS 上に実装し，実端末で動作させ評価を行った．

### 4.1 実験環境

測定に使用した端末は表 1 のとおりである．OS には，加速機能を有効にすると端末内の時刻が実時間比 2 倍で進むように改変を加えてある．

表 1　使用端末

| | |
|---|---|
| Device name | Nexus 7　(2013) |
| OS | Android 5.0.1 (時間加速可能改変済み) |
| CPU | Qualcomm Snapdragon S4 Pro,1.5 GHz |
| Memory | 2GB |

### 4.2 基礎動作評価 (時間指定型スレッド停止)

本節にて，提案手法による実装の基礎動作の検証を行う．まず，提案手法を導入した Android OS の端末内時間加速機能を有効にした状態で Android アプリケーション内にてスレッドの sleep を行ったときの，sleep メソッドの引数に指定した値(時間)と実際にスレッドが停止した実時間の関係の調査結果について述べる．実際にスレッドが停止した実時間は，Android 端末内で得られる時刻を用いず端末外の時間を人手で計測した．Thread.sleep の引数の時間は 120 秒，240 秒，360 秒とし，測定はそれぞれ 3 回ずつ行った．

測定結果を表 2 に示す．表の「端末外時間」は端末外で計測した実際の時間（以下「実時間」と記す）であり，3 回の計測の平均値である．表より，端末内の時間は実時間の 2 倍の速度で進んでいることがわかる．

Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

表 2　基礎動作評価結果

| Thread.sleep 指定時間 [秒] | 端末外時間 [秒] |
|---|---|
| 120 | 60 |
| 240 | 120 |
| 360 | 180 |

### 4.3 基礎動作評価 (時刻指定型起動)

本節では，将来の時刻を指定し処理を実行させる仕組み(Alarm)に対して加速機能が有効であるかを検証する．

評価のために，現在時刻の 90 秒後に自身を起動するように Alarm をセットし，Alarm より起動されたときに再度 90 秒後に Alarm をセットする動作を繰り返すアプリケーションを動作させた．評価実験は端末内時計で 30 分間行い，その間はスクリーン点灯状態かつ無操作状態とした．

端末内時間にて 30 分の間に実行された Alarm セットの回数，WakeLock の回数および送信された SYN パケット送出回数を表 3 に，それぞれの発生間隔およびその分布を図 1～図 9 に示す．第 2 章で述べたように，既存研究にて WakeLock の実行の観察と Alarm の設定の観察が重要であることが分かっているため，これらの回数を調査した．また，アプリケーションが外部サーバとの通信を開始する回数を調査するために，TCP の SYN パケットの送出回数を調査した．図 1 は，Alarm 設定の間隔(ある Alarm の設定の時刻と，次の Alarm の設定の時刻の差)の頻度分布を表しており，横軸は 1 の桁で四捨五入した設定間隔，縦軸はその間隔が発生した回数を表している．例えば，通常状態，加速状態ともに，発生間隔が約 90 秒(85 秒以上 95 秒未満)であった事例が 18 回であったことが分かる．横軸 100+は，横軸 100 秒(95～105)より大きな値を意味しており，具体的には 105 秒以上を意味している．図 4，図 7 は WakeLock 実行の間隔，SYN パケット送出を表しており，横軸は同様に 1 の桁で四捨五入した発生間隔である．

表 3 より，Alarm のセットの回数，WakeLock の実行の回数，SYN パケットの送出の回数に関して，通常状態と加速状態で大きな差が無いことが分かる．よって，これらの発生回数の観察に関しては，加速機能を用いることで短い実時間で本来と同等の観察を実現できていることが分かる．また，図 1，図 7 より，Alarm のセットおよび SYN パケットの送出に関しては発生間隔に関しても通常状態と加速状態でほぼ差が無く，図 3 と図 4 は類似しており，図 8 と図 9 も類似していることがわかる．よって，Alarm のセットおよび SYN パケットの送出に関しては生間隔に関しても加速状態でほぼ正確に観察できたことがわかる．

図 4 を見ると，WakeLock の発生間隔の頻度分布に大きな差は無く，通常状態，加速状態ともに横軸 0 秒から 30 秒に発生が集中していることが分かる．ただし，各間隔の頻度に関しては差異が見られる．図 5 と図 6 を比較すると加速を行うことにより WakeLock 実行間隔にばらつきが生じることが分かり，WakeLock 実行間隔の正確な再現が要求される観察には適さないことが分かる．

表 3　Alarm set，WakeLock，SYN パケット送信回数の比較

| | 通常状態 [回] | 加速状態 [回] |
|---|---|---|
| Alarm set | 20 | 20 |
| WakeLock | 166 | 160 |
| SYN パケット送出 | 24 | 22 |

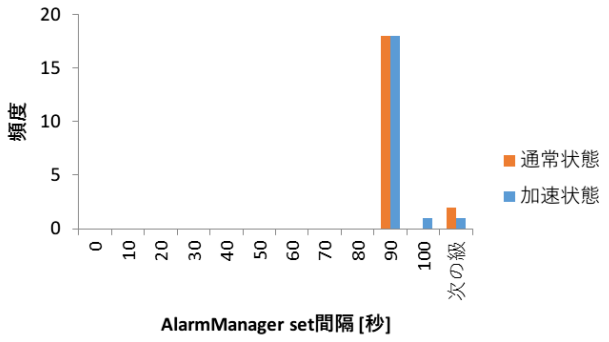


図 1　AlarmManager set 間隔の頻度分布

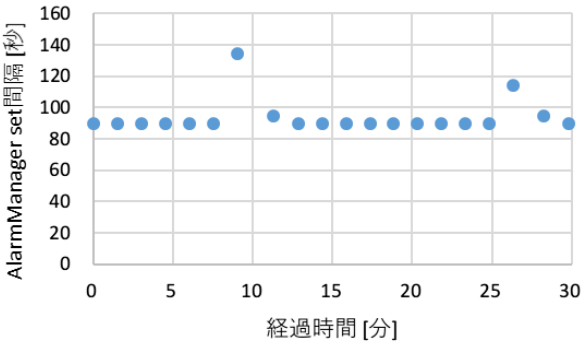


図 2　AlarmManager set 間隔（通常状態）

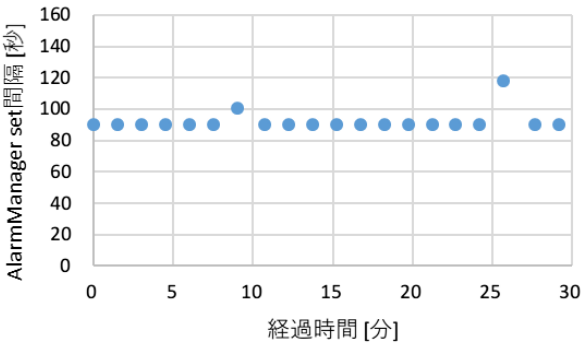


図 3　AlarmManager set 間隔（加速状態）

Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

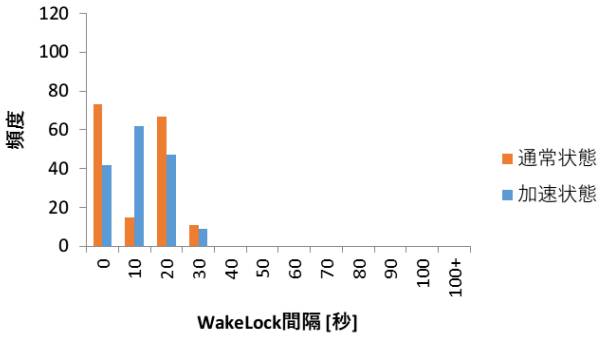


図 4　WakeLock 間隔の頻度分布

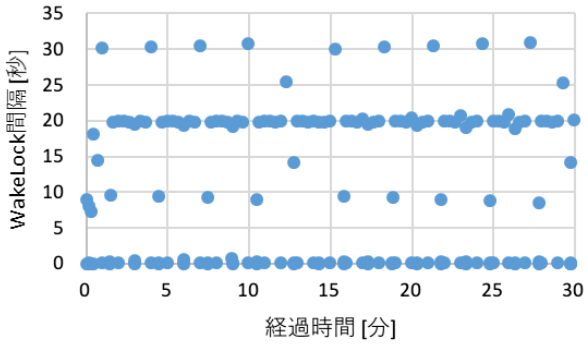


図 5　WakeLock 間隔（通常状態）

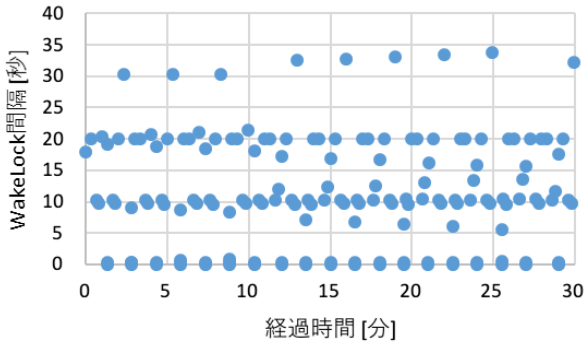


図 6　WakeLock 間隔（加速状態）

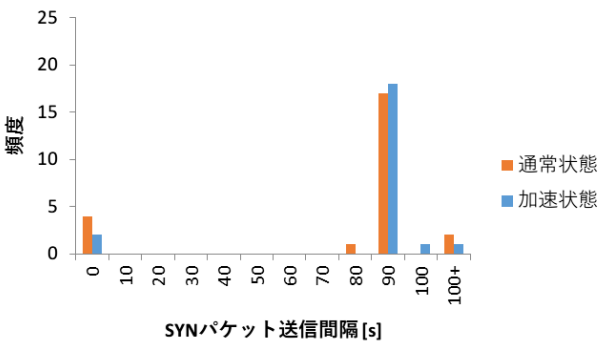


図 7　SYN パケット送信間隔の頻度分布

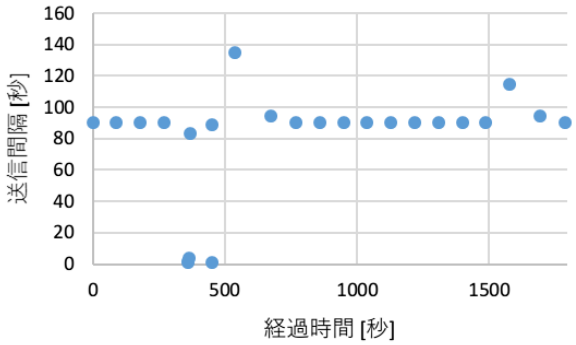


図 8　SYN パケット送信間隔（通常状態）

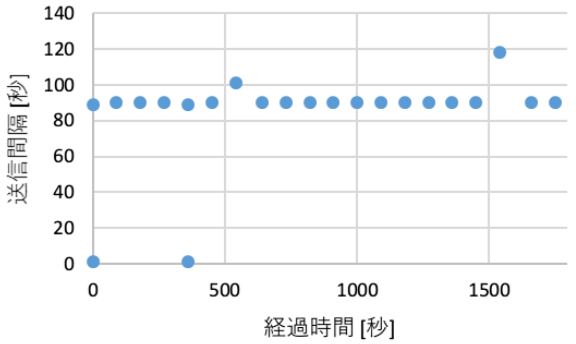


図 9　SYN パケット送信間隔（加速状態）

### 4.4 実アプリケーションを用いた評価 1

4.2 節および 4.3 節では，評価用のアプリケーションを用いて評価を行った．本節では，実アプリケーションを用いての評価を行う．

Google Play ストアで公開されているアプリケーションを提案手法適用済みの環境下で動作させて実験を行った．使用したアプリケーションは，2015 年 10 月 24 日における Google Play ストアのウィジェットカテゴリランキングの上位 10 件である．通常状態の端末と提案手法適用状態の端末にこれらのアプリケーションをインストールし，4.3 節と同様の実験を行った．

測定結果を表 4 および図 10～図 12 に示す．表 4 より，加速状態であっても通常状態とほぼ同数の発生回数（Alarm セット回数，WakeLock 回数，SYN パケット送出回数）が得られ，提案手法がこれらの実アプリケーションに対してもおおむね有効であることが分かる．図 10～図 12 に関しても，加速状態と通常状態で発生間隔に大きな違いは無く提案手法が有効であったことが確認できる．小さな差異は生じているが，通常状態であっても一般のアプリケーションの動作の再現性は必ずしも高くなく，多くの観察目的に関して本手法は有効であると考える．

Alarm セット間隔，WakeLock 実行間隔，SYN パケット送出間隔は付録 B に記す．

Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

表 4　Alarm set，WakeLock，SYN パケット送信回数の比較

| | 通常状態 [回] | 加速状態 [回] |
|---|---|---|
| Alarm set | 10 | 10 |
| WakeLock | 192 | 164 |
| SYN パケット送出 | 16 | 19 |

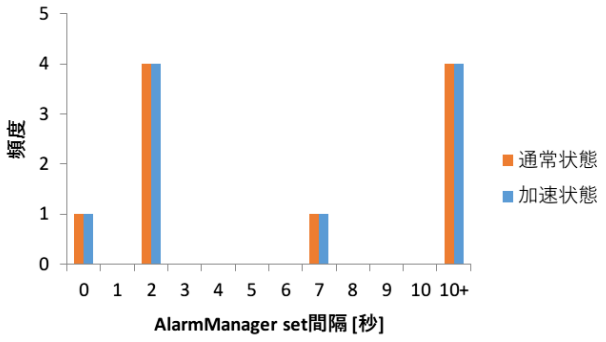


図 10　AlarmManager set 間隔の頻度分布

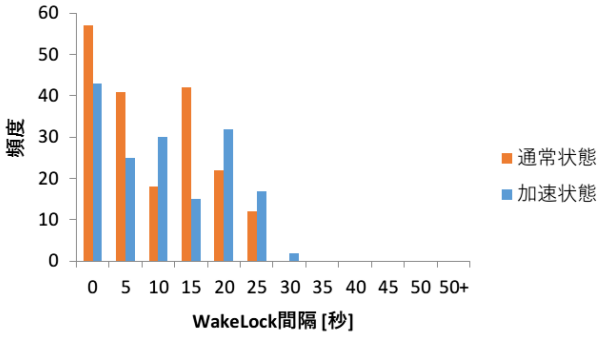


図 11　WakeLock 間隔の頻度分布

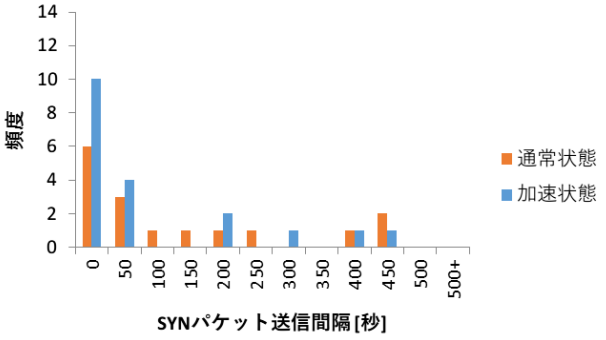


図 12　SYN パケット送信間隔の頻度分布

### 4.5 実アプリケーションを用いた評価 2

前節で用いたウィジェットランキングの上位 10 件のアプリケーションに加え，Alarm セットや WakeLock を高頻度で行うことが確認されている 2 件の実アプリケーションをインストールし，実アプリケーションが 12 件の状態で同一の実験を行った．測定結果を表 5 および図 13～図 15 に示す．

表 5 では，前節の実験結果(図 3)を大幅に上回る Alarm セットと WakeLock 実行の回数が確認されているが，通常状態における回数と加速状態における回数に大きな差異はなく，提案手法は本 2 アプリケーションに関しても適切に加速を可能であることが確認された．

Alarm セット間隔，WakeLock 実行間隔，SYN パケット送出間隔は付録 C に記す．

表 5　Alarm set，WakeLock，SYN パケット送信回数の比較

| | 通常状態 [回] | 加速状態 [回] |
|---|---|---|
| Alarm set | 613 | 613 |
| WakeLock | 812 | 781 |
| SYN パケット送信 | 50 | 46 |

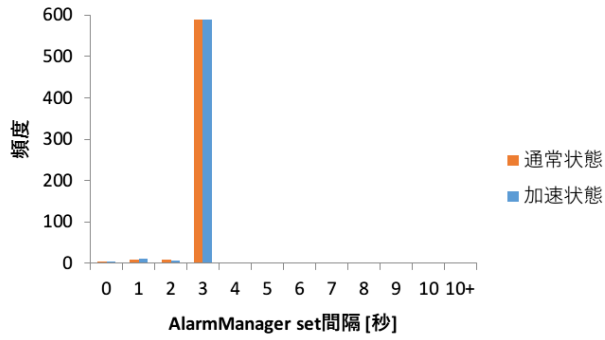


図 13　AlarmManager set 間隔の頻度分布

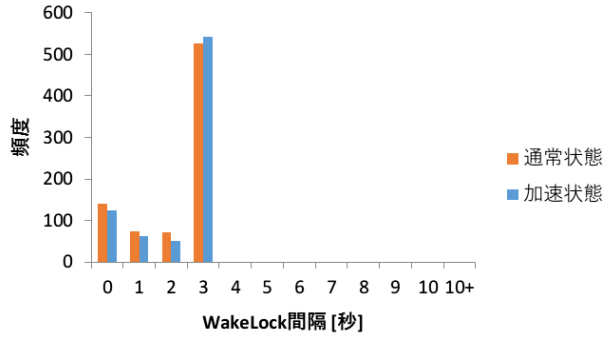


図 14　WakeLock 間隔の頻度分布

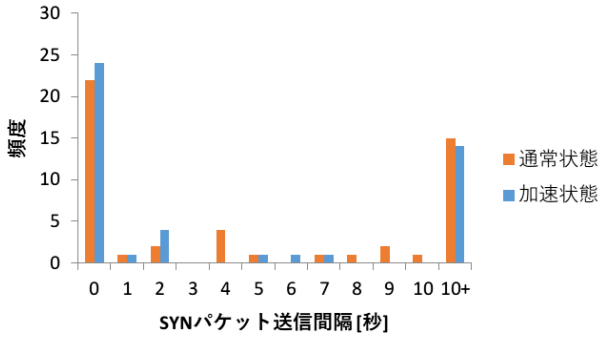


図 15　SYN パケット送信間隔の頻度分布

## 5. おわりに

本稿では，スマートフォンアプリケーションの動作の観察の重要性と，観察時間の長さに起因するその困難さを紹介した．そして，Android OS のカーネル(Linux カーネル)の時間管理実装を改変し，端末内部で動作するアプリケーションが認識する時間の流れを実時間より速くし，短い時間でのアプリケーション観察を可能とする手法を提案した．

提案手法を実装しベンチマークアプリケーションと実アプリケーションを用いて評価したところ，今回使用したアプリケーションのアラームセットや WakeLock 実行の観察に関しては加速された環境でも通常環境と同等の観察結果を得ることができ，提案手法が有効に機能することが確認された．

今後はより多くのアプリケーションを用いての評価，提案手法の適用可能範囲の調査を行っていく予定である．

**謝辞**　本研究は JSPS 科研費 25280022, 26730040, 15H02696 の助成を受けたものである．

## 参考文献

1)　IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012, http://www.idc.com/prodserv/smartphone-os-market-share.jsp
2)　Number of Google Play Store apps 2015 Statistic, http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
3)　Android and Security - Official Google Mobile Blog, http://googlemobile.blogspot.jp/2012/02/android-and-security.html
4)　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.
5)　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．
6)　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
7)　Google Play Store, https://play.google.com/store/apps
8)　NTT ドコモ　d マーケット　アプリ＆レビュー, http://app.dcm-gate.com/
9)　au Market, http://market.kddi.com/update_info/
10)　Creating Better User Experiences on Google Play | Android Developers Blog, http://android-developers.blogspot.ca/2015/03/creating-better-user-experiences-on.html
11)　Porn clicker keeps infecting apps on Google Play, http://www.welivesecurity.com/2015/07/23/porn-clicker-keeps-infecting-apps-on-google-play/
12)　McAFEE LABS THREATS REPORT June 2014, http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2014.pdf
13)　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
14)　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
15)　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

## 付録

### A.　加速環境のソースコード

time/timekeeping.c における加速環境の実装は図 16 のとおりである．

```
unsigned long long int accel_cycle_last=0; //過去の cycle_now（改変前）
unsigned long long int accel_cycle_now_delta=0; //cycle_now の増分
unsigned long long int accel_cycle_mod_last=0; //過去の cycle_now（改変後）
static inline s64 timekeeping_get_ns(void)
{
        cycle_t cycle_now, cycle_delta;
        struct clocksource *clock;

        /* read clocksource: */
        clock = timekeeper.clock;
        cycle_now = clock->read(clock);

        accel_cycle_now_delta = cycle_now - accel_cycle_last;
        accel_cycle_last = cycle_now; //改変前の cycle_now 記録
        if(accel_enabled){
              cycle_now = accel_cycle_mod_last + accel_cycle_now_delta*2;
        }else{
              cycle_now = accel_cycle_mod_last + accel_cycle_now_delta;
        }
        accel_cycle_mod_last = cycle_now; //改変後の cycle_now 記録

        /* calculate the delta since the last update_wall_time: */
        cycle_delta = (cycle_now - clock->cycle_last) & clock->mask;

        /* return delta convert to nanoseconds using ntp adjusted mult. */
        return clocksource_cyc2ns(cycle_delta, timekeeper.mult,
                                  timekeeper.shift);
}
```

図 16　ソースコードの改変

Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

timekeeping_get_ns 関数の呼び出し $n$ 回目におけるクロックソースから得られた（修正が加えられていない）cycle_now の値を $cn_n$ とすると，$cn_n$ - $cn_{n-1}$ により修正が加えられていない状態の cycle_now の増分を計算することができる．上のコードにおけるクロックソースから取得した直後の cycle_now が $cn_n$ であり，accel_cycle_last が $cn_{n-1}$ である．そして，$cn_n$ - $cn_{n-1}$ を変数 accel_cycle_delta に格納している．

accel_enalbed が加速の有効/無効を表しており，加速有効状態であれば cycle_now に $cn_{n-1}$ + ($cn_{n-1}$ - $cn_n$)*加速率を上書きし，加速無効状態であれば $cn_{n-1}$ + ($cn_{n-1}$ - $cn_n$)を上書きすることで，加速状態と非加速状態の時刻を管理している．ただし，図のソースコードでは加速率は 2 となっている．

### B. 実アプリケーションを用いた評価 1 の結果

実アプリケーションを用いた評価 1 における Alarm セット間隔，WakeLock 実行間隔，SYN パケット送出間隔は図 17～図 22 の通りである．

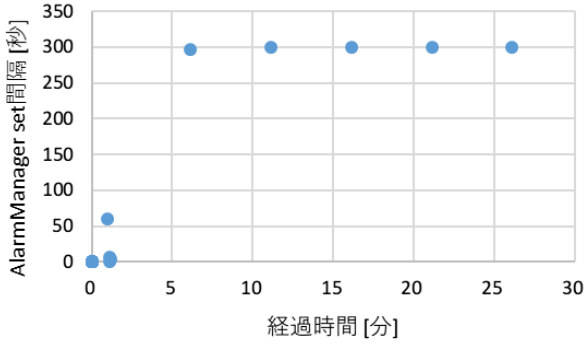


図 17　AlarmManager set 間隔（通常状態）

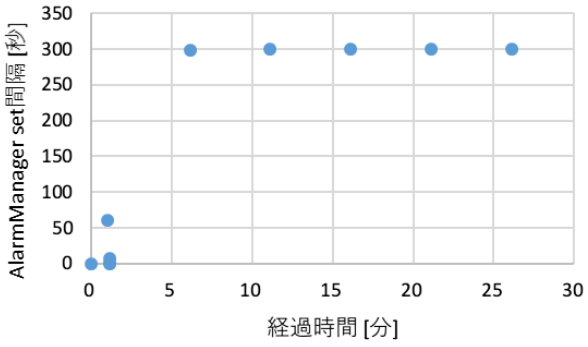


図 18　AlarmManager set 間隔（加速状態）

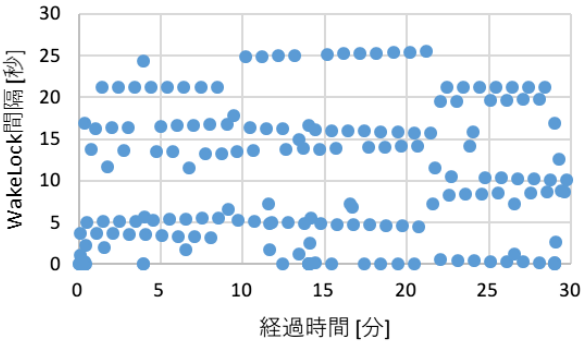


図 19　WakeLock 間隔（通常状態）

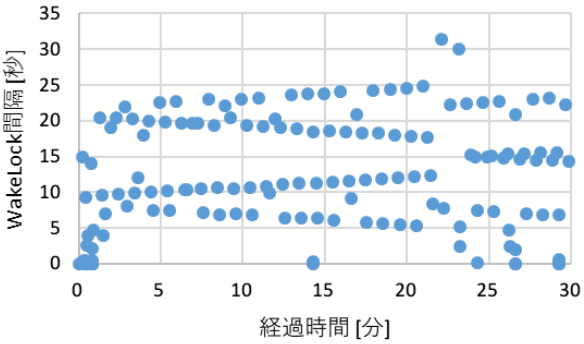


図 20　WakeLock 間隔（加速状態）

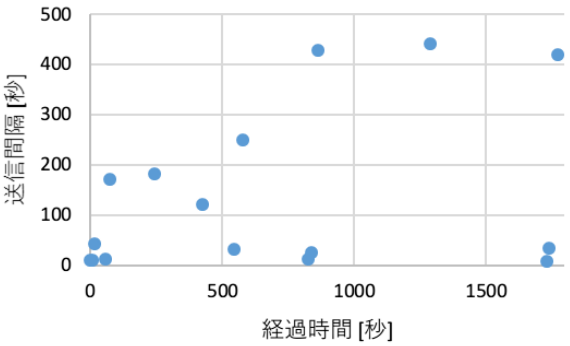


図 21　SYN パケット送信間隔（通常状態）

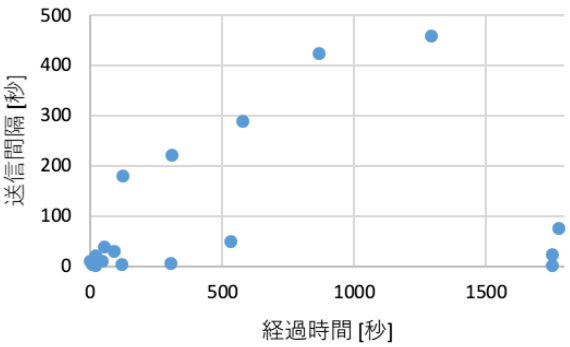


図 22　SYN パケット送信間隔（加速状態）

Vol.2016-GN-97 No.27
Vol.2016-CDS-15 No.27
Vol.2016-DCC-12 No.27
2016/1/22

### C. 実アプリケーションを用いた評価 2 の結果

実アプリケーションを用いた評価 2 における Alarm セット間隔，WakeLock 実行間隔，SYN パケット送出間隔は図 17～図 22 の通りである．

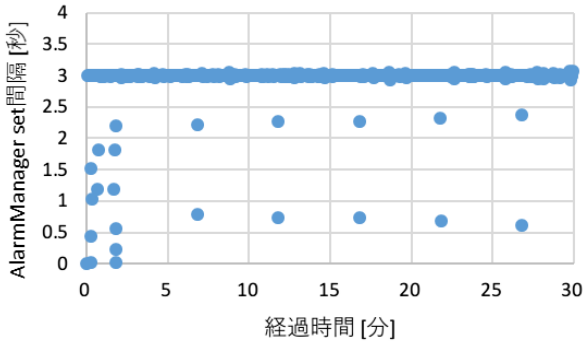


図 23　AlarmManager set 間隔（通常状態）

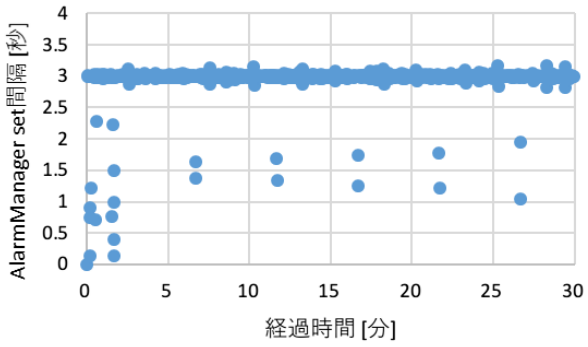


図 24　AlarmManager set 間隔（加速状態）

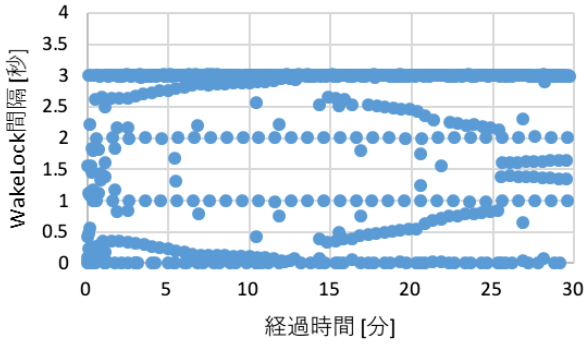


図 25　WakeLock 間隔（通常状態）

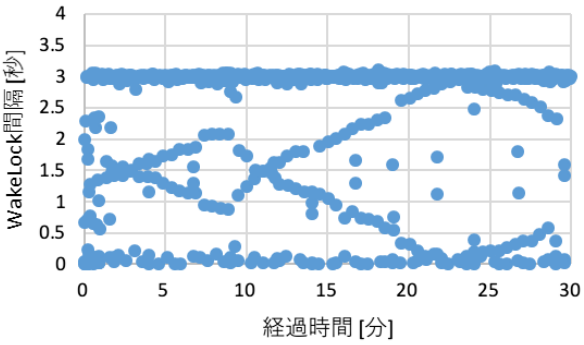


図 26　WakeLock 間隔（加速状態）

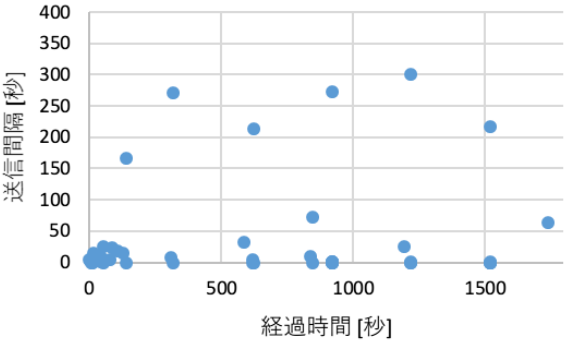


図 27　SYN パケット送信間隔（通常状態）

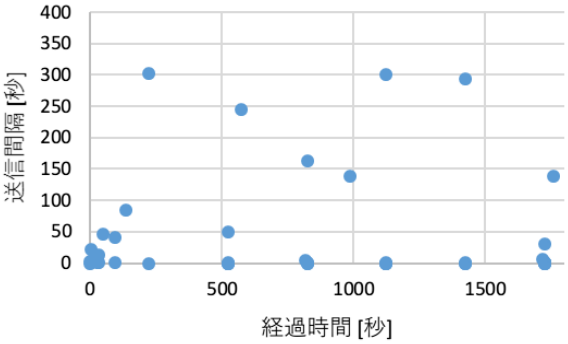


図 28　SYN パケット送信間隔（加速状態）