Vol.2019-MBL-92 No.1
Vol.2019-CDS-26 No.1
2019/8/29

Android 端末内部時間の加速時におけるパケット通信エラーに関する考察

小野里亮祐[1]　神山剛[2]　福田晃[2]　小口正人[3]　山口実靖[1]

概要：Android OS は主要なスマートフォン OS となり，Android OS 用のアプリケーションの開発や動作の検証は重要な事項となっている．Android アプリケーションを実際に動作させて検証する動的検証は，ソースコードや実行バイナリの解析に基づく静的検証よりも現実的な検証を行うことができる．しかし，動的動作検証には膨大な時間がかかるとの欠点も指摘されており，この検証時間の短縮は重要な課題となっている．アプリケーションの動的動作検証の時間を短縮する手法として，Android OS の Linux カーネルにおける時間管理実装を改変し，システム内で認識される時間の流れを速くする手法が提案されており，一部の環境にてその有効性が確認されている．ただし，Wi-Fi を用いた通信の動作については十分な検証がなされていない．本稿では，時間加速 Android の加速時における Wi-Fi 通信の性能(スループットと TCP エラー数)に着目し，この評価を行う．そして，スループット評価により加速時であって概ね安定的な通信速度を実現できることを示す．また，エラー数評価にて時間の流れを加速すると TCP エラーの数が増加してしまうことと，RTO(Retransmission Timeout)の拡大により TCP エラー数を削減できることを示す．

キーワード：Android，TCP，TCP 再送処理，RTO (Retransmission Timeout)，Linux カーネル，加速アプリケーション動作観察環境

1. はじめに

近年，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．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り，その重要性は年々増加している．

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，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一週間の振る舞いを観察するには観察時間として一週間が必要となる．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．

我々はこの課題に対して過去に，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する手法を提案した[7][8]．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[8]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[9]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た．また，本手法を 100 倍などの高い加速倍率で適用した環境における考察として，加速倍率と観察精度の関係の評価[10]や，高加速倍率環境におけるシステム安定性の評価を行い，高い加速倍率においてはシステムの安定性の低下が生じることを確示した．そして，システム案手性の低下が生じる原因の調査や，安定性向上手法の提案[10][11]，及び評価を行っている．

本稿では，加速 Android 環境下での加速倍率と，Wi-Fi パケット通信時における TCP エラー数と，スループットの関係についての評価を行う．

2. 関連研究

2.1 Android における時刻管理

本節にて，文献[13]にて述べた Android における時刻管理手法の解説を引用し紹介する．

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る．文献[13]の環境の例では 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

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

Vol.2019-MBL-92 No.1
Vol.2019-CDS-26 No.1
2019/8/29

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．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．

2.2 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[13][14][15]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．本節では，これら文献における発券手法の解説を引用し紹介する．

文献[14][15]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．よって，効率的に消費電力の大きいアプリケーションの推定を行うためには，実時間よりも速い速度で時間が流れる加速実験環境の実現が重要であるといえる．

2.3 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ている[7]．本節にて，文献[13]における解説を引用して紹介する．

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[17][18]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[19]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．

2.4 加速 Android 環境におけるシステム安定性の改善

文献[10][11]にて，加速 Android 環境におけるシステム安定性の改善手法が提案されている．

同文献では，高倍率加速環境において端末内時間で 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，高倍率加速環境では安定性が低下することが示されている．また，Activity Manager によるプロセスキルの無効化，ウォッチドッグタイマのタイムアウト時間延長，システムプロセスキルの無効化による安定性改善手法が提案され，同手法を用いる事で実際に安定性が改善されていることが示されている．

2.5 時間加速 Android 環境のシステム安定性のアプリケーションによる評価

文献[20]にて，我々は加速 Android 環境におけるシステム安定性の改善手法に対する実アプリケーションによる評価を行った．

同文献では 2.1 節のシステム安定性改善手法を用いて実アプリケーションで性能評価を行っており，結果としてシステム安定性改善手法は加速状態における実アプリケーションの動作においても有効であるということが示されている．

2.6 加速 Android における Wi-Fi 通信の TCP エラーの削減

我々は加速 Android 環境の Wi-Fi 通信時における TCP エラー数を調査し加速 Android 環境において加速倍率を増加させる毎に TCP エラー数が増えることを示した[12]. また，この TCP エラーの多くは再送タイムアウトによるものであると考え，Linux カーネル内の RTO（Retransmission Time-Out）値の増加速度を早め，また RTO 値の初期値を増加させることでタイムアウト回数を減少させ，TCP エラー数を減少させる手法を提案した．

エラー数削減手法は，以下の通りである．Linux カーネル内の include/net/tcp.h を修正し，TCP の Retransmit Time-Out を増加させる．具体的には，図 1 の様に RTO の初期値と，1 回の RTO 値推定処理毎に増える RTO 値の増加量を増加させる．コメントされている行が修正前のコードである．RTO の増加率は加速倍率に等しく，ExRatio は加速倍率である．

当該研究では，TCP エラー数の評価は行われているが，スループットの評価は行われておらず，本稿の様な総合的な評価は行われていない．また，エラー数の評価に関しても，非加速時の比較が行われおらず，加速による TCP エラー数が増加するか否についての考察はなされていない．

```
[include/net/tcp.h]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#define TCP_TIMEOUT_INIT ((unsigned)(1*HZ*ExRatio))
// #define TCP_TIMEOUT_INIT ((unsigned)(1*HZ))

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
static inline u32 __tcp_set_rto(const struct tcp_sock *tp)
{
        return ((tp->srtt >> 3) + tp->rttvar)*ExRatio;
        // return (tp->srtt >> 3) + tp->rttvar;
}
```

図 1. RTO 増加による TCP エラー削減手法[12]

3. 加速環境における Wi-Fi 通信性能の評価

本章にて, 加速 Android 端末における TCP 通信の性能(スループットとエラー数)の評価を行う.

3.1 スループットの評価

計測には，クライアントからサーバへ 1MB のデータを 10000 回送信するアプリケーションを作成し，これを動作させてスループットを測定した．計測環境は表 1 の通りである．受信したデータ量はサーバ側で tcpdump コマンドを使用して測定した．各加速倍率につき計測は 5 回行った．

なお、クライアント端末からルータへは無線 LAN IEEE 802.11n で接続し、ルータからサーバへは有線 LAN Gigabit Ethernet で接続している。クライアントからサーバへは LAN にて接続されている状況であり, RTT は小さい状況となっている.

通信スループットとは，単位時間あたりの転送ビット数であるが，本稿では単位端末内示時間あたりではなく単位実時間当たりの転送ビット数をスループットとする．例えば，加速倍率が 10 の場合，実時間 2 秒の間に端末内時間が 20 秒進む．この間に 800 ビットの転送が行われた場合は，スループットは 800 ビット/2 秒=400 ビット/秒とし, 800 ビット/20 秒=40 ビット/秒とはしない.

表 1.計測環境

Device（Client）	Nexus 7（2013）
OS（Client）	Android 6.0.1 (AOSP) with modified Linux kernel 3.4.0
Device（Server）	Intel(R) Core(TM) i3-4360 CPU @ 3.70GHz RAM：8GB
OS（Server）	Ubuntu 18.04.2 LTS

図 2 に加速倍率 10 倍以下における通常時と RTO 拡張時のスループット測定結果を示す．また，図 3 に通常時における加速倍率 10 倍以上におけるスループットを示す.

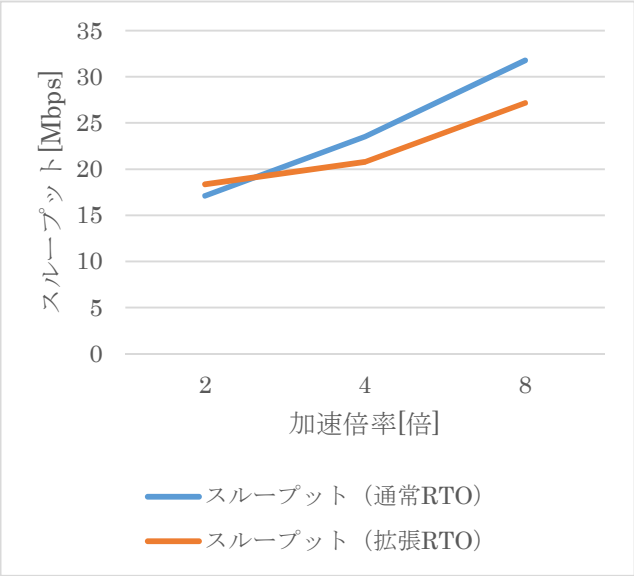

図 2. 通常時と RTO 拡張時のスループット測定結果

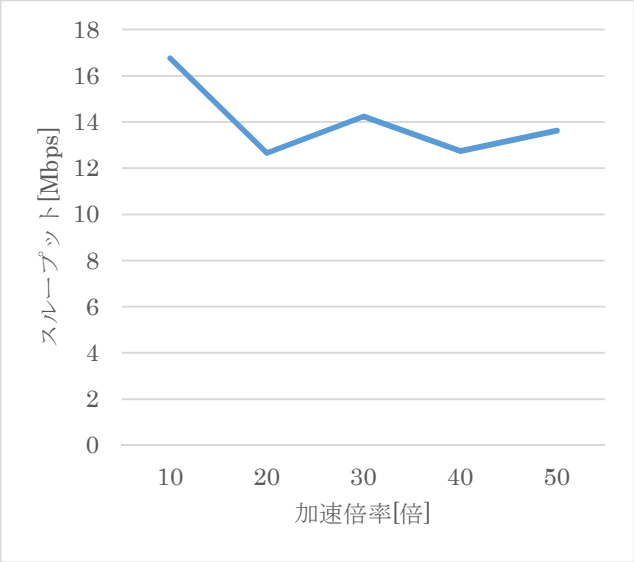

図 3. 通常時の加速倍率 10 倍以上におけるスループット

図より，加速倍率を増加させることにより，スループットの変化することが確認される．ただし，大きな変化は生じていないことがわかる．また，RTO 拡大を行った場合と行わない場合を比較すると，スループットに関しては大きな差が生じていないことが分かる.

3.2 TCP エラー数の評価

本節にて，時間加速 Android における加速倍率と Wi-Fi による TCP 通信における TCP エラー数の関係についての評価を行う．そして，エラー数改善手法[12]における TCP エラー数の削減効果についての評価を行う．非加速時(1 倍時)と加速時(2 倍，5 倍，10 倍)における Wi-Fi 通信時における TCP エラー数を図 4 の「通常 RTO」に示す．図より，非加速環境では TCP エラーが生じないが, Android OS のカーネルの時間管理を加速することにより TCP エラー数が

Vol.2019-MBL-92 No.1
Vol.2019-CDS-26 No.1
2019/8/29

増加することが確認できる．特に，5 倍や 10 倍などの高い倍率の環境にて大幅に増加していることが分かる．

また，図 5 に各倍率における TCP エラー種別ごとの発生回数を示す．図より，発生している TCP エラーの多くは，重複 Ack，再送のエラー，再送であることが分かる．

次に RTO 拡張による TCP エラー削減手法[12]適用時におけるエラー数を同図の「拡張 RTO」に示す．図より，「通常 RTO」と比較して，RTO 拡張手法を適用することにより多くの場合 TCP エラー数を大幅に削減できることが確認できる．特に，5 倍や 10 倍などの高い倍率の環境にて大幅に削減できていることが分かる．

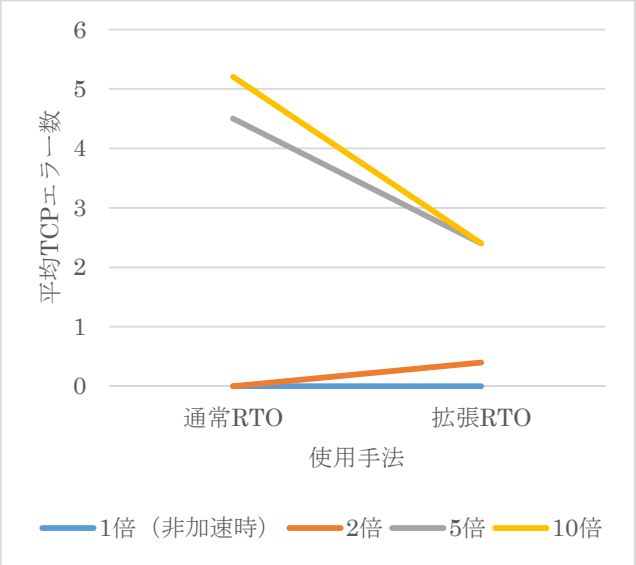

図 4. Wi-Fi 通信における非加速時と加速時における TCP エラー数の比較

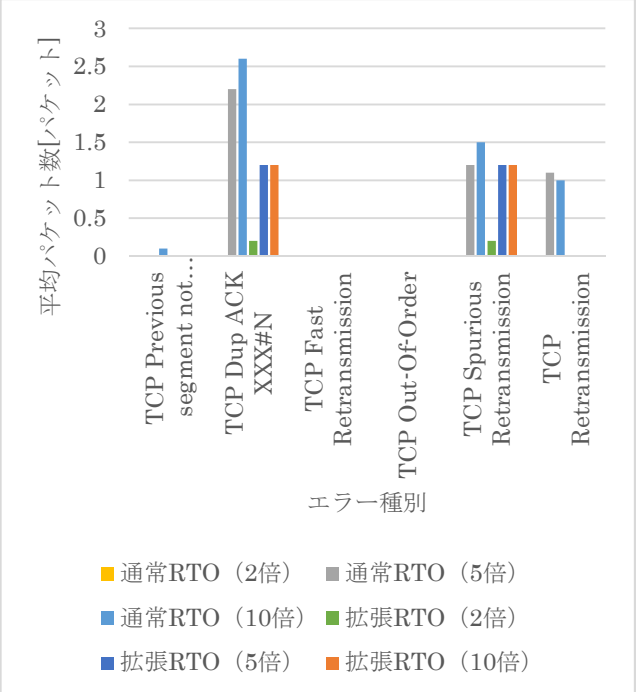

図 5. TCP エラー種別

4. 考察

本章にて，加速カーネルにおける TCP 通信のスループットと TCP エラー数についての考察を行う．TCP は，データパケットを送信した後に所定の再送時間(RTO)が経過するまでに確認応答(Ack)を受信できないとパケットロスが生じたと考え，パケットの再送を行う．RTO は，物理的な物理伝搬遅延時間が大きい環境では大きくする必要があり，小さい環境では小さくする必要がある．よって，伝統的には TCP は通信をしながら常に RTT(Round-trip time，往復遅延時間)を観察し，その観察 RTT をもとに RTO を決定していた．この場合は，カーネルにおける時間の進む速度を上げたとしても確認応答を待つ時間と RTO の両方が同等に拡大され，加速は待つ時間と RTO の大小関係に対して中立になり影響をほとんど与えないと予想される．しかし，近年の TCP 実装は必ずしも観測 RTT に基づいて RTO を定めておらず，時計の加速により確認応答を待つ時間が観測上拡大してしまい，待つ時間が RTO よりも大きくなりやすくなってしまい，タイムアウトの発生が増えることが予想される．

TCP はタイムアウトが発生すると，パケットを再送するとともに，自身の輻輳ウィンドウサイズを大幅に削減させる．よって，本稿の実験の様に RTT の小さい LAN 環境では大きな性能劣化は確認されてなかったが，インターネット通信の様な RTT の大きな環境における通信では加速により大幅な TCP 通信速度の低下が生じる可能性が考えられる．

5. おわりに

本稿では，Android OS のカーネルの時間管理実装の改変による時間加速 Android を紹介した．そして，加速時における Wi-Fi 通信の性能(スループットと TCP エラー数)の評価を行った．そして，加速によりスループットに変化が生じること，スループットの変化は大きくはないこと，加速により TCP エラーが増加すること，既存の TCP エラー削減手法によりエラーを削減できることを示した．

今後は，カーネルの動作の解析による TCP エラー増加の理由の解析，RTT の大きな環境における評価を行う予定である．

謝辞

本研究は JSPS 科研費 15H02696, 17K00109, 18K11277 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] 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の加速環境の構築", 情報処理学会 研究報告コンシューマ・デバイス＆システム（CDS），Vol. 2016-CDS-15，No. 27, pp. 1-8, 2016.
[8] 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
[9] 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
[10] 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
[11] 福田翔貴，栗原駿，神山剛，福田晃，小口正人，山口実靖, “加速 Android 環境におけるシステム安定性の改善” ，情報処理学会 研究報告コンシューマ・デバイス＆システム（CDS）, Vol. 2018-CDS-21, No. 31, pp. 1-6, CDS21-22
[12] Ryosuke Onozato, Akira Fukuda, Takeshi Kamiyama, Masato Oguchi and Saneyasu Yamaguchi: “Reducing TCP Errors in Accelerated Application Test in Android OS”, IEEE INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS – TAIWAN (IEEE 2019 ICCE-TW),2019
[13] 福田 翔貴,栗原 駿,神山 剛,福田 晃,小口 正人,山口 実靖, “加速 Android 環境におけるシステム安定性の改善”, 研究報告コンシューマ・デバイス＆システム（CDS），巻 2018-CDS-21，号 31, pp. 1 – 6, 2018 年 01 月
[14] 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.
[15] 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.
[16] 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
[17] 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
[18] 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.
[19] 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
[20] 小野里亮祐，福田翔貴，神山剛，福田晃，小口正人，山口実靖, "時間加速 Android 環境のシステム安定性のアプリケーションによる評価”，情報処理学会 研究報告コンシューマ・デバイス＆システム（CDS），巻 2018-CDS-22，号 11，pp. 1 – 8，2018 年 5 月