カーネルモニタを用いたAndroid端末の無線LAN通信時の通信性能の考察

三木香央理† 山口 実靖†† 小口 正人†

† お茶の水女子大学 〒112-8610 東京都文京区大塚 2-1-1
†† 工学院大学 〒163-8677 新宿区西新宿 1-24-2
E-mail: †kaori@ogl.is.ocha.ac.jp, ††sane@cc.kogakuin.ac.jp, †††oguchi@computer.org

あらまし 近年, スマートフォン市場の成長に伴い, 携帯端末で動作する組込み機器のソフトウェアプラットホームとして Google 社開発の Android が注目されている. アプリケーション開発や柔軟な拡張性において注目度の高い Android 携帯に対し, 本研究ではそのネットワークコンピューティング能力について評価する. Android の様な組込み機器は, 汎用の PC などとはアーキテクチャが異なるため, インタフェースやリソースの問題から, 通信などの動作時に, 組込み機器の中でどのような事が起こっているのか, 正確に把握することは難しく, その通信動作を解析する事は興味深い. 本研究では, 組込み機器において, カーネルの中の振舞を把握することができるカーネルモニタツールを開発し,Android のトランスポート層においてこれを動作させた. これを用いて, 組込み機器の通信時の内部動作を解析することが可能である事を示し, Android の通信を評価する.

キーワード Android, スマートフォン, 組込み機器, モバイル端末,Linux カーネル

A study about performance of communication on Android terminals in a wireless LAN with Kernel Monitor

Kaiori MIKI†, Saneyasu YAMAGUCHI††, and Masato OGUCHI†

† Ochanomizu University 2-1-1 Otsuka, Bunkyou-ku, Tokyo, 112-8610, JAPAN
†† Kogakuin University 1-24-2 Nishi-shinjuku, Shinjuku-ku, Tokyo,163-8677, Japan
E-mail: †kaori@ogl.is.ocha.ac.jp, ††sane@cc.kogakuin.ac.jp, †††oguchi@computer.org

1. はじめに

近年, １人１台の携帯電話を所有することが当たり前となってきている. また１人で複数台所有することも稀ではなく, サービスや用途によって使い分けているユーザが増加している. 以前は音声通話とメールが主な使われ方であったが, 最近は通信速度が向上し, インターネットや音楽, 動画, ラジオ, テレビ, 非接触型 IC など多機能化されている. 従って, キャリアごとに仕様が違う OS を用いると開発に膨大なコストが掛かるため, 独自の OS では対応しきれなくなっている.

そこで携帯電話向けの汎用 OS が求められてきた. この場合, 基本部分はプラットフォームとして共有化し, 独自の機能やサービスは個別に開発する. これによって開発の効率化が実現できる. またプラットフォームを公開し, オープンソフトウェアにすることで, 対応アプリケーションが作りやすくなり数も増えるというメリットがある. これを実現したものが Google 社により開発された, 携帯端末で動作する組込み機器のソフトウェアプラットホームである Android [1] である.

Android はこれまでの携帯端末用ソフトウェアとは異なり, オープンソースであるためアプリケーション開発における制約がない. また Android 搭載の携帯上ならキャリア, 端末を問わずアプリケーションを実行でき, カスタマイズの自由度が高い. これらの要因から多くのユーザにシェアが広まってきている. この様に Android はアプリケーション開発や柔軟な拡張性において注目度が高い.

最近の Android に関する研究は, 間嶋らの研究 [2] を除いてはほとんどがアプリケーションに関するものである. 本研究ではそのサービスを提供することを可能にしたシステムプラットホームとしての Android に焦点を当て, 特にそのネットワーク能力およびネットワークコンピューティング能力について掘り下げていく.

携帯を含めた組込み機器は, 汎用の PC などとはアーキテク

チャが異なるため, その通信動作を解析する事は興味深い. 特にスマートフォンなどの携帯はクライアント端末として主役になってきており, 様々な通信場面でこれらの端末の動作を解析する事が望まれる. しかし組込み機器の動作については, 入出力のインタフェースが乏しいため, 確認する方法が著しく制限されてしまったり, リソースが汎用 PC などと比べて乏しく, 動作解析のために割けるリソースが不十分で, 動作解析を行うことでシステムの振舞に大きな影響を与えてしまう可能性も考えられる. このような理由から, 通信などの動作時に, 組込み機器の中でどのような事が起こっているのか, 正確に把握することは難しかった.

本研究では, 組込み機器特有の困難な点を克服して,Android 携帯に対し汎用 PC におけるカーネルモニタ [3] と同様のツールを開発し, これをトランスポート層において動作させて, 組込み機器の通信時の内部動作を解析することが可能である事を示す.

2. Android のアーキテクチャ

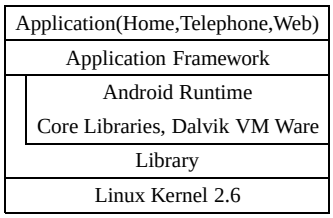

図 1　Android のアーキテクチャ

Android のアーキテクチャを図 1 に示す. Android は Linux2.6 カーネルを用いて構築されており, この OS に各種コンポーネントを追加し Android というプラットフォームを構成している. また,Linux カーネルの上に Android 独自のアプリケーション実行環境である Android Runtime を実装し, Dalvik と呼ばれる独自の仮想マシンを搭載している. これは Java の仮想マシン（JVM）に相当する. その上にアプリケーション・フレームワーク, アプリケーションが乗る形態であるため, アプリケーションは Dalvik にあわせて開発すればよく, ポータビリティが高い.

Android の開発環境はこれまでの携帯端末用開発ツールとは異なり, オープンソースでありキャリア間の制約がないため, カスタマイズ自由度が高く, 他キャリアおよび他機種への柔軟な拡張性があるといえる.

一方, 通信については Linux カーネルの中のプロトコルスタックを用いて行われているため, この TCP 実装部分などで性能が決まってくると考えられる. そのため, 本研究ではカーネル中のトランスポート層実装に焦点を当て評価を行う.

2.1　クロス開発

携帯端末はディスプレイも小さく,CPU 性能やメモリ容量が高くないため, 開発時と実行時に異なるコンピュータ環境を用いるクロス開発の形が取られることが一般的である. Android においても一般にクロス開発が行われ, 主に開発を行うコンピュータがホスト環境で,Android 実機がターゲット環境となる. ホスト環境に通常無いカメラ機器等はホスト内にあるエミュレータを動かすことで実行できるようになっている. Android は組込み機器であるため, 実機で実行できるコマンドにも制限があることから開発環境としては適していない. クロス開発を行うことで開発の効率を上げることができる.

アプリケーション開発だけでなく,Android 自身のビルドなどもクロス開発の形で行われる. 本論文で紹介するカーネルモニタもクロス開発を行って Android 端末に組み込む.

3. カーネルモニタ

この章では, 我々が開発したオリジナルシステムツールであるカーネルモニタとその導入法について説明する.

3.1　カーネルモニタ概要

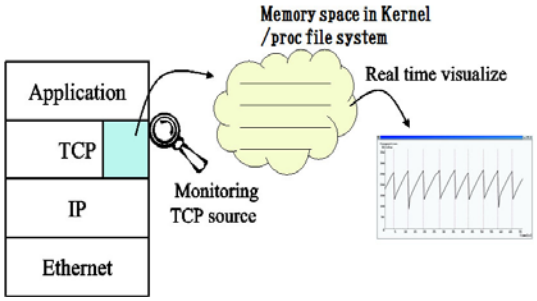

図 2　カーネルモニタの概要

カーネルモニタは通信時に, どの時間にカーネルのコードのどの部分が実行されて, その結果カーネル内部のパラメータの値がどのように変化したかを記録する事ができるツールである. 図 2 に示すように，カーネル内部の TCP ソースにモニタ関数を挿入しカーネルを再コンパイルすることで TCP パラメータをモニタ可能にしている. これによりモニタできるようになった値には，輻輳ウィンドウ，ソケットバッファのキュー長の他，各種エラーイベント (Local device congestion，重複 ACK，SACK 受信，タイムアウト検出) の発生タイミングなどがある．カーネルモニタよって, 正常動作時のカーネルの振舞を知る事ができ, さらには通信において問題が生じている場合に, その問題を特定し何が起こっているのか調べる事も可能となる.

カーネルは通常のアプリケーションとは異なる特殊なソフトウェアであり, 通常のアプリケーションのようなデバッグ手法は使えないため，汎用 PC においても, 通信時の OS のカーネルの中の振舞を知る事は容易ではない. しかし汎用 PC においては, カーネルモニタを用いる事により, この問題を解決することができる [8]. 本研究ではこのカーネルモニタを組み込み機器である Android に応用する.

3.2　Android 端末におけるカーネルモニタの開発

Android は Linux カーネルをベースとしており, その点において汎用 PC と同様のアプローチが行える可能性はあるが,Android 携帯は組込み機器であり汎用の PC と異なる点も多数ある. 例えばメモリやストレージ等のリソース量が制限されているため, 汎用 PC と同じアプローチはリソース不足で困難である可能性がある．動作解析のために割けるリソースが不十分で, 動作解析を行うことでシステムの振舞に大きな影響を与えてしまう可

能性も考えられる. またシステムもアプリケーションも開発はクロスコンパイルを用いる必要があり,OS のビルトは特殊な方法を用いて行い, さらにコンパイルした OS を Android 携帯の実機で立ち上げるためにも特別な手順が必要となる.

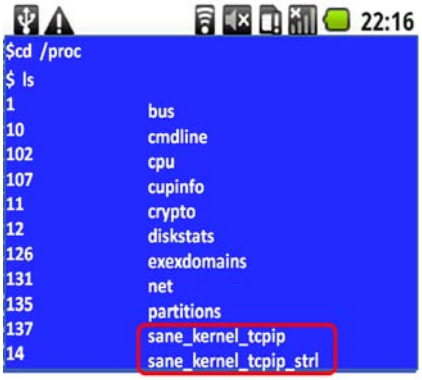

図 3　/proc ファイルシステム

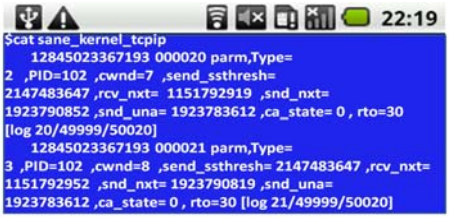

図 4　カーネルモニタのログ

本研究では, 組込み機器特有の先に示した難しい問題を克服し, Android 携帯の実機向けの汎用 PC におけるカーネルモニタと同様のツールを開発した. これをクロスコンパイルにより OS のコードに埋め込み,Android 携帯の実機に送り込んで立ち上げ, 実際にカーネルモニタが動作する事を確認した. そしてカーネルモニタを動作させて,Android の通信時の内部動作を解析することが可能である事を示す. 図 3 , 図 4 はカーネルモニタを動作させたときの Android 端末のキャプチャ画面である. 動作させた結果, 汎用 PC のカーネルモニタとほぼ同様な使い勝手で Android のカーネル内部の振舞を解析することが可能であるとわかり, そのようなツールを実現することができたといえる.

4.　実験システムと基本性能測定

本章では本実験で使用した測定ツール, 実験環境および実験手順を示す. Android 端末としては, HTC 社製の携帯電話（スマートフォン）を用いた.

表 1　Experimental Environment

Android	Model number	AOSP on Sapphire(US)
	Firmware version	2.1-update1
	Baseband version	62.50S.20.17H_2.22.19.26I
	Kernel version	2.6.29-00481-ga8089eb-dirty
	Build number	aosp_sapphire_us-eng 2.1-update1 ERE27
server	OS	Fedora release 10 (Cambridge)
	CPU	CPU : Intel(R) Pentium(R) 4 CPU 3.00GHz
	Main Memory	1GB

表 1 に本研究の実験環境を示す. 本研究では, スループット測定のために iperf-2.0.4 [4] をクロスコンパイルし,Android 端末に送り込んでソケット通信の性能を測定した. クロスコンパイラとしては arm-2008q3 [5] を使用した.

4.1　Android 間通信

図 5　Android 間通信スループット

まず, 図 5 に示すように Android 端末を IEEE802.11g 無線 LAN 機能を用いて AP を経由で通信した場合の基本性能を測定した. 図 6 に示すように Android 間 TCP 通信の平均スループットは 8.18(Mbps),UDP 通信においては 6.7(Mbps) となった. Android 実機の場合,UDP の方が性能が低いことが確認された. パケットロスは 0 である, x 86 型搭載 PC 上で動作する Android, 通称 Android-x86 で同様の実験を行うと,UDP 通信の方が性能が良いことが確認されており, これは Android 携帯電話実機の特性と考えられる [6].

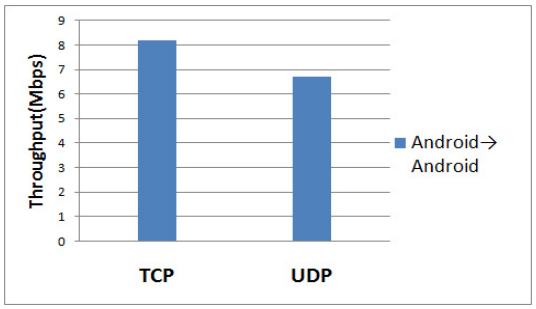

図 6　TCP thoughput of Android to Android Communication

4.2　高遅延環境におけるサーバとの通信

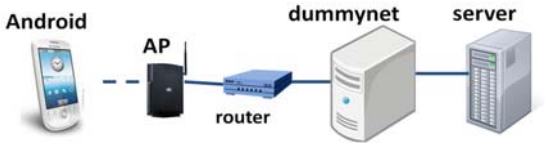

図 7　Android 通信性能測定の実験環境

図 7 に示すように, 間に人工的に遅延を発生させる装置である dummynet を用いて高遅延環境における Android 端末とサーバ間の通信性能を測定した. これは遠隔地に存在するモバイルクラウドを提供するサーバへアクセスする通信を想定している.

図 8 は横軸に dummynet により設定した往復遅延時間 (RTT) を取った時のスループットのグラフである. サーバの方が受信バッファが大きいため, 受信側がサーバの方が性能が良くなっている. しかし, 高遅延環境においては, 受信側がサーバである

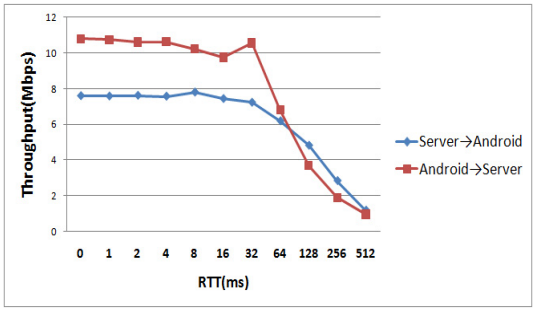

図 8　高遅延環境におけるサーバと Android 端末間の TCP スループット

方が性能が低下することが確認された. これは Android が高遅延環境に置いて十分に輻輳ウィンドウを増加させることができず, 輻輳ウィンドウ切れを起こしているためであると考えられる. カーネルモニタを導入し, 輻輳ウィンドウサイズを調べたところ,66 で頭打ちになって輻輳ウィンドウ切れが確認された. 本研究で用いた輻輳制御アルゴリズム [11] は default で実装されていた cubic アルゴリズム [10] である.

5.　複数台の Android 端末通信時の性能

この章では複数台の Android 端末を 1 台のアクセスポイントを使って通信させた時の通信性能をカーネルモニタを用いて評価する.

5.1　Android 端末 2 台の同時通信

今回は図 9 に示すように 2 台の Android 端末を使って実験を行った. この環境で実験を行った結果, 2 台で公平に通信できる場合と, 1 台のみが帯域を安定して使用し, もう 1 台の通信は不安定になるという 2 パターンに結果が別れることがわかった. その時のスループットと輻輳ウィンドウの関係をカーネルモニタを用いて解析する.

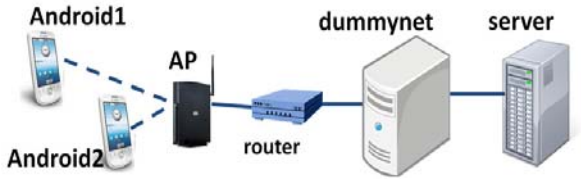

図 9　2 台の Android 端末による通信時の実験環境

5.2　安定した 2 台の Android 端末の通信

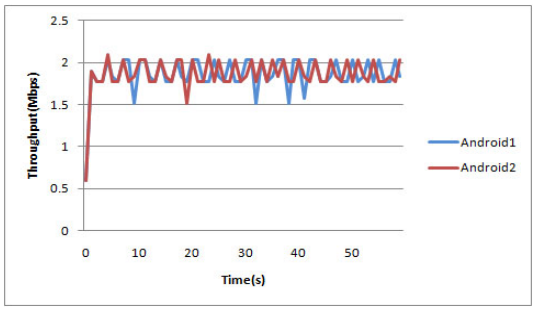

図 10　2 台通信時 TCP 通信スループット

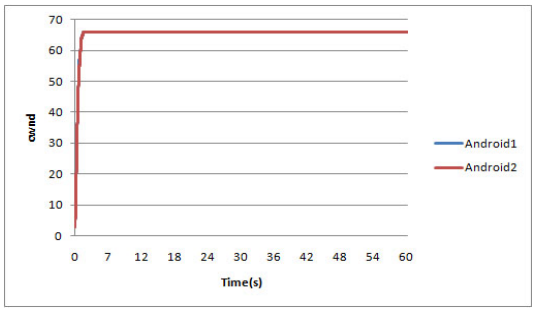

図 11　2 台通信時，輻輳ウィンドウサイズ

図 10,11 に 2 台とも安定した通信を行っている際のスループットと輻輳ウィンドウサイズを示す．このとき，RTT は 256(ms) でパケットロスは入れていない．図 11 では輻輳ウィンドウサイズの増加の仕方が同じで Android1 のグラフは重なって見えていない．輻輳ウィンドウが安定しているため，スループットも安定した結果が得られた．

5.3　１台が不安定な Android 端末の通信

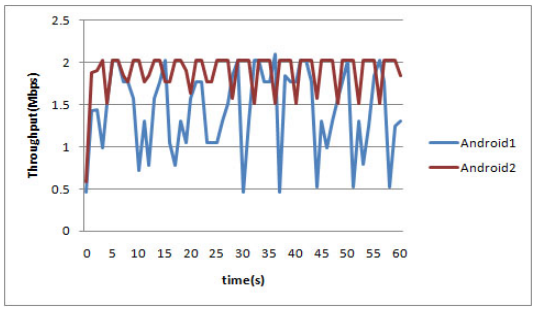

図 12　2 台通信時 TCP 通信スループット

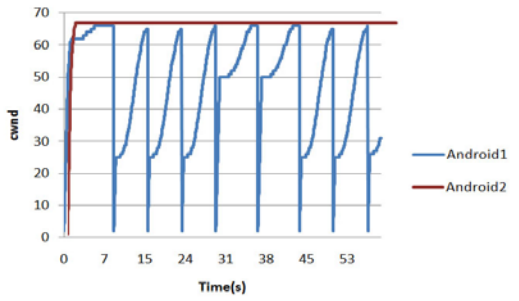

図 13　2 台通信時，輻輳ウィンドウサイズ

図 12,13 に 1 台が不安定な通信になる場合のスループットと輻輳ウィンドウサイズを示す．このとき，RTT は 256(ms) でパケットロスは入れていない．Android2 は輻輳ウィンドウが安定しており，スループットも安定した結果が得られたのに対し，Android1 は輻輳ウィンドウが安定せず，スループットも不安定な結果になることが確認された.

基本的に 2 台とも安定した通信を行うことができるが，タイミングや周囲の状況次第で結果が分かれていることが推測される.

6. 輻輳ウィンドウサイズ初期値の変更による性能向上と安定化

6.1 輻輳ウィンドウサイズ初期値変更

次に Google 社による文献 [9] を参考に initial Congestion Window Size（輻輳ウィンドウサイズ初期値）を 10 に変更しスループットの向上と安定を図った．このとき，RTT は 0(ms) でパケットロスは入れていない．輻輳ウィンドウサイズ初期値を 10 に変更した場合のスループットと輻輳ウィンドウサイズを図 14,15 に，輻輳ウィンドウサイズ初期値を変更せず 0 のままの場合のスループットと輻輳ウィンドウサイズを図 16,17 に示す．

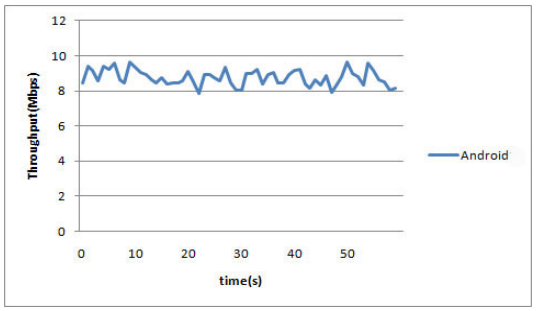

図 14　スループット（RTT=0ms,initial-cwnd=10）

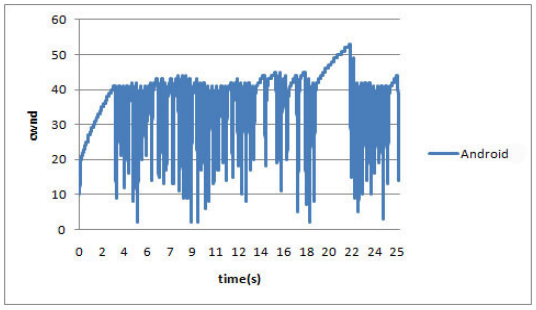

図 15　輻輳ウィンドウサイズ（RTT=0ms,initial-cwnd=10）

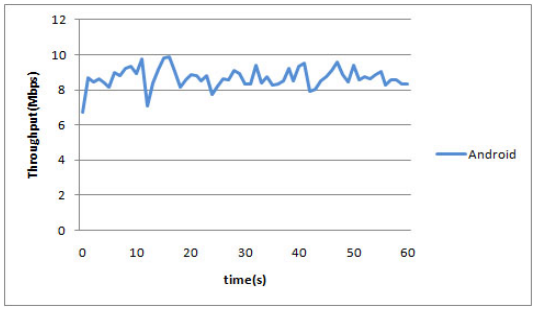

図 16　スループット（RTT=0ms,initial-cwnd=0）

図 14 から図 17 より，輻輳ウィンドウサイズ初期値が 0 のままの場合は通信開始時のスループットは 6.75(Mbps) であったのに対し，輻輳ウィンドウサイズの初期値を 10 にした場合，通信の初めから 8.45(Mbps) のスループットが測定され，性能が向上し安定することが確認された．

Android 端末を 2 台にし，2 台通信の性能も測定してみたところ，安定した通信が行える際は輻輳ウィンドウサイズの初期値を 10 にすることでの弊害は生じなかった．これは，TCP がスロースタートを行う際は極めて短時間で輻輳ウィンドウサイズを大きくするため，輻輳ウィンドウサイズはすぐに 10 に達するからである．

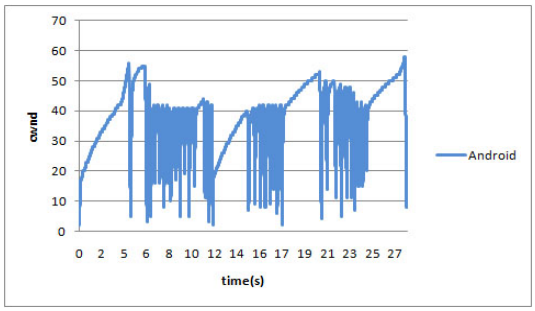

図 17　輻輳ウィンドウサイズ（RTT=0ms,initial-cwnd=0）

7. まとめと今後の課題

本論文では Android 実機にカーネルモニタツールを適用した．これを用い，Android 端末の通信時における輻輳ウィンドウの値を解析した．その結果，汎用 PC のカーネルモニタとほぼ同様な使い勝手で Android のカーネル内部の振舞を解析することが可能であるとわかった．またカーネルモニタを用いて輻輳ウィンドウとスループットの関係を解析することができた．

今後はこのモニタツールを用いて輻輳ウィンドウ以外の様々なパラメータを解析し，Android の特徴を見つけ，その詳細な振舞について研究を進めていきたい．本論文では手始めに輻輳ウィンドウサイズの初期値を 10 に設定しても通信に影響を与えることなくスループットが向上することを確認した．

次に複数台の Android 端末を通信させた際，2 台とも安定した通信ができる場合とそうでない場合に別れるためその原因を究明していきたい．無線の電波状況に影響されるかどうかも調べたい．また，不安定な通信を回避すべく，実行しているアプリケーションの要求やお互いの輻輳ウィンドウサイズなどパケットの情報を交換するミドルウェアを開発し，安定で効率的な通信を目指していきたい．

謝辞　本研究を進めるにあたり，ご指導して下さった株式会社 KDDI 研究所の竹森敬祐さん，磯原隆将さんに深く感謝致します．

文　　献

[1] Android:http://www.google.co.jp/mobile/android
[2] 間島　崇、横山　哲郎、曾　剛、神山　剛、冨山　宏之、高田　宏章: ”Androd プラットフォームにおける Dalvik バイトコードの CPU 負荷解析”, 情報処理学会研究報告,2010 年 3 月
[3] 山口実靖，小口正人，喜連川優：”iSCSI 解析システムの構築と高遅延環境におけるシーケンシャルアクセスの性能向上に関する考察”，電子情報通信学会論文誌 Vol.J87-D-I，No.2，pp.216-231，2004 年 2 月
[4] Iperf:http://downloads.sourceforge.net/project/iperf/iperf/2.0.4
[5] Sourcery G++ Lite 2008q-3-72 for ARM GNU/Linux:http://www.codesourcery.com/, http://www.codesourcery.com/sgpp/lite/arm /portal/release644
[6] 三木香央理, 山口実靖，小口正人, ”Android 端末の無線 LAN 通

信時のトランスポート層の振舞に関する一検討”, In Summer United Workshops on Parallel, Distributed and Cooperative Processing (SWoPP2010),2010 年 8 月.

[7] BUSYBOX:http://busybox.net/downloads/busybox-1.10.1.tar.gz

[8] Reika Higa, Kosuke Matsubara, Takao Okamawari, Saneyasu Yamaguchi, and Masato Oguchi, ”Analytical System Tools for iSCSI Remote Storage Access and Performance Improvement by Optimization with the Tools,”In the 3rd IEEE International Symposium on Advanced Networks and Telecommunication Systems (ANTS2009), December 2009.

[9] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin, ”An Argument for Increasing TCP’s Initial Congestion Window” ACM SIGCOMM Computer Communications Review, vol. 40 ,No.3, pp. 27-33,July2010. http://research.google.com/pubs/pub36640.html

[10] Sangtae Ha, Injong Rhee, and Lisong Xu, ”CUBIC: A New TCP-Friendly High-Speed TCP Variant” ACM SIGOPS Operating Systems Review, Volume 42 Issue 5, pp.64-74, July 2008.

[11] Habibullah Jamal and Kiran Sultan, ”Performance Analysis of TCP Congestion Control Algorithms” International Journal of Computers and Communications, Issue 1, Volume 2, pp.30-38, 2008.