

カーネルモニタを用いた Android 端末の 無線 LAN 通信性能の解析と性能向上のための一検討

三木 香央理^{†1} 山口 実 靖^{†2} 小 口 正 人^{†1}

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み込み機器のソフトウェアプラットフォームとして 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 の様な組み込み機器は, 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, 組み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, 組み込み機器において, カーネルの中の振舞を把握することが出来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, 組み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 の通信を評価した. さらに通信性能向上を目指し, TCP のソースコードに手を加えた.

A Study about Performance Improvement and Analysis of Communication on Android in a Wireless LAN with Kernel Monitor

KAORI MIKI,^{†1} SANEYASU YAMAGUCHI^{†2}
and MASATO OGUCHI^{†1}

In recent years, with the rapid growth of smart phone market, Android is drawing an attention as software platform of embedded system, used as a personal digital assistance developed by Google. While Android is taken notice for its flexible development of application software and expansion of the system, we are interested in optimization and performance evaluation of network computing ability of Android. Because an embedded system like Android has architecture different from that of general-purpose PC, and due to the poor function of I/O interface, it is difficult to grasp what happens inside the embedded system precisely. Therefore, it is interesting to analyze the communication behavior of Android. In this paper, we have developed a Kernel Monitor tool suitable for an embedded system that is able to observe the behavior of kernel. We have applied this tool for the Trans-

port layer of Android. We have shown that internal operation when an embedded system is communicating can be analyzed with our approach. Thus we have evaluated Android's communication performance with kernel monitor, and in order to improve communication, we have modified TCP source code.

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た. 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み込み機器のソフトウェアプラットフォームである Android¹⁾ 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⁾ など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フォームとしての 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.

携帯を含めた組み込み機器は, 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

^{†1} お茶の水女子大学

^{†2} 工学院大学

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。しかし組み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、組み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、組み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、組み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。また、理想的な通信を目指し、TCP のソースコードに手を加え、その挙動を評価する。

2. Android の概要

2.1 Android のアーキテクチャ

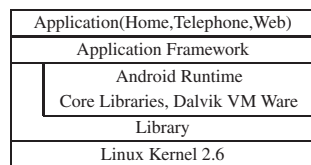


図 1 Android のアーキテクチャ

Android のアーキテクチャを図 1 に示す。Android は Linux 2.6 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 Android へほぼそのまま移植された 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。

2.2 クロス開発

携帯端末はディスプレイが小さく、CPU 性能やメモリ容量も高くないため、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 は組み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

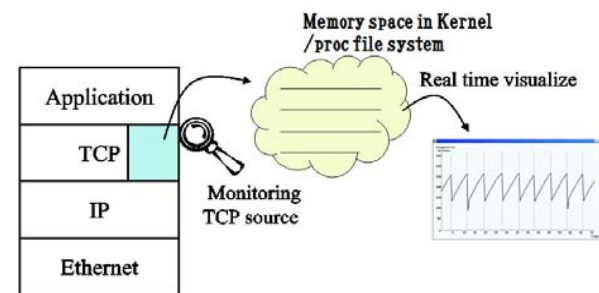


図 2 カーネルモニタの概要

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。カーネルモニタによって、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き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 携帯は組み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。

本研究では、組み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。

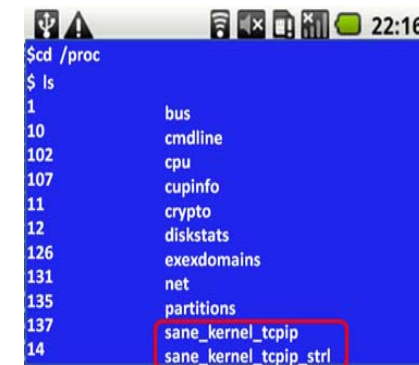


図 3 /proc ファイルシステム

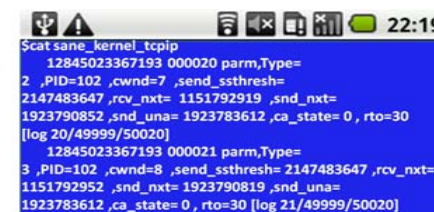


図 4 カーネルモニタのログ

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 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-2.0.4⁴⁾ 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-2008q3⁵⁾ を使用した。

5. 基礎実験：2 台の Android 端末通信時の性能

この章では 2 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

表 1 Experimental Environment

Android	Model number	AOSP on Sapphire(US)
	Firmware version	2.1-update1
	Baseband version	62.50S.20.17H_2.22.19.261
	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

性能をカーネルモニタを用いて評価する．

5.1 実験環境

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．今回は図 5 に示すように 2 台の Android 端末を使って実験を行った．

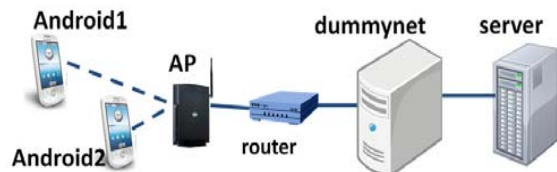


図 5 2 台の Android 端末による通信時の実験環境

この環境で 2 台の Android 端末それぞれとサーバ間の一対一通信実験を行った結果，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性をカーネルモニタを用いて解析する．

5.2 安定した 2 台の Android 端末の通信

図 6,7 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 dummynet においてパケットロスは入っていない．図 7 では輻輳ウィンドウサイズの増加の仕方が同じであり，Android1 のグラフは重なっているた

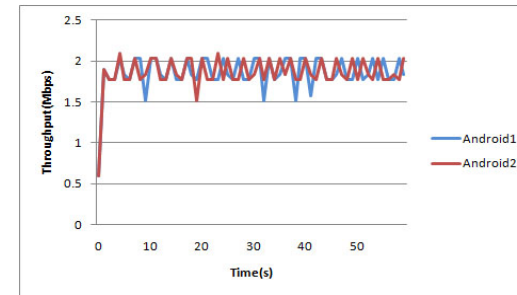


図 6 2 台通信時 TCP 通信スループット

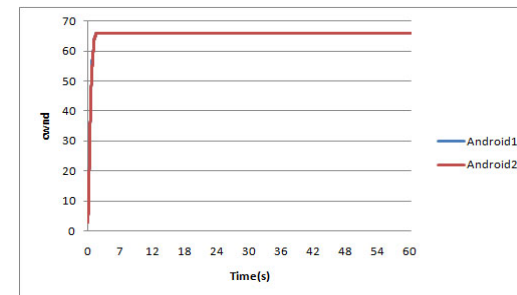


図 7 2 台通信時，輻輳ウィンドウサイズ

め表示されていない．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 8,9 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 dummynet においてパケットロスは入っていない．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可能な環境にはあるが，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．

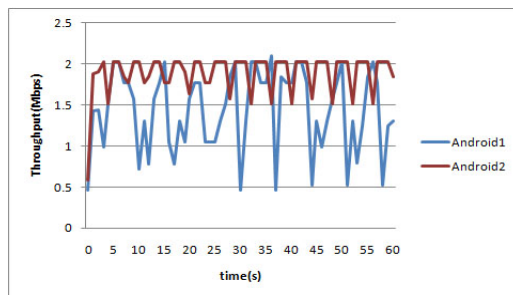


図 8 2 台通信時 TCP 通信スループット

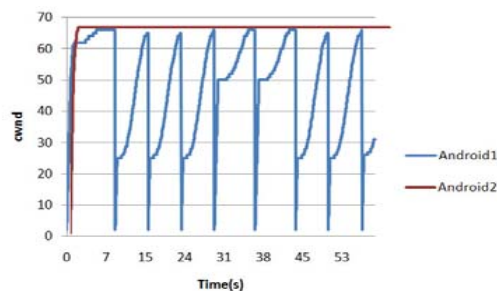


図 9 2 台通信時、輻輳ウィンドウサイズ

6. カーネルモニタによる通信状態を利用した通信

6.1 通信制御方針

本研究では基礎実験で示したような、1 台の AP に複数の端末が接続された場合に、全ての端末が安定した通信を行えることを目指す。そのために、カーネルモニタによって取得した通信状況を相互に交換し、安定した効率的通信を目指すミドルウェアの開発を考える。本論文では、基礎実験の 5.3 のように、本来は安定して通信が行える帯域が利用可能な状況でありながら、1 台のみが不安定な通信になってしまった場合は、輻輳ウィンドウサイズを下げないフェーズに移行させスループットの安定化を目指す。

基礎実験の 5.3 のような不安定な場合の結果に対し、基礎実験の 5.2 のように安定しているときには、2 台の端末のスループットはほぼ等しく、また輻輳ウィンドウサイズも 2 台と

も安定しており、一度上昇すると下がらないことが確認されている。従ってお互いの状況を把握できている場合に、意図的に輻輳ウィンドウサイズをフェーズに移行させることは合理的である。また、実際の利用形態として、Android は広帯域有線ネットワーク接続されたクラウドのサーバと通信する場合が多く、無線の限られた帯域を通過できれば、その先の有線通信ではパケットが溢れてしまうことは考えにくい。つまり AP までスムーズに通信することができれば、その後は支障なく通信できる可能性が高いと予想される。従って、その前提においては、AP まわりの状況に応じて最適化を行うことが効果的であると考えられる。

TCP は汎用性が高くなるように実装されているため、その制御がどのような状況においても最適であるとは限らない。特に、この実験のように AP で同時に 1~2 台しか通信を行っていないという状況は頻繁に起こると想像されるが、デフォルトの TCP の制御は控え目に抑えずぎてしまうと考えられる。従って何らかの手法により、そのような状況にあると判断できた場合に、より積極的な通信を行うよう制御を変更することを検討する。本論文ではそのような状況にある場合に適した振舞となるよう TCP のカーネル実装を変更し、性能評価を行う。

なお、本研究の以下の実験では 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。また輻輳ウィンドウサイズ初期値を 10 にすると、遅延がない状態で立ち上がりのスループットが 1.25 倍 (1.7(Mbps)) 程向上することを確認した¹²⁾。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増加させ、輻輳ウィンドウサイズはすぐに 10 に達するため、輻輳ウィンドウサイズ初期値を 10 にすることの弊害は生じない。また輻輳ウィンドウは送信側におけるパケット送出レートの自主制限であるため、送信側の制御を独自のものに変更したとしても、受信側との間で互換性の問題が生じる心配はない。

6.2 輻輳ウィンドウサイズ調節

本研究では、何らかの理由で輻輳ウィンドウサイズが必要以上に低下し、スループットが安定しない場合に輻輳ウィンドウサイズを意図的に大きくし、スループットの安定化を図る。実際に変更を行ったファイルは kernel/net/ipv4 配下の tcp_input.c, tcp_output.c, tcp_cong.c, tcp_ipv4.c, tcp_cubic.c の 5 つのファイルである。1 台のみが不安定なフェーズに入るときは、AP から Android 端末までの間で ACK パケットが何らかの理由で到達できなかったときである。その様な場合、実際には輻輳が起きている訳ではないにも関わらず不必要に輻輳ウィンドウを下げてしまうと、輻輳が起こる前の状態まで輻輳ウィンドウサイズを増加させるのに時間が掛かってしまう。そのため、上記 5 つのファイルを変更し、輻輳ウィンドウサイズが下がり

ないように、輻輳ウィンドウサイズダウンのフェーズを省き、また輻輳ウィンドウサイズ閾値 (ssthresh) の値を変更し、安定した輻輳ウィンドウサイズを保てるようにした。以下に実験結果を示す。

6.3 高遅延環境におけるサーバとの一対一通信

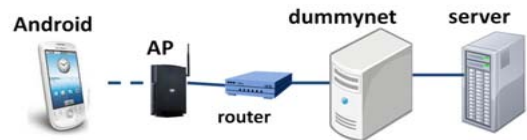


図 10 サーバとの一対一通信実験環境

図 10 に示すように、輻輳ウィンドウサイズ変更後の一対一通信の往復遅延時間(RTT) ごとのスループットを測定した。Android1 は輻輳ウィンドウサイズが未変更のもので Android2 は輻輳ウィンドウサイズを変更させたものである。

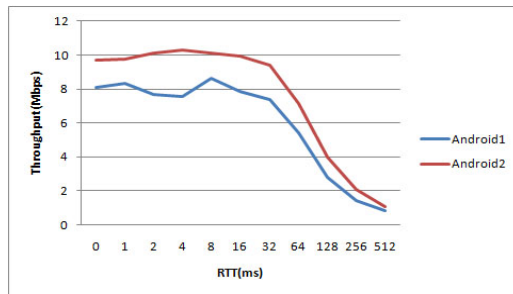


図 11 高遅延環境におけるサーバと Android 端末間の TCP スループット

一対一通信の場合、図 11 に示すように輻輳ウィンドウサイズを変更したもの (Android2) が未変更のものよりも 20 % ~ 40 % 程スループットが向上することが確認された。輻輳ウィンドウサイズを変更しても安定したスループットを取得することができたことから、条件のいい環境では、こちらの輻輳ウィンドウサイズ変更のフェーズに切り替える方が良いといえる。

6.4 輻輳ウィンドウサイズ変更フェーズへの切替え

次に、不安定な状況になった場合に先程の輻輳ウィンドウサイズ変更のフェーズに切替える実験を行う。基礎実験と同様の環境で測定を行う。図 12,13 に輻輳ウィンドウサイズ未変更の Android1、輻輳ウィンドウサイズ変更の Android2 のスループットと輻輳ウィンドウサイズを示す。

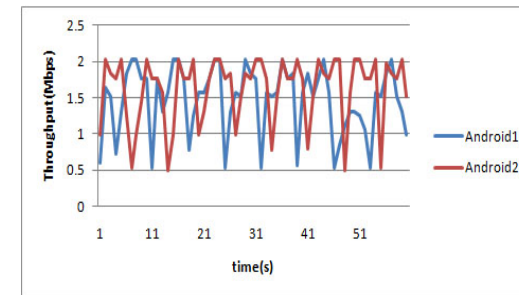


図 12 輻輳ウィンドウ調節時スループット

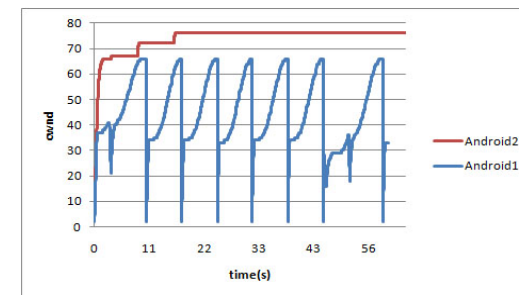


図 13 輻輳ウィンドウ調節時輻輳ウィンドウ

このとき、RTT は 256(ms) で dummynet においてパケットロスを入れていない。それぞれの平均スループットは Android1 は 1.47(Mbps), Android2 は 1.63(Mbps) となり、Android2の方が 1.12 倍 (12 %) ほど向上することが確認された。図 13 から Android1 は輻輳ウィンドウサイズが安定しないのに対し Android2 は輻輳ウィンドウサイズが安定することが確認さ

れ、一時的にスループットが低下することはあっても、Android2 のように変更した方が常に平均スループットが良いことが確認された。

6.5 2 台の Android 端末通信時の性能

次に、図 5 に示すような 2 台の Android 端末が同時に通信を行う環境における評価を行った。RTT に対するスループットの平均を表 2 に示す。表 2 は TCP を変更した Android と未変更の Android を 2 台同時に通信させた場合である。網掛けしている方が高性能であることが確認された。この実験結果は、ローカル接続のような低遅延環境では、輻輳ウィンドウサイズを大きく保つ今回の変更を適応すると、かえってスループットが低下することを表している。これは送信データに確認応答がすぐ返る低遅延環境においては輻輳ウィンドウを大きく保つとパケットが多く飛びすぎ、かえって性能を低下させてしまうことを意味している。しかし携帯端末の通信相手としてはローカル接続のマシンより、遠方のクラウドサーバ等の方が一般的と考えられ、そのような高遅延環境においては本手法が有用であることが確認できた。

また 2 台とも輻輳ウィンドウサイズを変更した端末で実験を行うと、RTT=64(ms) 以上の高遅延環境では本手法が 2 台とも安定した通信に有用であることが確認されたが、遅延の小さい環境では、2 台とも本来の TCP を用いた方が性能が良いことが確認された。2 台とも同じ TCP を用いた結果の 2 台のスループットの総和を表 3 に示す。表 3 は未変更の Android 端末 2 台を同時に通信させたものの 2 台の平均スループットの総和と、2 台とも TCP を変更した Android 端末のスループットの平均の総和である。網掛けしている部分の方が高性能であることが確認された。

つまり本手法は RTT=64(ms) 以上の高遅延環境では 1 台のみ変更 TCP を用いる場合も、2 台とも変更 TCP を用いる場合も有用であるということが確認された。

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性を解析することができた。

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ことが確認された。そのような不安定な通信を回避すべく、実行しているアプリ

表 2 未変更対変更通信

RTT	未変更	変更
0	7.34	4.22
1	7.02	4.36
2	6.65	4.05
4	7.27	4.66
8	7.21	4.57
16	7.43	5.18
32	7.09	5.71
64	4.99	5.9
128	2.86	3.64
256	1.47	1.68

表 3 同 TCP 同士 2 台通信の総和

RTT	未変更 2 台の総和	変更 2 台の総和
0	15.13	10.44
1	14.71	9.35
2	14.82	11.15
4	13.08	11.22
8	15.09	11.68
16	15.02	12.42
32	15.01	13.94
64	10.46	12.05
128	5.85	6.78
256	2.89	3.33

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を目指しており、本論文では AP で同時に通信を行っている端末の台数が少ない場合に適した振舞となるように輻輳ウィンドウサイズの調節を行った。本手法を用いることで、不必要に輻輳ウィンドウサイズを低下させることなく通信を行うことが可能になり、一対一通信、または高遅延環境における 2 台通信においてスループットが向上することが確認された。

本手法は RTT64(ms) 以上の高遅延環境では有用であるということが確認された。Android のような携帯端末はモバイルクラウドにアクセスして通信することが一般的と考えられるため、本手法は有用であると考えられる。

今後は、カーネルモニタにより取得した値から本手法を適応するか、本来の TCP を適応するか、状況に応じて柔軟に通信を切り替えることができるミドルウェアを開発していく。

謝 辞

本研究を進めるにあたり、ご指導して下さいました株式会社 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。

参 考 文 献

- 1) Android: <http://www.google.co.jp/mobile/android>
- 2) 間島 崇、横山 哲郎、曾 剛、神山 剛、富山 宏之、高田 宏章: "Android 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", 並列/分散/協調処理に関するサマー・ワークショップ SWoPP2010.05-4, 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, July 2010. <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.
- 12) 三木香央理, 山口実靖, 小口正人, "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", A study about performance of communication on Android terminals in a wireless LAN with Kernel Monitor(DEIM2011), 2011 年 3 月.