Vol.2015-GN-93 No.24
Vol.2015-CDS-12 No.24
Vol.2015-DCC-9 No.24
2015/1/27

Android 端末における高遅延環境通信性能に関する一考察

神津智樹† 秋山友理愛† 中村優太† 野村駿† 坂本寛和† 山口実靖†

Android OS はスマートフォンやタブレット PC など様々なデバイスで採用されており，重要なプラットフォームになっている．スマートフォンによるインターネットへの接続者数は年々増加しており，今後更に増加すると考えられる．よって，Android OS の通信速度の向上や，性能公平性の向上は重要な課題であると言える．Android OS は Linux をベースとして開発された OS であり，標準実装の TCP アルゴリズムには Linux で標準となっている CUBIC TCP を用いている．本稿では，高遅延環境における Android OS の通信性能，RTT 公平性を評価し，性能改善手法について考察する．

1. はじめに

スマートフォンを用いたインターネットサービスへのアクセス者数が増加しており[1]，スマートフォンはインターネットアクセスを行う主要なデバイスの一個となっている．スマートフォンやタブレット PC のプラットフォームに Android がある．Android の 2014 第三四半期の世界市場におけるシェアは 84.4%であり[2]，Android 搭載の端末は重要なインターネット接続端末となっている．よって，Android 搭載のスマートフォンやタブレット PC の通信性能の評価や改善は，重要な課題であると考えられる．特に，災害発生時などのネットワーク負荷が急激に増大する状況では，スマートフォンやタブレット PC によりネットワーク接続を行うユーザが多いと考えられ，急激な負荷変動発生時の性能に関する考察は重要であると考えられる．

本稿では，Android OS を搭載した端末の TCP 通信速度に着目し，性能評価，RTT 公平性の評価，性能向上手法について述べる．また，負荷変動時性能を想定して通信遅延時間変動時に関する評価と考察を行う．

2. Android

Android OS はスマートフォンやタブレットなどの携帯端末用に開発されたオープンソースであるソフトウェアプラットフォームである．アプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイム，Linux カーネルで構成されている**エラー! 参照元が見つかりません。**．

Android のカーネルは Linux カーネルを元に作成されており，現在のところ大きな修正は行わずに Linux カーネルが使用されている．本稿で着目する TCP 実装部も，Linux カーネルの TCP 実装を元に作成されている．よって，Linux で標準 TCP アルゴリズムとして採用されている CUBIC TCP **エラー! 参照元が見つかりません。**が Android でも標準 TCP アルゴリズムとして採用されている．

アプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイムは Android のために新規に開発された実装が多く含まれている．

3. 関連研究

3.1. TCP 輻輳制御アルゴリズム

過剰なパケットを送出しネットワークの輻輳を招くことを避けるために，TCP 実装には輻輳制御アルゴリズムが搭載されている．TCP によるパケット送出量はこの輻輳制御アルゴリズムにより制限されており，TCP の通信性能はこのアルゴリズムに大きな影響を受ける．OS により様々な TCP が実装されおり，輻輳制御の仕方が異なる．

TCP の輻輳制御手法は主に，ロスベース手法，遅延ベース手法，両者を組み合わせたハイブリッド型の手法に分類することができる．

ロスベース手法はパケットロスの検出に基づき輻輳ウィンドウを制御する手法である．通常時は確認応答を受信するたびに輻輳ウィンドウを増加させ，パケットロス検出時に輻輳ウィンドウを大幅に減少させる．従来の TCP である TCP Reno がロスベース手法であり，代表的なロスベース手法の高速 TCP に BIC TCP[5] や，CUBIC TCP[4]がある．

遅延ベースの輻輳制御アルゴリズムは，RTT の増減にあわせて輻輳ウィンドウを変化させる手法である．ロスベース手法の様に輻輳が発生してから速度を減少させるのではなく，RTT の増加からネットワークの混雑状況を推定し，輻輳が発生する前に速度を減少させるため安定した通信速度が期待できる．欠点としてロスベース手法とネットワークを共有したときに得られる性能が低くなってしまうということが指摘されている[6]．代表的な遅延ベース手法に TCP Vegas[7] がある．

ハイブリッド型手法はロスベースと遅延ベースの両者を組み合わせた手法であり，代表的なハイブリッド型手法の TCP に Windows Vista 以降の Windows 系 OS に標準で搭載されている Compound TCP[8]がある．

本研究では，Android の標準 TCP アルゴリズムである CUBIC TCP に焦点を当て考察を行う．また，PC で広く使われている Window OS の TCP である Compound TCP に関する考察も合わせて行う．

3.2. CUBIC TCP

CUBIC TCP[4]は，BIC TCP のスケーラビリティを維持し

† 工学院大学大学院 電気電子工学専攻
Electrical Engineering and Electronics,
Kogakuin University Graduate School

Vol.2015-GN-93 No.24
Vol.2015-CDS-12 No.24
Vol.2015-DCC-9 No.24
2015/1/27

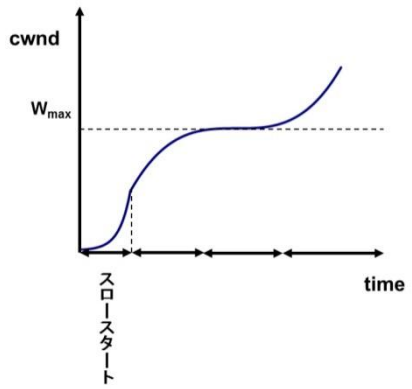

図 1. CUBIC TCP の輻輳ウィンドウの推移

ながら，TCP-Fairness，RTT-Fairness，制御手法の複雑さを

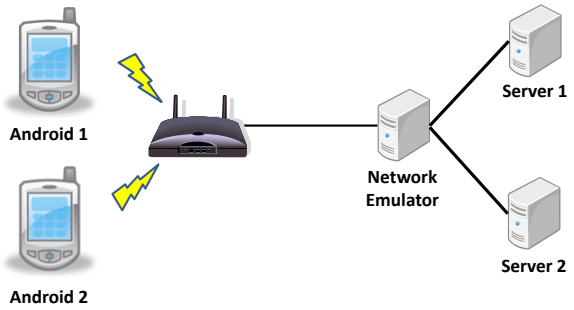

図 2. 実験環境

表 1. 実験機使用

Android 1，Android 2	
Device nar	Nexus 7
OS	Android4.4.4
CPU	NVIDIA Tegra 3 T30L 1.3GHz クアッドコア
Memory	1GB
Server 1，Server 2	
OS	Cent OS 6.3
kernel	linux-2.6.32.27
CPU	Intel(R) Celeron(R) CPU G530 @ 2.40GHz
Memory	2GB

改善した高速 TCP である．CUBIC TCP は Linux2.6.19 以降で標準の TCP として搭載されている．

CUBIC TCP では，BIC TCP のバイナリーサーチを用いて利用可能帯域を検索するアルゴリズムを式(1)，(2)のような 3 次関数を用いた制御によって実現している．その輻輳ウィンドウの推移は図 1 のようになる．

$$cwnd = C(t-K)^3 + W_{max} \quad (1)$$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \quad (2)$$

ここで，$cwnd$ は輻輳ウィンドウサイズ，t はパケットロス検出時からの経過時間，W_{max} はパケットロス検出時の輻輳ウィンドウサイズ，C は増加幅を決めるパラメータ，β はパケットロス検出時のウィンドウサイズ減少幅を表している．通常，C には 0.4 が，βには 0.2 が用いられている．CUBIC TCP では，上記のようにパケットロス検出時からの経過時間を用いて輻輳ウィンドウの値を定めている．これは，RTT の影響を強く受ける Ack の受信を輻輳ウィンドウサイズの増加処理から排していることを意味し，これにより RTT-Fairness が向上することが期待できる．加えて，BIC TCP の低遅延環境で輻輳ウィンドウサイズを急速に成長させすぎる問題もこれにより解決している．

3.3. CUBIC TCP の RTT 公平性改善手法

文献**エラー! 参照元が見つかりません。エラー! 参照元が見つかりません。**では，異なる RTT 同士の CUBIC TCP が競合すると，RTT の大きいコネクションのスループットが低くなり，RTT 公平性が低くなることが示されており，CUBIC TCP の輻輳ウィンドウ回復時間式である K を調整することにより RTT 公平性を改善することが示されている．当該手法では，K を次式のように調整し，RTT が大きいコネクションの輻輳ウィンドウが短い時間で回復できるように制御している．

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \times \frac{1}{\sqrt[3]{RTT}} \quad (3)$$

4. 高遅延環境における Android 通信性能

4.1 高遅延環境における Android OS の通信性能評価

本章では，高遅延環境における Android OS の通信性能の評価を行う．

図 2 のネットワークを構築し，2 台の Android 端末(Android 1 および Android 2)から PC サーバ(Server 1，Server2)のデータ送信を行い，スループットを測定した．ネットワークエミュレータは人工的にネットワーク遅延を発生させる装置であり，FreeBSD Dummynet を用いて構築した．Android 1，Android 2，Server 1，Server 2 の仕様は表 1 の通りである．

通信性能は，以下の方法で計測した．Android 端末(Android 1 と Android 2)から PC サーバ(Server 1 と Server2)にファイルのアップロードを行い，その時のスループットを測定した．ファイルのアップロードは，プロトコルとして HTTP で行い(HTTP PUT によるファイルをアップロード)，ソフトウェアとしては Android 搭載の標準ブラウザと，Apache httpd を用いた．

性能評価実験は，Android 端末一台と PC サーバ Server 一台の間における通信(“単独通信”と呼ぶ)の速度の測定と，Android 端末 2 台と PC サーバ 2 台間の同時通信(“競合通信”)の速度の測定を行った．単独通信では，ネットワークエミュレータによりネットワーク遅延を 128ms，256ms，512ms，1024ms に設定し，それぞれでスループットを測定した．競合通信では，ネットワークエミュレータにより Android

Vol.2015-GN-93 No.24
Vol.2015-CDS-12 No.24
Vol.2015-DCC-9 No.24
2015/1/27

1-Server 1 間の RTT を 128ms に固定し，Android 2-Server 2

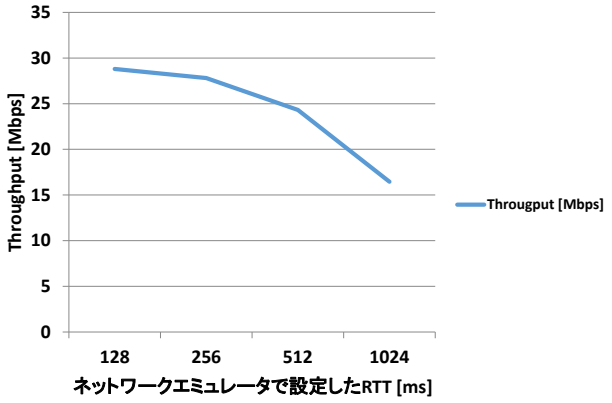

図 3. 単独通信による測定結果

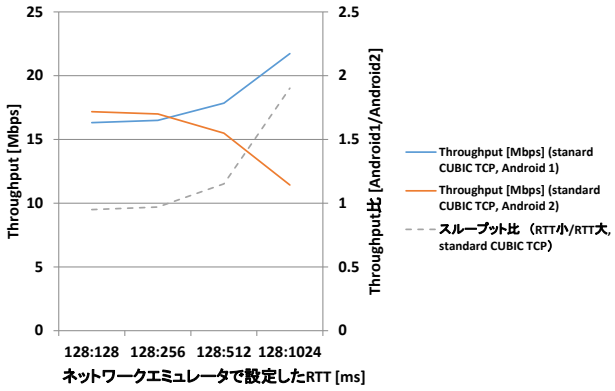

図 4. 競合通信による測定結果　(standard CUIBC TCP)

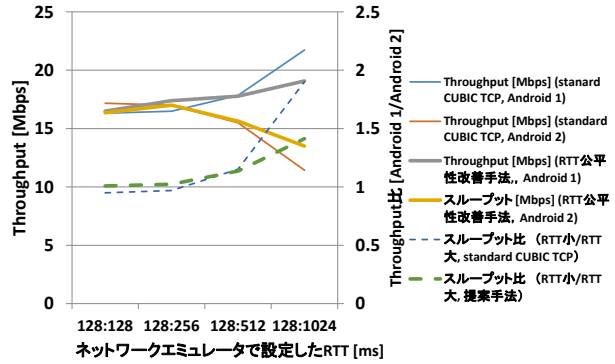

図 5. 競合通信による測定結果（RTT 公平性改善手法）

間の RTT を 128ms，256ms，512ms，1024ms と変動させそれぞれにおいてスループットを測定した．

実験結果を図 3〜5 に示す．図 3 は単独通信によって得られたスループットであり，RTT が大きくなるに従い，スループットが下がっていることがわかる．ネットワーク遅延が 512ms や 1024ms など巨大な環境では無線 LAN で得られる限界性能に近い性能を維持できないことが確認できる．図 4 に競合通信によって得られた評価結果を示す．実線で示している線が測定によって得られたスループットであり，左側の縦軸に対応している．破線で示している線が Android 1 で得られた性能と Android 2 で得られた性能の公平度であり，右側の縦軸に対応している．公平度は Fairness Index **エラー! 参照元が見つかりません。** を用いて計算し，以下の式で計算される．

$$\text{Fairness Index} = \frac{(\sum_{i=1}^{n} b_i)^2}{n \times \sum_{i=1}^{n}(b_i^2)}$$

Fairness Index は 0 から 1 の間の値をとり，1 に近いほど公平度が高いことを示している．公平度に注目すると，RTT の差が小さい時は Fairness Index がほぼ 1 であり公平性が高いことがわかるが，RTT の差が大きくなるにつれ Fairness Index が低下していることがわかる．これより，RTT の差が大きいほど Android 1 と Android 2 で得られたスループットの差が開いていることがわかる．すなわち，RTT の異なるコネクションが競合する場合は通信性能に不公平が生じている(RTT 不公平性が生じている)ことがわかる．

4.2 RTT 公平性改善手法による通信性能評価

本節では，3 章で説明した CUBIC TCP の RTT 公平性改善手法を用いた通信性能評価を行う．

図 2 のネットワークを用いて，4.1 節の競合通信と同様の測定を行った．測定方法は前節の競合通信と同じである．評価結果を図 5 に示す．図より，CUBIC TCP の RTT 公平性改善手法を用いることにより，RTT 公平性を大幅に改善できることが分かる．

4.3 競合通信時の *K*

次に，4.1 節(標準 CUBIC TCP)，4.2 節(RTT 公平性改善手法適用 CUBIC TCP)の実験における *K* の推移を示す．3 章で述べた様に，当該 RTT 公平性改善手法は RTT が大きな TCP コネクションの *K* を短くし，そのコネクションの輻輳ウィンドウの回復を促し，RTT が大きなコネクションの通信性能の改善を行っている．図 6 から図 9 に，標準 CUBIC TCP および改善手法適用 CUBIC TCP を使用した時の *K* の推移を示す．図より，標準 CUBIC TCP では RTT が大きいコネクションの方が *K* が大きくなっており，これにより RTT 不公平性が発生していることが分かる．これに対して改善手法適用 CUBIC TCP では RTT が大きいコネクションの方が *K* が小さくなっており，これにより RTT 公平性が改善されていると考えられる．

4.4 通信遅延時間が動的に変動環境での評価

次に，ネットワーク遅延時間が動的に変動する環境における評価を行う．ネットワーク遅延を全通信時間の中心で変更して性能を評価した．標準 CUBIC TCP および改善手法適用 CUBIC TCP の性能を図 10 に示す．図より，改善手法適用を適用することにより公平性が悪化していることを確認できるが，改善手法適用前と適用後の通信性能を比較すると全てのコネクションにおいて通信性能向上が実現されていることが分かり，当該改善手法は有効であると考えられる．

Vol.2015-GN-93 No.24
Vol.2015-CDS-12 No.24
Vol.2015-DCC-9 No.24
2015/1/27

4.5 PC と競合する環境での評価

最後に，Android 端末と PC が競合する環境における両端

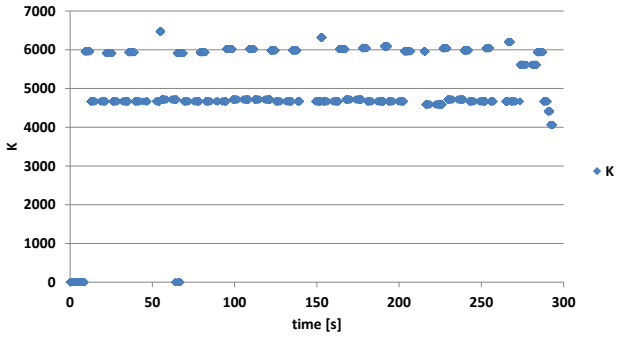

図 6. standard CUBIC TCP による K の推移 (RTT=128ms)

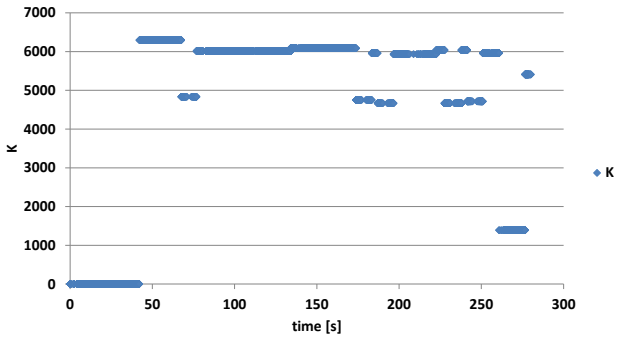

図 7. standard CUBIC TCP による K の推移 (RTT=1024ms)

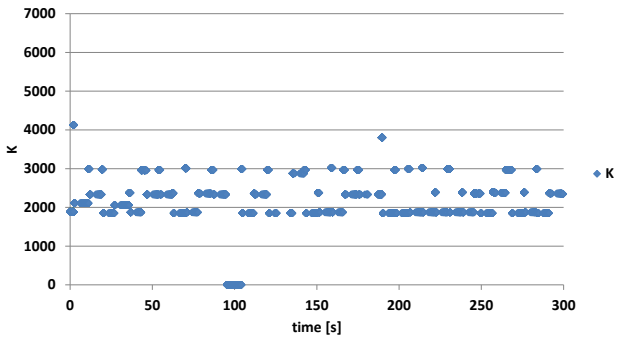

図 8. RTT 公平性改善手法による K の推移 (RTT=128ms)

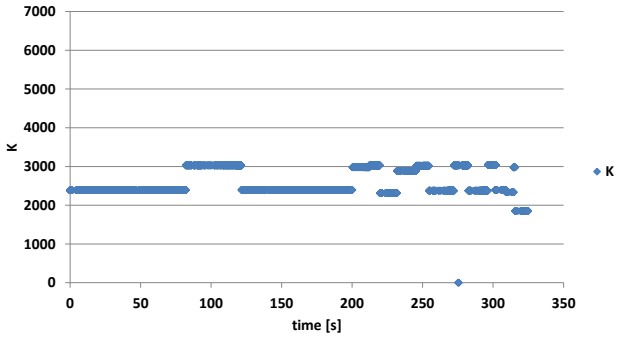

図 9. RTT 公平性改善手法による K の推移 (RTT=1024ms)

末の通信性能の評価を行う．PC としては，代表的な OS ある Windows が搭載されているものを使用した．4 章で前述の様に，Windows には Compound TCP が搭載されている．使用した PC の仕様は表 2 の通りである．

表 2. PC(Windows)の使用

PC	
モデル	ASUS EeePC X101H
OS	Windows 7 32bit
CPU	Intel® Atom™ CPU N570 @ 1.66GHz 1.67GHz
Memory	1GB

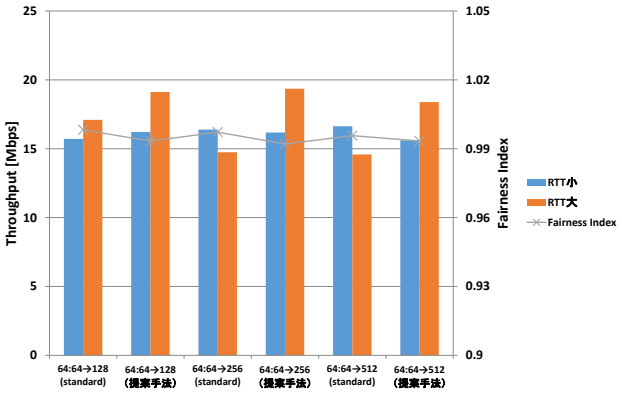

図 10. 動的な通信時間遅延による測定結果

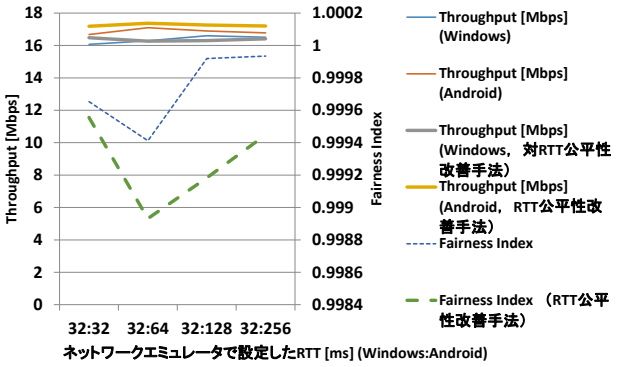

図 11. PC(Windows OS)との競合による測定結果

性能測定結果を，図 11 に示す．図より，改善手法を適用しても Windows OS 搭載端末との競合時には RTT 公平性に大きな改善がないことが分かる．同時に，Windows OS 搭載端末の性能を大きく劣化させていないことも分かり，当該改善手法は他の OS 搭載端末に負の影響を与えることなく CUBIC TCP の通信間の公平性改善を実現していることがわかる．

5. おわりに

本稿では，RTT の異なる環境や RTT が動的に変化する環境における Android 搭載端末の通信性能に着目し，通信性能の評価，RTT 公平性の評価，RTT 公平性改善手法の効果

Vol.2015-GN-93 No.24
Vol.2015-CDS-12 No.24
Vol.2015-DCC-9 No.24
2015/1/27

の評価を行った. 評価の結果, 標準の CUBIC TCP では RTT の差が通信性能に影響を与えることが分かり，RTT 公平性改善手法によりその差を軽減できることが分かった.

今後は，様々な OS を使用しての評価，実ネットワークを介しての評価を行っていく予定である.

謝辞

本研究は一部，総務省戦略的情報通信研究開発推進事業（SCOPE）先進的通信アプリケーション開発推進型研究開発によるものである.

文　献

[1] ニールセン
http://www.netratings.co.jp/news_release/2014/12/Newsrelease20141216.html

[2] IDC
http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[3] What is Android?|Android Developers:
http://developer.android.com/guide/basics/what-is-android.html

[4] Injong Rhee and LisongXu “CUBIC: A New TCP-Friendly High-speed TCP Variant,” Proc.Workshop on Protocols for Fast Long Desitance Networks, 2005

[5] L. Xu, K. Harfoush and I. Rhee, “Binary Increase Congestion Control for Fast Long-Distance Networks,” Proc. IEEE Info COM 2004, March 2004

[6] Jeonghoon Mo, Richard J. La, VenkatAnantharam, and Jean Walrand, “Analysis and comparison of TCP reno and vegas”, in Proceedings of IEEE INFOCOM’99, March 1999.vegas

[7] comound L. S. Brakmo and L. L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet”, IEEE Journal on Selected Areas in Communication, Vol.13, No.8, pp.1465-1480, October 1995.

[8] Kun Tan, Jingmin Song, Qian Zhang, and MurariSridharan,” A Compound TCP Approach for High-speed and Long Distance Networks” Proc.IEEE Info COM 2005,July 2005.

[9] Tomoki Kozu, Yuria Akiyama and Saneyasu Yamaguchi, "Improving RTT Fairness on CUBIC TCP", The First International Symposium on Computing and Networking

[10] Tomoki Kozu, Yuria Akiyama and Saneyasu Yamaguchi, "Improving RTT Fairness on CUBIC TCP", International Journal of Networking and Computing

[11] D.Chiu and R.Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” Computer Networks and ISDN Systems, Volume 17, Issue 1, pp. 1-14, 1989.