無線LAN環境におけるAndroid端末向け通信制御ミドルウェアの実装

平井　弘実† 　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: †{hiromi,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を用いて無線通信でクラウドにアクセスし，データ通信を行うことが頻繁に行われるようになった．クラウドを構成するデータセンタは遠隔地にあることが多く，高遅延通信を行うこととなる．高遅延通信においては，輻輳ウィンドウの値が通信スループットに大きな影響を与える．またクライアント・サーバ間の通信においては，モバイル環境における身近なアクセスポイントまでが無線通信で，アクセスポイントからサーバまでは有線通信で行われる．すなわち発生するパケットロスは大部分が無線空間で起きたものだと考えることができる．本研究は，同一アクセスポイントを共有する無線空間で互いの通信状況を知らせ合うことにより，各々の端末の輻輳制御を行おうとする試みである．その制御としてカーネル内に組込んだ独自の TCP との切り替えを行うミドルウェアを開発した．

キーワード　Android，モバイルネットワーク，TCP/IP，輻輳制御

An implementation of Transmission-Control Middleware on Android Terminal in a Wireless LAN Environment

Hiromi HIRAI†, 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: †{hiromi,kaori}@ogl.is.ocha.ac.jp, ††sane@cc.kogakuin.ac.jp, †††oguchi@computer.org

Key words　Android, Smart phone, Mobile network, TCP/IP, Congestion control

1. はじめに

近年，日本独自仕様の携帯電話に代わりスマートフォンが爆発的に普及し始めている．従来の携帯電話は電話・電子メールに加えて低トラフィックなインターネットアクセスが可能であったが，スマートフォンは小型コンピュータという位置付けであり，多くの機能が実現された．従来の携帯電話では，OS に組み込まれた唯一のメーラやブラウザしか利用できなかったが，スマートフォンでは，自分の気に入ったメーラ・ブラウザのアプリケーションをインストールして，利用形態に合うようにカスタマイズすることができる．またアプリケーションとインターネットの連携により，いつでも現在地周辺の地図を表示するアプリケーションや毎日天気やニュースなどの情報を届けるアプリケーションなどがある．ネットワークを利用するアプリケーションは，これらのアプリケーションのようにサーバと連携して情報を受発信するものと，Skype や LINE のようにクライアント同士で情報を受発信するものに分けることができる．本研究は，Ustream のようなクライアント・サーバ間通信におけるクライアント側からの発信におけるパケット転送制御に注目した．

クライアント・サーバ間通信は，クライアントのモバイル端末から身近なアクセスポイントまでの無線通信と，アクセスポイントからサーバまでの有線通信で繋がっている．モバイル端末が発信するデータ量のみでは広帯域な有線通信経路上でバッファ溢れを起こす可能性は低いと考えられる．無線空間では，ユーザの移動によって，一台のアクセスポイントに繋がっている端末数も変わりやすく，トラフィックにも変動がある．すなわち発生するパケットロスの大部分は同じアクセスポイントを共有する端末が多い時や一人一人の転送量が多大な時に起きていると考えることができる．

本研究は，クライアント・サーバ間通信において，クライアント側からのパケット発信の際に，クライアントのアクセスポイント周りで，互いの通信状況を知らせ合うことにより，輻輳を回避し最適な通信環境を実現する制御を行おうとする試みである．この概念図を図 1 に示す．既存研究においても，無線通

信を行うモバイル端末間で連携して通信の最適化を行うという手法はこれまでになく，本研究が初の試みと言える．これまでのモバイル端末はリソースが最小限であったため，大きな負荷に耐えられず，端末間で高度な制御を行う手法は現実的ではなかったが,，現在スマートフォンの需要が高まり，ハードウェアのスペックがますます向上したため，このような他端末と連携した制御が可能になりつつある．

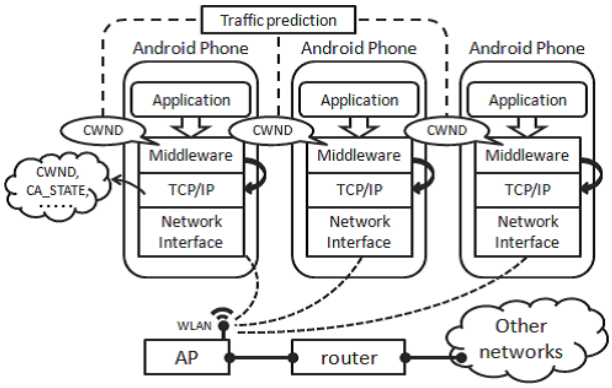

図 1　本研究が提案する通信制御手法の概念図

本稿では以下 2 つの制御手法を紹介する．1 では Android1.6 を搭載した HTC 社製 HT-03A を用いた制御であり，帯域占有時に独自のアルゴリズムに切り替え強気な通信をするというものである．2 では，Android2.3 を搭載した Samsung 社製 Nexus S を用いて，HT-03A との混在環境で公平性を向上させる制御である．

1.HT-03A の現行の輻輳制御が，輻輳崩壊やバッファ溢れを起こさないために，輻輳ウィンドウを増加させにくく，減少しやすい慎重なアルゴリズムとなっており，帯域を余らせてしまう場合が生じるというデメリットに注目した．本実験では，周囲の端末と通信状況を通知し合うミドルウェアを導入し，アクセスポイントを占有した時のみ，オリジナルのアルゴリズムに切り替えるという制御を行った．オリジナルの制御とは独自に開発した輻輳ウィンドウを増加させやすく，減少させにくいアルゴリズム [6] を指す．

2. 比較的新しく HT-03A よりも高性能である Nexus S と HT-03A が共存する環境における不公平を明らかにし，公平性を向上する実験を行った．本手法では，HT-03A の慎重な輻輳制御アルゴリズムを Nexus S に移植し，Nexux S と HT-03A が混在するヘテロな環境で公平性が向上したという報告に基づき，そのような状況になったことをミドルウェアが感知するとアルゴリズムを切り替えるように実装した．

2. Android OS

Android は，OS，ミドルウェア，アプリケーション，ユーザインタフェースをセットにしたモバイル端末向けプラットフォームであり，Google 社を中心として開発が行われている．図 2 に示すように，Android は Linux2.6 をベースとし，スマートフォンやタブレット端末をターゲットに，それらに適したコンポーネントが追加されている [2]．他の Linux OS と大きく異なる部分は，独自に開発された Android の Runtime である Dalvik 仮想マシンを搭載している点である．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 仮想マシンに合わせて開発すれば，直感的な操作性に優れた UI を利用することができ，移植性も高い．

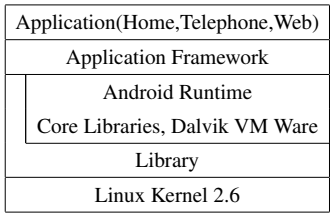

図 2　Android のアーキテクチャ

2.1 Android アプリケーション

Android は，無償で提供される開発環境において構築することができ，オープンソースである点からも対応アプリケーションが開発しやすく数も増えるというメリットがある．また Android はキャリア間の制約がないため，アプリケーション開発においても自由度及び汎用性が高いだけでなく，一度マーケットに登録すると，世界中の Android ユーザからインストールが可能となる．現在 Android マーケットでは，このような大きなビジネスチャンスを提供されているため，毎年多くのアプリケーションが登録されており，アプリケーション市場は賑わっている．

Android マーケットの存在により，ユーザから見てもアプリケーションの入手は容易である．Dalvik 実行形式のバイトコードの状態で配布されているため，必要なアプリケーションをインストールして，スマートフォンを自由にカスタマイズできる．広告から収益を得ることによりアプリケーション自体は無償で提供されているものも多く，気軽にインストールして利用できる．

本研究はこれらのサービスを提供するシステムプラットフォームとしての Android に焦点を当て，通信システムの高速化を目指しているが，このように Android 端末においてアプリケーションの存在を無視することはできない．そこで本研究ではアプリケーションからの無線通信利用を前提として，通信スループットの高速化を目指す．

3. 輻輳制御の既存研究

輻輳とは帯域が混み合うことにより，パケット損失率が高くなる状態である．損失率が高いとパケットを再送する必要があるので，通信スループットは低下してしまう．輻輳制御とは，データ送信側が帯域の混雑具合を予測し，輻輳を未然に防ぐための制御である．輻輳ウィンドウという，転送先からの確認応答を受信することなく，一度に連続して送り出せる最大のセグメント数を示すパラメータの大きさを調節することで輻輳が起きないように制御している．

特に高遅延環境において，通信スループットへの影響が大きく，輻輳ウィンドウが大きくなれば，連続して送り出せるセグ

メント数も多くなり，通信スループットは高くなる．つまり，より賢い制御を行い輻輳が起きない範囲で輻輳ウィンドウを大きく保つことで，通信スループットは向上する．

これまでも輻輳制御アルゴリズムに関する多くの研究がなされているが，それらの大部分がシミュレータを使った理論的な測定であり，特にモバイル端末において実測値を得た研究は少ない．輻輳ウィンドウはカーネル内部のパラメータであるが，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 TCP の振舞を知る事は困難である．そこで本研究では，汎用 PC 用のカーネルモニタを Android に組み込むことで，実機の輻輳ウィンドウの遷移を観察した [5].

カーネルモニタとは，TCP の処理が行われるごとに各パラメータの値をログとして残すツールであり，TCP のどの部分のコードがいつ実行されているかを明らかにすることができる．図 3 に示すように，TCP のソースコードにモニタ関数を挿入し，カーネルを再構築することで，メモリからログを得ることが可能となる．カーネルモニタは輻輳ウィンドウの他にも，タイムスタンプ，ソケットバッファキュー長などのパラメータや各種エラーイベントの発生タイミングも取得することができる．

カーネルモニタを用いて出力したログを解析することにより汎用 PC においては実験を行うことが可能であるが，Android 端末は CPU パワー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困難であった．カーネルモニタはパラメータが切り替わるごとにログを残すため，解析には大量のメモリを消費する．実際に通信中に解析処理を並列で行ったところ，通信システムの動作を阻害してしまうことを確認した．そこで，通信処理と解析処理を並列で実行するために，カーネルモニタが最新のログのみを出力するように対象となるソースコードの出力処理を書き換えて，Android に組み込んだ．さらに本研究では，カーネルモニタによって得られた

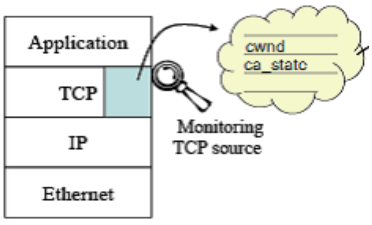

図 3　カーネルモニタ

ログを解析し，輻輳ウィンドウの値をブロードキャストするミドルウェアで，発信されたデータを受信し，時系列にそってグラフに描画するアプリケーションを開発した．これは著者らの開発した Android の通信システム可視化アプリケーションを拡張したものである [8]．このカーネルモニタによって得られた，実際の輻輳ウィンドウの遷移を図 4，5 に示す．グラフの縦軸は輻輳ウィンドウ，横軸は時間 (s) である．

3.1　現行の輻輳制御アルゴリズム

これまでに実装された輻輳制御アルゴリズムは，遅延ベース方式，損失ベース方式，及びそれらのハイブリットに分けることができる．遅延ベース方式とは，データ転送中に計測された RTT の実測値と理論値を比較し，CWND の大きさを調節する制御手法である．遅延に基づく正確な制御であるが，損失ベース方式との競合によるスループットの低下を招くというデメリットがある．損失ベース方式は，正常な確認応答を受信したら輻輳ウィンドウを増加させ，エラーを検出すると減少させる制御である．有線通信においては，通信エラーは一般に経路の混雑によるルータのバッファ溢れを示すため，この手法はシンプルかつ効果的であるが，無線通信においては，ノイズによる通信不良も帯域の混雑とみなしてしまうため，特に未使用帯域を大きく余らせてしまう可能性が高くなる．

Android のデフォルト輻輳制御アルゴリズムは，損失ベース方式を採る TCP-CUBIC である．CUBIC とは BIC の派生アルゴリズムであり，BIC アルゴリズムと同様の制御を三次関数状で行う．図 4 は実際に HT-03A から得られた TCP-CUBIC の輻輳ウィンドウの遷移である．

3.2　独自の輻輳制御アルゴリズム

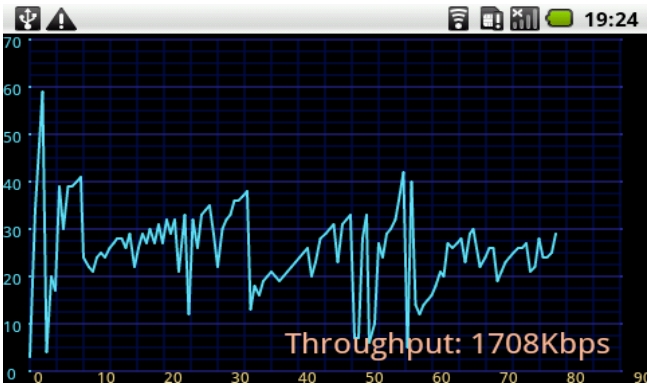

図 4　TCP-CUBIC の輻輳制御

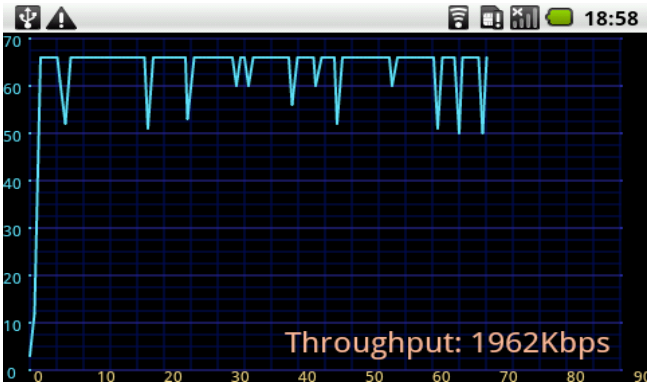

図 5　独自 TCP の輻輳制御

独自の TCP 輻輳制御アルゴリズム [6] は，デフォルトの TCP-CUBIC が無線通信において不必要に輻輳ウィンドウを低下させている場合があるという点に着目し，輻輳ウィンドウを下がりにくく上がりやすい，強気な通信をするように改良したものである．図 5 は，独自のアルゴリズムによる輻輳ウィンドウの遷

移である．このアルゴリズムを利用した TCP 通信は RTT=32ms 以上の高遅延環境において，デフォルトの場合より性能が良いことが確認されている．本ミドルウェアでは切り替え用の TCP としてこの独自の輻輳制御アルゴリズムを利用する．また，ミドルウェアからの指示で輻輳ウィンドウの最大値を抑えることを可能にしたため，今後は環境に応じたフレキシブルな適応制御を実装する事が可能である．本研究では競合時の通信スループット向上とともに，多端末共有時の通信公平性についても注目していく．

図 4，5 では，TCP の切り替えは行っておらず，単一の TCP によって得られた遷移である．どちらも往復遅延時間 128ms，人工パケットロス率 1%の環境で観測された結果である．

4. 通信制御ミドルウェア開発

4.1 HT-03A の通信スループット測定

本実験で使用した実験環境を表 1 に示す．図 6 に示すように，アクセスポイントとサーバの間に人工遅延装置 FreeBSD の dummynet を設置し，経路上に人工遅延と人工パケット損失を加えた．

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

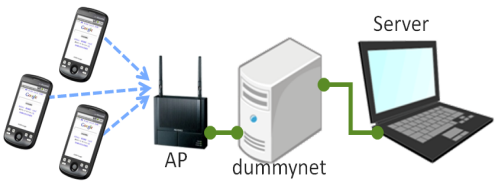

図 6　測 定 方 法

4.2 現行の TCP 性能

HT-03A で各端末数ごとに通信実験を行った実験結果を図 7〜9 に示す．これらは Iperf を利用し，各端末が 20 秒間の TCP 通信を行った時の通信スループットである．

実線は各端末の通信スループットの平均値，点線は各端末の通信スループットの総和であるトータルスループットを示す．理論的には，往復遅延時間と人工パケットロス率，同時に通信する端末数が増加するに従って，各端末の通信スループットは低下していくと考えられる．

RTT=128ms 以上の高遅延環境においては，端末数に関わらず 1 台あたりの通信スループットはほぼ一定となっており，端末数が増えるごとにトータルスループットは増している．この結果から RTT=128ms 以上の環境においては端末数各端末の転送量は Android の TCP-CUBIC の限界によって抑えられているため，帯域のキャパシティを超えていないことがわかる．

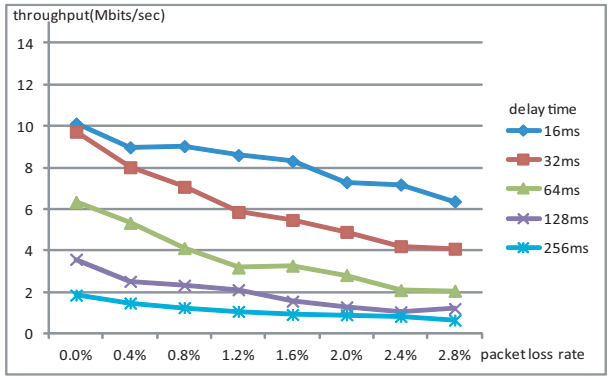

図 7　測定 1: 1 台同時通信時の通信スループット

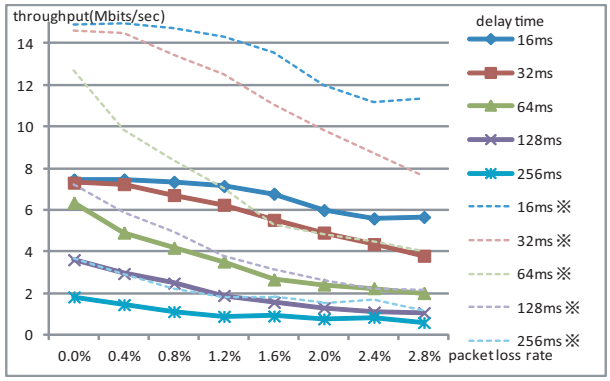

図 8　測定 2: 2 台同時通信時の通信スループット

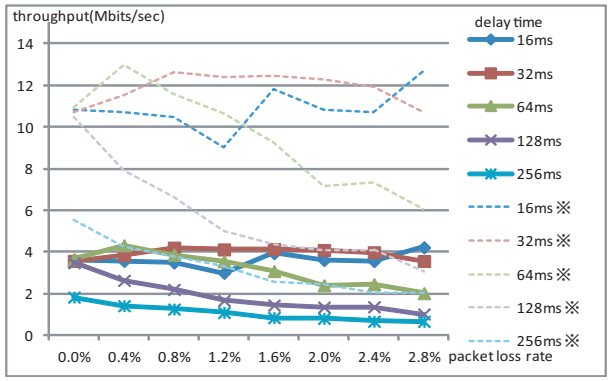

図 9　測定 3: 3 台同時通信時の通信スループット

次に RTT=64ms 以下の低遅延環境を見ていく．測定 2 と測定 3 を比較すると，パケットロス率が 1.2%以下の時は 2 台同時通信の方がトータルスループットが高く，1.6%以上の時は 3 台同時通信の方がトータルスループットが高いことがわかる．TCP-CUBIC が限界を迎えると，端末数が増えるに従いトータルスループットがさらに高くなるが，測定 3 の低遅延低パケットロス環境ではその逆の結果であるため，TCP-CUBIC が飽和していないにも関わらず，実環境で使用可能な帯域を使い切れていないため，パケットの衝突により輻輳ウィンドウを下げ過ぎたと考えられる．

4.3 通信制御ミドルウェアの目的

本研究では未使用帯域を使いきれる通信システムの確立を目指す．独自の TCP は輻輳ウィンドウを上がりやすく，下がりにくい制御アルゴリズムを実現したが，このような強気な TCP を常に利用することは他端末の通信を阻害する可能性がある．そこで本研究では適切なタイミングで TCP を環境に適したものに切り替えるミドルウェアを開発する．

3.1 章に示したように，遅延ベース方式は RTT によって，損失ベース方式はパケットロス率によって輻輳ウィンドウを調節しているが，本研究は，Android 端末が広帯域有線ネットワーク接続されたクラウドサーバと通信する場合を想定し，輻輳が懸念されるアクセスポイント・Android 端末間の無線帯域を共有している他端末の通信状況を考慮した制御を目指している．そこで本ミドルウェアは，同一アクセスポイントを共有する無線 LAN 空間内において，互いの端末の通信状況，すなわち輻輳ウィンドウを通知し合い，周囲の他端末の通信状況に応じて，輻輳制御アルゴリズムを切り替え適応させる．輻輳ウィンドウはカーネルモニタを利用して実際の通信中の値を取得している．

4.4 通信制御ミドルウェアの実装

本稿では，既存研究の TCP とデフォルトの TCP を切り替えるミドルウェアを紹介する．プログラム上から TCP を切り替える基礎実験は，Android アプリケーションで開発された通信テストツールに組み込まれたプロセスから適宜シェルスクリプトを実行していたが [10]，汎用性を高めるために TCP を切り替える部分をアプリケーションから切り離しミドルウェアとした．Android アプリケーションの Service を利用することもできるが，通信テストツールと併用した場合，大きな負荷となり，システムに障害を齎すため，Android の C ライブラリである bionic を利用したネイティブコードに移植した．ネイティブコードのクロスコンパイラとして，arm-2008q3 [4] を使用した．

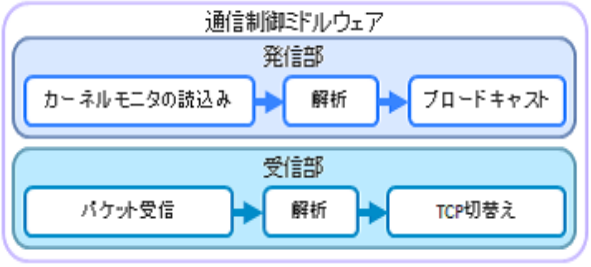

図 10　ミドルウェアの構成

ミドルウェアは図 10 に示すように，発信部と受信部に分かれて，独立した 2 つの実行ファイルとして機能する．発信部は常駐でカーネルモニタのログを監視し，ログがあれば解析して輻輳ウィンドウの値を UDP パケットで，周辺端末へブロードキャストする．受信部はこの UDP パケットを受信し，その UDP パケットが自分の IP アドレスと異なる送信元から転送されたものであり，一定値を超えた輻輳ウィンドウであれば，それらの個々の値と端末数から状況を判断し，最適なアルゴリズムと最大値を設定する．

5. ミドルウェアを導入した通信実験

5.1 1. HT-03A 同士の通信制御

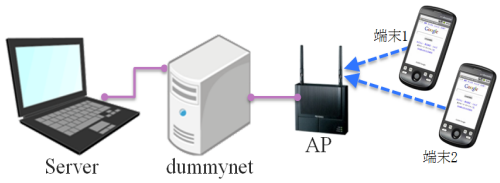

図 11　ミドルウェアの制御を確認する実験モデル 1

本実験では図 11 に示すように，1 台のアクセスポイントを 2 台の Android 端末で共有する．Android 端末を送信側とし，アクセスポイントと人工遅延装置を経由してサーバに対してパケット転送を行う．端末 1 が 50 秒間の通信を行っている間に，端末 2 が途中から加わり 20 秒間のパケット転送を行う．パケット転送は今回は Iperf を利用したが，今後は Dalvik 上の通信テストを行い，アプリケーションから高速通信ができることを確認する方針である．TCP の切替えを確認するために 2 つの実験を行った．実験 1 では Android 端末の TCP がデフォルトの状態で切り替えは行わない．実験 2 では，30 以上の CWND 値を受信すると TCP を CUBIC に切り替え，30 以上の CWND 値を 5 秒以上受信しなかったら独自の TCP に切り替えるようにした．実験 1，実験 2 では共に人工パケットロス率 1.0%，往復遅延時間 64ms を加えて実験をおこなった．

5.2 実験結果と評価

端末 1 はその 5 秒後に帯域を独占したと判断し，独自の TCP に切り替えた．

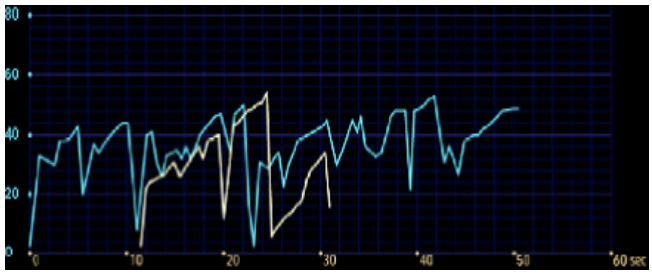

図 12　実験 1: デフォルトの処理

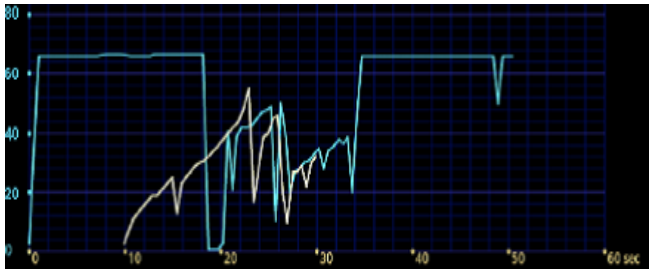

図 13　実験 2: ミドルウェアを導入した処理

TCP の振舞は自然環境の影響を受けやすく毎回同じ結果を得られるわけではないが，これら実験で得た典型的な振舞を図 12，図 13 に示す．グラフは 2 台の Android 端末で通信実験を

行った際の各端末の輻輳ウィンドウの時系列変化である．各端末のミドルウェア発信部からブロードキャストされた輻輳ウィンドウの通知パケットを受信し，時系列に従って可視化を行った．縦軸は輻輳ウィンドウ値，横軸は時間 (s) を示す．

実験 1 では，TCP の切り替えはなく，常にデフォルトの TCP を利用して通信を行った．実験 2 では，初期状態を独自の TCP とし，端末 1 の通信開始から 10 秒後に端末 2 が通信を開始した．端末 2 の輻輳ウィンドウ値が 19 秒後に 30 を上回り，端末 1 はこの通知を受けて，デフォルトの TCP に切り替えた．また端末 1 の開始から 30 秒後に端末 2 の通信が完了し通知を終了した．

5.3 2. Nexus S と HT-03A の混在環境における通信制御

本手法では，HT-03A と Nexus S の混在環境において HT-03A がハイスペックかつ積極性の強い輻輳制御アルゴリズムを持つ Nexus S に帯域を取られてしまい，公平性が保てなくなってしまうというデメリットに対し，Nexus S に HT-03A のアルゴリズムを移植し，最大値を制限することで，HT-03A にある程度帯域を譲るように設計された制御を利用している．

Android 実機における TCP の切り替えを確認するため，本ミドルウェアを導入した Nexus S と HT-03A を用いて，RTT=8ms における通信実験を行った．本実験モデルを図 14 に示す．本実験における輻輳ウィンドウの最大値の最適値及び最適なアルゴリズムは [7] で示された表 2 TCP 制御方法に従う．

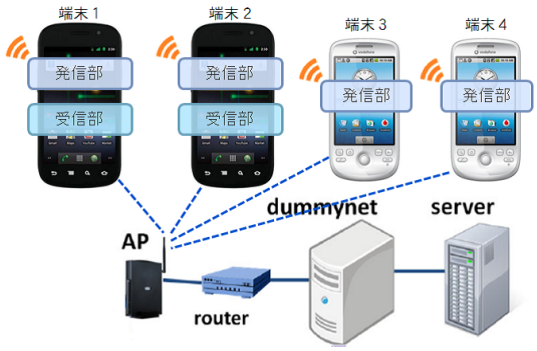

図 14　ミドルウェアの制御を確認する実験モデル 2

表 2　TCP 制御方法

RTT	Nexus S		HT03-A	
	TCP	上限値	TCP	上限値
0	default	20	original	70
1	default	20	original	70
2	default	20	original	70
4	default	20	original	70
8	default	20	original	70
16	default	20	original	100
32	default	30	original	100
64	default	55	original	100
128	default	55	original	100
256	default	100	original	100

HT-03A は常時独自の TCP を用い，TCP の切り替えを行わないため，HT-03A ではミドルウェアの発信部のみを実行した．発信部からは通信中の輻輳ウィンドウ値が通知パケットとして発信され，Nexus S 上のミドルウェア受信部に受信される．Nexus S のミドルウェア受信部は，通知を受けて，適切な輻輳ウィンドウ最大値とアルゴリズムの選択を行う．1 台の Android 端末が 50 秒間の通信を始め 10 秒が経過した時点で，残りの 3 台の Android 端末が 30 秒間の通信を開始する．3 台の Android 端末が通信を終えると 50 秒間の通信を行っている Android 端末は，1 台で帯域を独占して残りの 10 秒間の通信を行うという実験を各条件下で行った．実験 3，4 では Nexus S の端末 1 が 50 秒間通信を行い，端末 2〜4 が後から割り込むように通信を開始する．実験 5，6 では HT-03A の端末 3 が先に 50 秒間通信を開始し．端末 1，2，4 が後から加わる．実験 3，5 ではミドルウェアの TCP 切り替え機能を適用せず，対象実験としてデフォルトの処理を行った．実験 4，6 ではミドルウェアを適用し，TCP を切り替える処理を行った．

5.4 実験結果と考察

これらの実験結果を図 15〜図 18 に示す．これらのグラフは 4 台の Android 端末で通信実験を行った際の各端末の輻輳ウィンドウの時系列変化である．実験 3，4 においては，水色と黄色のグラフが Nexus S の輻輳ウィンドウの振舞，桃色と緑色のグラフが HT-03A の振舞を示す．実験 3 では，2 台の Nexus S が輻輳ウィンドウを大きく増加させ，一方で HT-03A はなかなか輻輳ウィンドウを上げられず控え目の制御となっている．同じ帯域を共有していても，Nexus S は HT-03A に比べて異常状態に陥りにくいため，Nexus S は強気な通信を続けた結果，HT-03A の異常状態が増え，輻輳ウィンドウを増加させにくくなったと考えられる．実験 2 では，Nexus S のミドルウェアが他端末の存在を確認した際に，輻輳ウィンドウの上限値を抑えているため，HT-03A は残された帯域を積極的に利用するために，実験 1 よりも輻輳ウィンドウを増加させている．また Nexus S においても他端末が通信を終えたと判断した際に，TCP をデフォルトに戻しているため，帯域独占時には輻輳ウィンドウを増加できているため，帯域を無駄にしていない．

実験 5，6 においては水色と黄色のグラフが HT-03A の輻輳ウィンドウの振舞，桃色と緑色のグラフが Nexus S の振舞を示す．実験 5 では，Nexus S にミドルウェアが機能していないことから強気な通信で帯域を確保してしまうため，HT-03A が先に通信を開始していても，輻輳ウィンドウが増加しにくくなっていることがわかる．実験 6 で割り込んで通信を行う 2 台の Nexus S は，既に他端末の存在を確認し，上限値を下げた HT-03A の TCP を選択した状態で通信を開始するため，最初から最後まで，一定の輻輳ウィンドウ値を保っている．この時 HT-03A は，Nexus S の輻輳ウィンドウが制限されているため，自分の輻輳ウィンドウを増加させやすくなっていることがわかる．

このように通信制御ミドルウェアを導入したことにより，デフォルトの TCP の処理よりも安定した輻輳制御処理が可能となった．

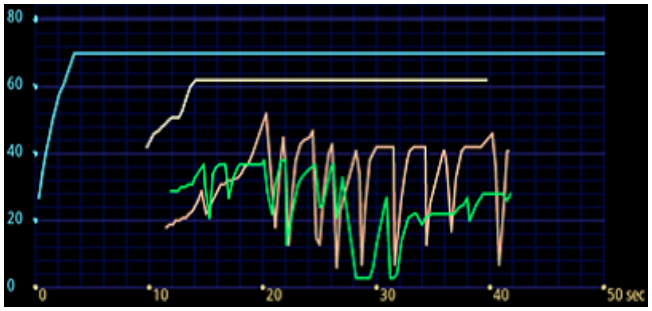

図 15　実験 3: TCP 切り替え機能を導入していない Nexus S を 50 秒間通信させた際の輻輳ウィンドウ時系列変化

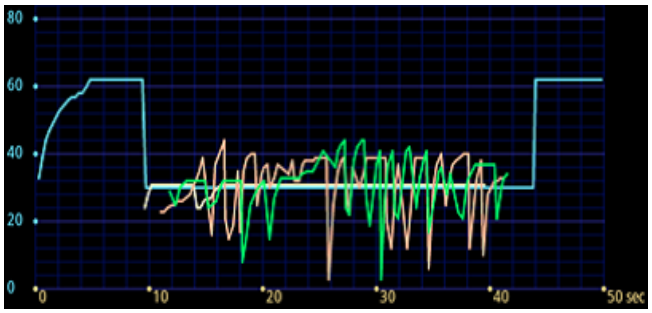

図 16　実験 4: TCP 切り替え機能を導入した Nexus S を 50 秒間通信させた際の輻輳ウィンドウ時系列変化

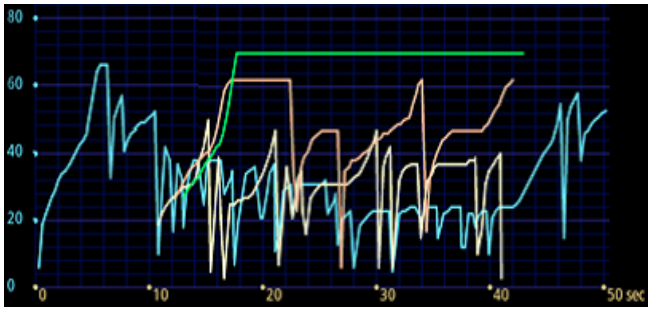

図 17　実験 5: TCP 切り替え機能を導入していない Nexus S を用い，HT-03A を 50 秒間通信させた際の輻輳ウィンドウ時系列変化

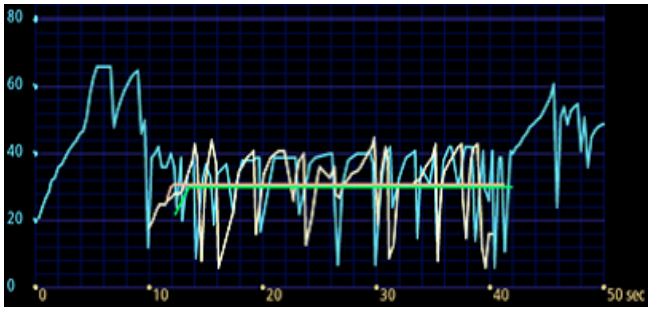

図 18　実験 6: TCP 切り替え機能を導入した Nexus S を用い，HT-03A を 50 秒間通信させた際の輻輳ウィンドウ時系列変化

6.　まとめと今後の課題

本ミドルウェアは，TCP の通信処理中に周囲の環境に合わせて自動的に TCP を切り替えることに成功した．実験 1，2 では，帯域を余らせている時は，強気の TCP に切り替え，混み合った時に default に切り替えることで，効率良く通信を制御した．実験 3〜6 では，混在時に適切な TCP と上限値を設定することで，環境に適応して公平性を確保した．

今後は他端末の輻輳ウィンドウ値に応じて TCP の上限値をフレキシブルに変動させる制御を開発する．独自の輻輳制御アルゴリズムは輻輳ウィンドウを増加させやすく，減少させにくいが，一定値を上回らないことで，他端末の通信を阻害することなく，存在している帯域を限りなく使い切ることで，公平性を確保しながら，トータルスループットの向上を目指す．

謝　　辞

本研究は一部，独立行政法人情報通信研究機構の委託研究「新世代ネットワークを支えるネットワーク仮想化基盤技術の研究開発・課題ウ 新世代ネットワークアプリケーションの研究開発」によるものである．

また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，株式会社 NEC Technologies の Iain Williams さんに深く感謝致します．

文　　献

[1] W.Richard Stevens 著，橘康雄，井上尚司訳，詳解 TCP/IP Vol.1 プロトコル，ピアソン・エデュケーション，東京，2000.
[2] android developers:http://developer.android.com
[3] Iperf:http://downloads.sourceforge.net/project/iperf/iperf/2.0.4
[4] Sourcery G++ Lite 2008q-3-72 for ARM GNU/Linux:http://www.codesourcery.com/, http://www.codesourcery.com/sgpp/lite/arm /portal/release644
[5] 三木香央理，山口実靖，小口正人:Android 端末におけるカーネルモニタの導入，Comsys2010，2010 年 11 月.
[6] 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 通信性能の解析と性能向上のための一検討，DICOMO2011，7H-2，2011 年 7 月.
[7] 三木香央理，山口実靖，小口正人: 無線 LAN 通信環境におけるカーネルモニタを用いた TCP 解析による Android 端末の性能向上手法，DEIM2012，C6，2012 年 3 月.
[8] 平井弘実，三木香央理，山口実靖，小口正人: Android 端末における通信性能の可視化ツール，情報処理学会第 73 回全国大会，5V-9，2011 年 3 月.
[9] 平井弘実，三木香央理，山口実靖，小口正人: 無線環境下における Android 端末の通信制御ミドルウェア構築に向けた一検討，DICOMO2011，7H-3，2011 年 7 月
[10] 平井弘実，三木香央理，山口実靖，小口正人: Android 端末を用いた無線通信時の制御ミドルウェアに関する考察，電子情報通信学会 NS 研究会，NS2011-105，2011 年 11 月