情報処理学会論文誌　コンピューティングシステム　Vol.7　No.1　23–34 (Mar. 2014)

Android OSにおける状態変化通知による通信集中の削減手法

川崎 仁嗣[1,a)]　神山 剛[1]　小西 哲平[1]　大久保 信三[1]　太田 賢[1]　稲村 浩[1]

受付日 2013年7月25日, 採録日 2013年12月21日

概要：Android OSでは，ユーザ操作や通信状況の変化，アプリの動作にともない発生した端末の状態変化を他のアプリへ通知する仕組みが提供されている．あるアプリの挙動による状態変化にともなって他のアプリが動作するため，アプリの挙動によっては開発者の意図していなかったタイミングで通信が発生してしまうことがある．このため，電車がトンネル区間を抜けたときなどネットワーク接続状態の変化が複数の端末でいっせいに起こった場合，意図しない通信により通信集中が引き起こされることがある．本研究では，複数の端末で同一タイミングに発生する状態変化の場合，通知タイミングを遅延させることで通信集中を削減する制御手法を提案する．実端末を用いた評価において，複数端末で同期したタイミングでのアプリ動作が抑制され，通信タイミングが端末間で分散することを確認した．

キーワード：Android，状態変化，通知，大域同期，通信集中，輻輳

A Method for Reducing the Congestion Caused by the Notification Behavior of Android OS

Satoshi Kawasaki[1,a)]　Takeshi Kamiyama[1]　Teppei Konishi[1]　Shinzo Ohkubo[1]　Ken Ohta[1]
Hiroshi Inamura[1]

Received: July 25, 2013, Accepted: December 21, 2013

Abstract: Android OS has a mechanism that raises an event notification to indicate a change in state caused by user's operation, other tasks' behavior, or a change in connectivity. The last is of significant interest since, for example, all devices on a train that exits a tunnel will experience the triggering of tasks to receive the event. Therefore, traffic bursts may occur unintentionally. This paper proposes a congestion avoidance method that offsets notification timing to distribute among the devices. An evaluation confirms that our method staggers unintended invocations which suppresses traffic bursts.

Keywords: Android, state change, notification, global synchronization, concentration of network traffic, congestion

1. はじめに

昨今のモバイルインターネット利用環境は，今までのフィーチャーフォンからAndroid OS [1] やiOS [2]，Windows Phone [3] を搭載したスマートフォンに移行しつつある．スマートフォンでは様々な開発者が作成したアプリケーション（以降ではアプリと呼ぶ）をインストールして利用することができるため，通信の発生するタイミングや頻度はそれぞれのアプリ開発者が指定したものに従うこととなる．インストールされるアプリの種類が変われば，その通信量や頻度も変化するだけでなく，アプリの普及度合いや仕様も短期間で変化することがあるため，通信事業者としては安定したネットワークサービスを提供するために，各アプリの挙動やユーザの利用動向を把握する必要性が高まってきた [4], [5], [6], [7].

[1] 株式会社NTTドコモ
NTT DOCOMO, INC., Yokosuka, Kanagawa 239–8536, Japan
[a)] kawasakisa@nttdocomo.co.jp

一方でアプリ開発者の観点で考えると，スマートフォン向けの OS ではアプリを起動していない状態や画面オフの状態など，ユーザが操作を行っていないタイミングにおいても，データをサーバと同期させるために定期的にアプリをバックグラウンドで動作させて，通信などの処理を実行させることができる．バックグラウンドでのアプリ動作タイミングを OS 側で制御している iOS [8] や Windows Phone [9] と比べると，Android OS は動作タイミングをアプリ開発者が自由に指定することが可能である [10], [12]．しかし，それゆえに注意深く実装を行わないと特定のタイミングに偏ってアプリが動作する場合があり，これによってプロセッサへの処理が集中したり，限られた通信リソースへのバースト的なアクセスが発生したりしかねない．特に，複数の端末において同一のタイミングで通信を発生する挙動となっていると，ネットワーク側での一時的な通信集中が発生する原因となりうる．通信集中による問題としては，データ自体の通信量が設計容量を超えてしまい通信速度が極端に低下してしまうことと，通信を開始するためにやりとりされる制御信号が設計容量を超えてしまい，通信用のセッションを確立できないことがあげられる．この状況をふまえ，定期的な通信処理を行うアプリが発生させるトラフィックについての分析 [11] が行われるようになってきた．

バックグラウンドでのアプリ動作タイミングが偏る要因の 1 つとして，Android OS では，あるアプリの挙動に影響を受けて他のアプリが動作を開始する挙動（以降ではカスケード起動と呼ぶ）が発生しており，これによりアプリ開発者の意図していないアプリ起動が発生する場合があり，動作タイミングが集中しないよう考慮した実装を行うことが難しい点があげられる．たとえば，目覚ましアプリが AM7:00 に鳴動しディスプレイが画面オン状態へと遷移する挙動を想定すると，画面オン状態へと遷移したことで端末状態の変化を示す通知が配送され，この通知を受け取ったアプリがカスケード起動することにより動作タイミングが集中する事例が考えられる．我々が実施したユーザ調査（調査条件などは 5 章を参照）の結果，AM7:00 にアラームを設定しているユーザが約 5%存在していることが分かった．また，画面オン状態へ遷移した際に通信を発生するアプリがインストールされている端末の割合は約 50%であった．今後もスマートフォンへの移行が進むことを考えると，無視できない割合であり潜在的な問題であると考えられる．その他の事例としては，電車がトンネル内などの圏外エリアから圏内エリアへと移動することにより，乗客の持つ端末が圏内状態へと復帰し，この状態変化による通知を受けたアプリが複数の端末でいっせいにカスケード起動する例も考えられる．

本論文では，2 章で状態変化の通知によって発生するカスケード起動が通信集中を発生する要因の分析を行い，3 章で関連研究について考察し，4 章でアプリ開発者が輻輳回避のための処理を行わなくても通信集中が発生しないよう，OS 側でカスケード起動によるアプリの動作タイミングを制御する手法について検討する．5 章で本提案手法を実装した端末を用いて通信タイミングが分散することで通信集中が削減されていることを実験環境において検証する．最後に 6 章で本論文をまとめる．

2. カスケード起動による通信集中

本章では，カスケード起動により通信集中が発生する挙動を明らかにするため，大域同期起動と同期干渉という概念を定義し，定時実行動作による大域同期起動と状態変化通知動作による大域同期起動を述べる．大域同期起動と同期干渉の関係を**図 1** に示す．

2.1 大域同期起動

複数の端末間でアプリの動作タイミングが揃っている状態，つまり大域同期して起動する状態を大域同期起動と呼ぶ．図 1 の例では，アプリ A が端末 A と端末 B の双方において 7:00 に動作するよう設定されている．そのため，時刻が 7:00 になった時点で双方の端末でアプリ A が起動することとなり，大域同期起動している．大域同期起動したアプリが通信を発生する場合，複数の端末からいっせいに通信が発生することとなり，ネットワーク側での輻輳要因となる．

したがって，大域同期起動するアプリでは通信をともなわない処理を行うか，通信処理のタイミングが端末間で分散される実装とすべきである．一方で，特に同期した動作を意図していないアプリを開発する際には，動作タイミングが元々，端末間で分散していることから，通信処理のタイミングを分散化する必要性はないように思える．しかし，端末上で大域同期起動するアプリが存在する状態でカスケード起動による同期干渉が発生することにより，アプリ開発者の意図に反して大域同期したタイミングで動作する場合がある．

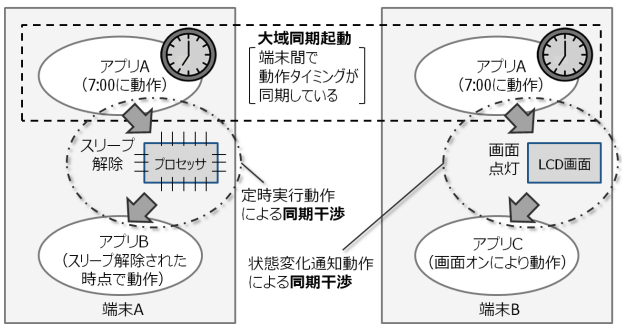


図 1　大域同期起動と同期干渉の関係

Fig. 1　Relations of global synchronized invocations and sync interference.

表 1　同期干渉の関係
Table 1　Relations of sync interference.

	定時実行動作	状態変化通知動作
干渉源アプリ	wakeup 型指定の Alarm を利用（自律型）	特定の状態変化に応じた Broadcast Intent を受信
カスケード起動アプリ	non-wakeup 型指定の Alarm を利用（他律型）	状態変化に応じた Broadcast Intent を受信

2.2　同期干渉

あるタイミングに同期した動作を意図してはいないアプリであっても，同一端末上で動作している他のアプリからカスケード起動されることにより，双方のアプリの動作タイミングが同期する事象を，我々は同期干渉 [18] と呼ぶ．アプリ自身が設定した特定のタイミングで動作する自律型アプリや特定の状態変化を受け取って動作するアプリで，動作中に状態変化を発生するアプリが干渉源となる．また，特定のタイミングで動作するのではなく，状態変化通知などのアプリ外で発生した契機で受動的に起動されるアプリや，動作タイミングがシステムに委ねられており任意の同時実行が期待される他律型アプリが干渉を受けカスケード起動される（以降ではカスケード起動アプリと呼ぶ）．干渉源アプリとカスケード起動アプリが同一端末上で動作しているとき，同期干渉によって動作タイミングが同期する．

同期干渉の発生要因としては，大きく分けて以下の 2 つが存在する．

(1) 定時実行動作
　アプリをあらかじめ設定した時刻において起動させる挙動．Android OS においては Alarm Manager [12] と呼ばれる機構が定時実行動作を実現している．図 1 の例では，アプリ A が動作時刻になることでプロセッサのスリープ状態が解除され，非スリープ時に動作するアプリ B が動作しており，同期干渉が発生している．

(2) 状態変化通知動作
　端末の状態に変化が発生したことをアプリ側に通知する挙動．Android OS においては状態変化を検知すると Broadcast Intent [13] を配送することで通知を行う仕組みが提供されている．図 1 の例では，アプリ A の動作により画面が点灯することで，画面点灯を契機に動作するアプリ C が動作しており，同期干渉が発生している．

同期干渉が発生する干渉源アプリとカスケード起動されるアプリ（被干渉アプリ）との関係は**表 1** のとおりであり，詳細な挙動について次節以降で述べる．

2.3　定時実行動作による大域同期起動

定時実行動作は，アプリが事前に指定した時刻において処理を実行する挙動である．スマートフォン端末では一般的にバッテリ容量が限られていることから，スマートフォン向け OS では電力消費を抑えるために指定した時刻でただちに処理を実行するのではなく，電力効率の良くなるタイミングで処理を行うように実装されている場合が多い．スマートフォン端末ではつねに CPU などのデバイスが動作しているのではなく，たとえば画面オフ状態においては必要がなければ各デバイスをスリープ状態へと遷移させ，消費電力を削減するようになっている．したがって，電力効率を考慮すると，できるだけスリープ状態を維持し，非スリープ状態（CPU が動作している状態）となる頻度や期間をおさえることが重要となる．

Android OS においては，指定した時刻において端末がスリープ状態であっても強制的に非スリープ状態へ遷移させてから処理を開始するよう指定する方法（以降では wakeup 型指定と呼ぶ）があり，目覚まし時計などの特定時刻で必ず動作する必要があるアプリで用いられる．一方で，指定時刻を経過して次に端末が非スリープ状態となった時点で動作を開始するよう指定する方法（以降では non-wakeup 型指定と呼ぶ）も存在し，実行タイミングが厳密に特定の時刻でなくてもよい処理は non-wakeup 型指定で実装することにより，他の処理と重畳して実行されるため結果として非スリープ状態となる頻度を抑えることができる．

non-wakeup 型指定の処理は，ユーザによる画面オン操作により非スリープ状態となったタイミングや，wakeup 型指定を用いている別のアプリにより非スリープ状態へ遷移したタイミングにおいて実行される．この挙動により，他律型アプリの動作タイミングが自律型アプリの動作タイミングに同期することとなり同期干渉が発生しうる．また，自律型アプリの動作タイミングが複数の端末間で同一である場合，同期干渉を受けた他律型アプリも大域同期起動する状態となる．

たとえば，目覚まし時計アプリ（自律型アプリ）が AM7:00 に動作するよう設定され，同様に AM7:00 に設定されている端末が複数存在すれば，他律型アプリは他の端末と同期した動作を本来は意図していないにもかかわらず，実際には他の端末と同期したタイミング，この場合は AM7:00 に動作することとなる．

2.4　状態変化通知動作による大域同期起動

状態変化通知動作は，端末の状態に変化が発生したことをアプリ側に通知する挙動である．状態変化通知動作による大域同期起動が発生する過程を理解するにあたって，まずは状態変化通知の概要を述べる．

2.4.1　状態変化通知

スマートフォンにおいては，端末の状態などが変化したタイミングで動作するアプリが多い．たとえば，電話アプリは音声通話の着信が発生した際に，画面を点灯させて着信中の画面を表示するとともに着信音を再生する．このようなアプリを実装するために，端末の状態に変化が発生し

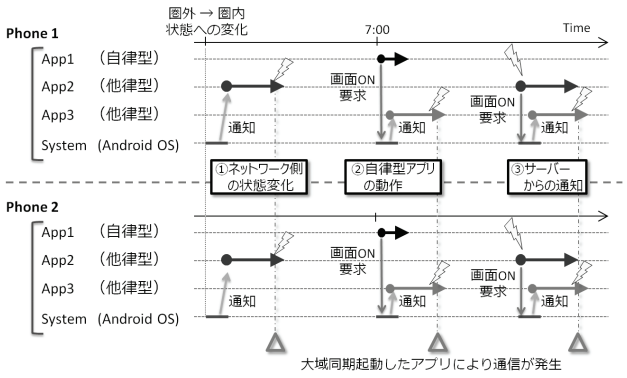


図 2　状態変化通知動作による大域同期起動
Fig. 2　Global synchronization caused by an event delivery mechanism.

たことをアプリ側に通知する機能が OS 側で提供されていることが多い．この通知機能を実現する仕組みを状態変化通知機構と呼ぶ．

Android OS においては，Broadcast Intent を配送することで状態変化の通知が実現されている．アプリは受け取りたい通知を事前にシステム側へ登録する必要があり，登録を行ったアプリのみが通知対象となる．たとえば，電話アプリは電話の着信に関する通知は受け取るがメールの着信については扱わないため，電話着信の通知のみを受け取るよう指定する．

状態変化は，その発生要因に応じて以下の 2 種類に大別できる．

(1) 内部要因状態変化
　バッテリ残量の変化や，他アプリの実行による画面の点灯，および音楽再生の開始など，端末内での要因により発生する状態変化

(2) 外部要因状態変化
　ユーザ操作による画面の点灯や，ネットワーク側からの電話やメール着信，および電波状態の変化など，外部からの要因により発生する状態変化

2.4.2　大域同期起動の要因となる状態変化

状態変化のうち内部要因状態変化はアプリの実行によっても引き起こされることがあり，あるアプリが動作したタイミングにおいてアプリ挙動により状態変化が発生すると，その状態変化の通知を受け取ったアプリも動作を開始する．つまり，カスケード起動が発生する．これは本来想定される動作であり，これだけで問題となるものではないが，**図 2** に示すように，状態変化を引き起こすアプリが大域同期起動する状態となっている場合，通知を受け取り動作する他律型アプリに同期干渉が発生し，他律型アプリも大域同期起動することとなり通信集中の原因となる．サーバからの通知による大域同期とは，同報的な配信が行われた場合が該当し，たとえば緊急情報の配信など，速報性が求められるサービスの利用が考えられる．このようなサービスでは即時性が求められるため，サーバ側で通知タイミングを分散させるなどの処理を行うことが難しい．そのほかにも，大量のフォロワが存在するユーザによる SNS アプリでのメッセージ配信も大域同期を引き起こす可能性が想定される．一般的に SNS サービスでは固定回線に接続された PC からのアクセスも想定しており，更新通知は即座に行われることが多い．したがって，モバイルネットワークからのアクセスを前提とする SNS アプリでは，通知タイミングが大量の端末で同期しないよう考慮した実装となっていることがほとんどである．ところが，一部の SNS アプリなどでは開発者が十分に大域同期の可能性を考慮しきれていないこともあり，大域同期した通知を発生しうる挙動を示すことがある．また，外部要因状態変化であっても，複数の端末に対して同時に変化が発生する場合は同様に大域同期起動を引き起こし通信集中の原因となる．

表 2　同期干渉の発生率
Table 2　Occurrence frequency of sync interference.

	同期干渉発生率	大域同期参加率
ネットワーク側の状態変化	84%	12%
自律型アプリの動作	48%	8%
サーバからの通知	48%	3%

2.5　同期干渉の発生しやすさ

大域同期起動による同期干渉が発生するには，同一の端末上に干渉源アプリとカスケード起動アプリの両方がインストールされていることや，干渉源アプリが大域同期したタイミングで動作することなどが条件となる．そのため，すべての端末において発生しうるとは限らないが，多くのユーザにインストールされているアプリが干渉源アプリやカスケード起動アプリの挙動を示す場合は影響が大きい．

同期干渉の発生しやすさを確認するために，119 名のモニタユーザの協力を得てインストールされているアプリ数やアプリ名，アプリ起動履歴のログなどを 1 カ月分ほど収集した．こられのログを解析することで同期干渉の発生率や大域同期への参加率を調査した．同期干渉の発生率とは，全端末数に対してログの期間内に同期干渉が発生した端末数の割合のことであり，大域同期しているかどうかにかかわらず，同期干渉が発生する確率を表している．大域同期への参加率とは，全端末数に対して同期干渉が発生した際のタイミングが複数端末（2 台以上）間で同一となったことがある端末数の割合のことであり，同期干渉が発生した際に，その端末が大域同期している確率を表している．調査結果を**表 2** に示す．なお，ネットワーク側の状態変化とは，圏外状態と圏内状態との切替わりや，モバイルネットワーク接続から Wi-Fi 接続への切替わりが発生した状態変化を指す．また，自律型アプリの動作とは，目覚まし時計

アプリによる画面点灯の状態変化を指し，サーバからの通知とは，メッセージングアプリによる画面点灯の状態変化を指す．目覚まし時計アプリは朝の正時に設定されることが多く，6:00，6:30，7:00 のタイミングに動作が集中しており，大域同期起動した状態となっていた．

同期干渉の発生率は高いものの，大域同期への参加率はあまり高くはなかった．しかし，通常時に発生している通信に上乗せする形で同期干渉による通信が加わることとなり，さらに，現在のモバイルネットワークでは通信開始に先立ち通信セッションの確立を行うために多くの制御信号がやりとりされており，この通信が短期間に集中することはあまり考慮されていない．このような事情により，接続している基地局や交換機によっては設計容量を超えてしまう可能性があるため，大域同期している同期干渉を極力減らすことが重要となる．

3. 関連研究

本研究で対象とする，状態変化通知動作による同期干渉という概念に類する研究は筆者らの知る限り他にない．定時実行動作における同期干渉については小西らの研究 [18] で提案がある．ここでは，もう 1 つの主要な概念である，大域同期と輻輳の回避や起動の集中を扱っている関連研究について整理する．本研究で扱う課題は，大域的な知識なしに同期の発生を検知すること，同期の発生が想定される場合はその分散を図ることの 2 つである．

ネットワーク上での輻輳制御手法は古くから研究されてきており，Jacobson らの研究 [14] では，TCP 通信における輻輳を回避するために，BSD の TCP スタックに対して再送信のバックオフタイマの改善やスロースタートなどの修正を行っている．ネットワーク機器などにおいて，転送すべきパケット量が増加して処理可能な量を超えそうになった際，単純に後から来たパケットを破棄するテールドロップ方式では TCP における再送コネクションの大域同期が発生しやすい．RED（Random Early Detection）[15] や，RED の派生手法 [16], [17] ではルータのキューの長さなどによって輻輳の検知を行い，確率に基づいてパケットを破棄することにより TCP コネクションの大域同期を回避可能な輻輳制御手法を提案している．この研究で採用されている，輻輳に対する確率的な分散は本研究でも有用であると考える．

2.3 節で述べた定時実行動作による大域同期起動について，小西らの研究 [18] では，non-wakeup 型指定タスクの起動タイミングを，特定時刻に実行される wakeup 型指定タスクの起動タイミングとは重ならないように制御している．これにより，特定時刻に実行される wakeup 型指定タスクからカスケード起動されなくなるため，定時実行動作による同期干渉が発生しない．しかし，本研究で対象としている状態変化通知動作による同期干渉については扱われていない．

Kashibuchi らの研究 [19] では，クライアント端末からサーバに対し Handoff の通知を行った際，以降の処理をクライアント端末ごとに分散させるために応答のパケットを送出するタイミングを一定時間内で乱数的に決定していることが述べられている．このように，負荷集中の排除を目的としたクライアント端末ごとの確率的な分散手法は本研究でも有用であると考える．

本研究で扱う 2 つの課題のうちの 1 つである，同期の分散を図ることに対して既存研究 [15], [19] に見られる確率的な分散手法を本研究でも検討する．状態変化通知動作による同期干渉におけるもう 1 つの課題である，Android OS での状態変化通知動作によって起こる大域同期の検知については類似のものがないため，次章より分析する．

4. 提案手法

本研究ではアプリ開発者やユーザが意図しないタイミングにもかかわらず，同期干渉の発生によるアプリ起動で発生する通信集中を削減することを目的とし，大域同期起動を発生する可能性のある状態変化を検知し，その通知タイミングを制御することにより通信集中を削減する手法を提案する．大域同期起動を発生する可能性のある状態変化を 3 種類に整理し，それぞれの状態変化について大域同期性を判定する手法について述べる．また，大域同期性がある状態変化の通知を制御する手法として，遮断，遅延，通信のみ遅延の 3 パターンを検討し，アプリ挙動への影響と実装コストに基づいて選定した最適な制御手法についても述べる．

4.1 大域同期性の分類

状態変化が発生し状態変化通知機構が通知を行う際に状態変化を発生させた要因を取得し，大域同期起動の要因となる状態変化であるかどうかを判定する．大域同期を発生させる可能性がある場合には大域同期型，発生させない場合には大域非同期型として，状態変化を分類する．

状態変化通知動作による大域同期起動が発生しうる状態変化は，以下の 3 種類が想定される．これらの状態変化を契機として動作するアプリが端末にインストールされている場合には大域同期起動が発生する．

(1) ネットワーク側における状態変化
　圏外状態から圏内状態へと遷移するなど，エリア内の複数の端末に影響を及ぼす状態変化
　例）ラッシュ時の満員電車がトンネル内などの圏外エリアから圏内エリアへと移動することによる，乗車している人が所有する端末の圏内状態への変化
(2) 自律型アプリが動作時に引き起こす状態変化
　異なる複数の端末においても同一の時刻に動作する自律型アプリが，設定された時刻に動作して発生させる

状態変化
例）目覚ましアプリの鳴動時刻を毎朝7時ちょうどに設定された複数の端末がいっせいに画面オンおよび鳴動状態へと遷移することによる状態変化

(3) サーバからの通知を受信するアプリが引き起こす状態変化
複数の端末に対し同報を行うサーバからの通知を契機として動作するアプリが発生させる状態変化
例）複数の端末に対してメッセージを配信するサービスのクライアントアプリが画面オン状態へと遷移させることによる状態変化

4.1.1　ネットワーク側における状態変化

複数の端末で同一のタイミングにネットワーク状態の変化が発生すると，状態変化の通知動作により大域同期起動が発生する．しかし，ネットワーク状態の変化は機内モードへの切替えなど，ユーザ操作によっても発生する場合があるが，これはユーザが任意のタイミングで切り替えているため大域同期起動とは判断しない．本提案手法が対象とする事象は，ユーザが意図した動作でないにもかかわらず発生してしまう同期干渉であり，システムの挙動を変化させることで改善することを目的としている．一方で，もし多くのユーザが，たとえば10秒ほどの短期間にいっせいに操作を行った場合は大域同期起動による通信集中となる可能性はあるが，機内モードの解除をした際にデータの更新が発生することはユーザが意図した動作であることから，本提案手法では対象とはしない．

本提案手法では，ネットワーク側における状態変化のうち，ユーザ操作に起因する変化であれば大域非同期型，それ以外であれば大域同期型と判定する．具体的には，状態変化の発生前後における Wi-Fi や Bluetooth，モバイルデータ通信，機内モードなどのネットワーク接続状態に影響する設定項目の状態を比較し，設定状態が変化していればユーザ操作によるものと判定し，設定状態に変化がない場合はネットワーク側での状態変化と判定する．ただし，自律型アプリが API 経由で設定状態を変更した場合は，ユーザ操作ではなく自律型アプリ契機での状態変化と判定する．これにより，ネットワークでの接続性の変化を大域同期型とする．もちろん，ネットワーク側での接続性の変化がすべて大域同期を引き起こすとは限らないが，接続性の変化が複数の端末で発生しているかどうかは端末側だけで判断できないため，接続性の変化はつねに大域同期型と判断することとした．

4.1.2　自律型アプリが動作時に引き起こす状態変化

複数の端末において，同一のタイミングで動作する自律型アプリが存在する場合，この自律型アプリが発生させる状態変化の通知動作により大域同期起動が発生する．自律型アプリの開発者が任意に動作時刻を設定する場合に，この問題が発生しやすい．

自律型アプリが動作時刻を任意に設定するために Android OS では，Alarm Manager に対して Alarm を設定する方法が用いられることが多い．Alarm を設定する際には，2.3 節で述べたように，指定時刻において端末のプロセッサがスリープ状態の場合，プロセッサをスリープ状態から解除してただちに動作させる wakeup 型指定と，次に何らかの要因でスリープ状態から解除された時点で動作させる non-wakeup 型指定が存在する．また，指定時刻の形式として開発者が指定した時刻で動作させる指定方法（以降では Exact 指定と呼ぶ）と，システム側で生成した時刻で動作させる指定方法（以降では Inexact 指定と呼ぶ）とが提供されている．

本提案手法では，Alarm の登録状況を取得し，Exact 指定かつ wakeup 型指定である Alarm により動作したアプリを識別する．この種別の Alarm で動作するアプリは大域同期する可能性がある．識別したアプリのユーザ ID（uid）をリストとして保持しておき，次に状態変化が発生した際の指示要求元情報の uid と比較し，リストに含まれていれば大域同期型の状態変化，含まれていなければ大域非同期型の状態変化と判定する．なお，指示要求元情報とは，システムに対して状態変化を引き起こす指示を要求したアプリの uid のことを指し，たとえば，画面点灯を指示する API 呼び出しや，4.1.1 項で述べたネットワーク接続状態を変化させる API 呼び出しを行ったアプリが該当する．

4.1.3　サーバからの通知を受信するアプリが引き起こす状態変化

複数の端末に対してサーバから同報的に通知を送信する，つまり同一タイミングに通知が送られる場合，この通知を受け取ったアプリが発生させる状態変化の通知動作により大域同期起動が発生する．

サーバからの通知方法として，Google 社の提供する Google Cloud Messaging [20]（GCM）や WAP-Push [21] がよく用いられる．GCM によるアプリの起動は，起動対象となるアプリに対して特定の Intent を送信することで実現されている．WAP-Push についても同様に，WAP Push Manager から対象のアプリへ特定の Intent が送信される．

したがって，本提案手法では，送信される Intent を監視することにより起動されるアプリを識別する．4.1.2 項の場合と同様に，識別したアプリの uid をリストとして保持しておき，次に状態変化が発生した際の指示要求元情報の uid と比較し，リストに含まれていれば大域同期型の状態変化，含まれていなければ大域非同期型の状態変化と判定する．

4.2　大域同期性に基づく通知の制御

大域同期性を持つアプリによる端末状態変化を通知する場合，通知先アプリがカスケード起動することになるので，同期干渉が発生しないようタスクの動作タイミングを制御

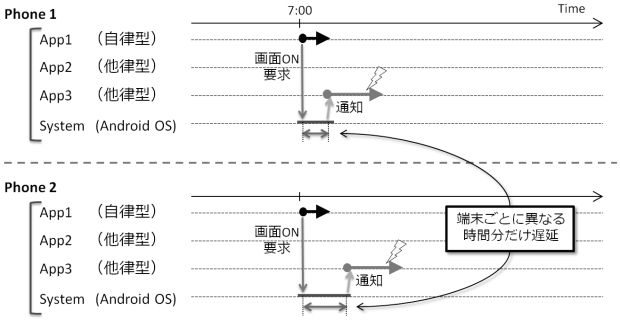


図 3　通知の遅延制御
Fig. 3　Control to delay of notifications.

表 3　通知制御方法の比較
Table 3　Comparison of controlling notifications.

	アプリ挙動への影響	実装コスト
通知の遮断	×（大きい）	○
通知の遅延	△（小さい）	○
通信のみ遅延	○（ない）	×（カーネル改変あり）

する必要がある．制御方法としては，以下の 3 つが考えられる．

- 通知の遮断
  通知先アプリへの通知を破棄する．通知先アプリがカスケード起動しないため，同期干渉は発生しない．
- 通知の遅延
  通知先アプリへの通知を遅延させる．**図 3** のように，遅延時間を端末間で異なる値とすることで，カスケード起動するタイミングが通知元アプリの動作タイミングとずれるため，同期干渉は発生しない．
- 通信のみ遅延
  通知先アプリへの通知は既存の機構と同様に即時通知を行う．通知先アプリでは同期干渉が発生することになるが，通信処理の開始タイミングを遅延させることで通信集中を避ける．ただし，通知先アプリの起動タイミングは大域同期した状態となるため，アプリが次回の起動タイミングを特定間隔（たとえば 30 分後）に設定してしまう場合，これを検知および回避することはできない．

それぞれの制御方法について，アプリ挙動へ与える影響，および実装コストを比較した結果を**表 3** に示す．

通知の遮断を行った場合，アプリは通知を受け取れることを前提に実装されているため，挙動に影響を与える可能性が高い．たとえば，電話の着信通知を受け取り，指定時間経過後に留守録機能を起動するアプリを考えた場合，そもそも通知を受け取れないため正常に動作しない．

通知の遅延を行った場合，遅延は発生するがアプリは通知を受け取れるためまったく動作しない状況は回避できる．また，Android OS の場合，通知の種類によっては通知対象アプリに対し優先順位の高いアプリから順に通知が行われており，優先度の高いアプリが処理を完了しないと通知が行われない．そのため，既存の機構においても通知が遅延する可能性があり，遅延制御を行うことが挙動として大きな変更ではないと考える．しかし，低遅延で通知が受け取れることを前提にしたアプリが存在する可能性はあり，その場合はアプリ挙動への影響もあると考えられる．

通信のみ遅延を行った場合，通知自体は即時に受け取れることとなり，既存の機構と何ら変わらないため，アプリ挙動への影響は軽微である．一方で，通知先アプリによる通信を遅延させるための処理が必要となり，Android OS の広範にわたって修正が必要となる．特に，Android NDK [22] を利用して作成されたアプリは Android OS のフレームワークを経由せずに直接カーネル側のソケットを利用できてしまうため，通信の遅延処理を行うためにはカーネル側の改変も必要となる．

以上より，我々は，実装コストおよびアプリ挙動への影響を抑えることが可能な，通知の遅延による制御を用いることとした．

5. 評価

提案手法を実装した端末に対して実際に端末状態変化を発生させ，意図しない大域同期起動が発生せずに，通信タイミングが分散することを検証する．一方で，ユーザ操作などの大域非同期型ではない端末状態変化に同期してアプリが動作する挙動は，アプリ開発者の意図した同期動作であるため，これらの動作については提案手法を実装していない既存機構と同様の通知が行われていることもあわせて検証する．

評価で用いた実装では，ベースとして AOSP（Android Open Source Project）プロジェクトよりオープンソースとして公開されている AOSP 4.0.3 r1 をベース実装として用い，フレームワーク層の JAVA 実装箇所に対し提案手法を追加した．実装の構成を**図 4** に示す．なお，図中では通知されるデータ，つまり BroadcastIntent を BI として記載している．状態変化発生要因の取得や通知の制御機能を実現するために，フレームワークに含まれるいくつかの Manager サービスに修正を加えている．具体的には，Broadcast Intent の配送を司る Activity Manager Service，画面点灯状態の切替えを制御する Power Manager Service および Window Manager Service，自律型アプリの動作タイミング契機となりうる Alarm を管理する Alarm Manager Service，そしてネットワーク接続状態を管理する Connectivity Service である．また，取得した発生要因に基づき，Broadcast Intent 配信の遅延が必要か判断する処理を，新たに設けた Broadcast Manager Service 内に実装した．なお，遅延時間は 30〜60 秒の範囲内で端末起動時刻を基に算出しており，端末ごとに異なるランダムな値

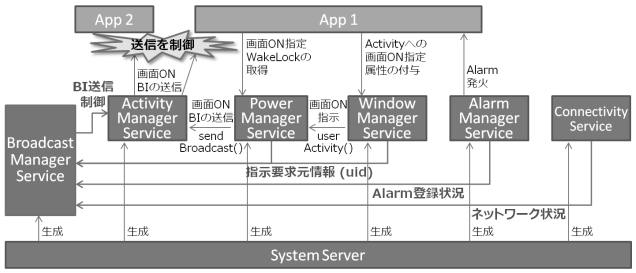


図 4　実装構成
Fig. 4　Architecture of implementation.

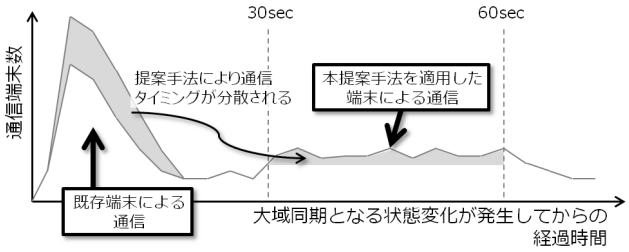


図 5　経過時間と通信端末数の関係概念図
Fig. 5　Relations between elapsed time and number of communicating devices.

としている．本来，遅延時間はネットワーク側での通信集中が発生しにくい程度の期間で設定するべきであり，たとえば基地局に在圏登録されている端末数や通信設備の設計容量などから決定する方法が想定される．また，各端末にインストールされているアプリケーションが通信処理を継続する時間によっても，必要とする遅延時間が変化する．なぜならば，1 つの端末あたりの通信時間が長くなるほど，ネットワーク側で通信が行われている確率が増えるためである．今回の評価実験で用いた端末では，同期干渉が発生した時点からおよそ 15 秒でインストールされているアプリの通信が完了することを確認したため，通信自体の遅延や端末の時計の誤差も考慮して本評価では遅延時間として余裕を持たせた 30 秒とした．一方で，端末ごとに遅延時間を変化させているため，遅延時間の範囲を 0 秒からとしてもよいように見えるが，実際の環境では本手法を適用していない端末との混在が想定されることから，最短でも 30 秒の遅延を発生させることとした．既存端末と本手法を適用した端末が混在した状態において，大域同期となる状態変化が起きてからの経過時間と通信を発生する端末数との関係は**図 5** のようになる．

遅延時間が短い場合は，アプリの動作タイミングが大きく変わらないためアプリ挙動への影響も軽微に済むものの，各端末の通信開始時間を十分に分散させるだけの時間幅が確保できないため通信集中を削減するには不十分となってしまう．一方で，遅延時間が長い場合は，各端末の通信開始時間が十分に分散するため通信集中を削減できるものの，アプリの動作タイミングが遅延することとなり，ユーザが遅延動作を意識しなければならなくなってしまう．したがって，遅延時間は通信集中を削減可能な範囲で短い期間とすることが望ましい．

表 4　インストールアプリの種類
Table 4　Types of installed apps.

	有	無
バックグラウンド動作	196 個	75 個
定時実行動作	163 個	33 個
状態変化通知時動作	147 個	49 個
状態変化通知時通信	84 個	63 個

評価実験の結果，ネットワーク側における状態変化，自律型アプリが動作時に引き起こす状態変化，サーバからの通知を受信するアプリが引き起こす状態変化の 3 パターンにおいて他律型アプリへの同期干渉は発生せず，意図しない大域同期起動も起こらなかった．通知タイミングが端末ごとに異なる時間分だけ遅延されるため，通信が発生するタイミングも分散し，通信の集中は見られなかった．また，ユーザ操作や大域非同期型に分類される状態変化の場合には，既存機構と同様に即座に通知が行われ，アプリ挙動にも変化はなかった．

同期干渉の発生有無は端末上で動作するアプリの種類などに依存するため，2.3 節でも述べたようにモニタユーザの協力を得てインストールアプリ数やインストールアプリ名を収集した．インストール数上位アプリの挙動を調べ，端末状態変化の通知を受け取った際の動作や，定時実行動作を模倣したアプリを作成し，ユーザ調査での平均的なアプリインストール状態とほぼ同等となるアプリ環境を再現した．インストールしたアプリのうち，バックグラウンドで動作するアプリ数や，バックグラウンド動作するアプリのうち，定時実行動作や端末状態変化通知を受け取った際に動作するアプリ数，および，その際に通信を発生するアプリ数を**表 4** に記載する．また，2.3 節で述べた定時実行動作による同期干渉を排除するため，端末がつねに非スリープ状態となるようにした．なぜならば，端末がスリープ状態において状態変化が発生すると端末が非スリープ状態へと遷移するので定時実行動作による同期干渉も同時に発生する可能性があるためである．これにより，状態変化通知動作による同期干渉との区別が困難となってしまうため，つねに非スリープ状態として定時実行動作による同期干渉を排除した．

詳細な実験環境は以下のとおりである．

実験環境

- 端末仕様
  プロセッサ：ARM v7 デュアルコア 1.2 GHz
  RAM 容量：1 GB
  ROM 容量：16 GB
  通信方式：UMTS(3G)，GSM，IEEE802.11a/b/g/n

表 5　状態変化時の起動回数
Table 5　Count of invocations when states is changed.

	状態変化時 起動回数	状態変化時 通信有無	通知配送時 起動回数
ネットワーク接続 状態変化（既存機構）	5 回	有	—
ネットワーク接続 状態変化（提案手法）	0 回	無	5 回
自律型アプリによる 画面オン（既存機構）	5 回	有	—
自律型アプリによる 画面オン（既存機構）	0 回	無	5 回
サーバ通知による 画面オン（既存機構）	5 回	有	—
サーバ通知による 画面オン（既存機構）	0 回	無	5 回

（無線 LAN），Bluetooth 3.0+HS

- OS バージョン
  Android AOSP 版 4.0.3r1
- インストールアプリ
  インストール数上位アプリの動作模倣アプリ：271 個
- スリープ状況
  つねに非スリープ状態
- 通信状況（実験用環境で測定）
  3G による通信のみ
  ただし，5.1.1 項の評価時は Wi-Fi と 3G による通信

5.1　同期干渉の回避

4.1 節で述べた大域同期を発生させる可能性のある状態変化について，必ず大域同期が発生する条件を設定した．つまり，複数の端末で同一のタイミングに状態変化を発生させた．各状態変化を発生させる方法については各項に記載する．この条件で状態変化を 5 回発生させ，その際に他律型アプリが起動した回数，および，通信の有無を確認することで意図しない大域同期起動が発生していないかを検証した．また，遅延させた通知が配送されたタイミングにおいて，他律型アプリが起動した回数も確認した．

5.1.1　ネットワーク側における状態変化

ネットワーク側における状態変化を発生させるため，Wi-Fi のアクセスポイントで無線接続を無効化することにより，複数端末において Wi-Fi での接続状態から 3G での接続状態への切替えを発生させることで評価を行った．接続状態切替え時における他律型アプリの起動回数および通信有無を**表 5** に示す．既存機構では接続状態が変化したタイミングに同期してアプリが動作しており，その際に通信が発生した．つまり，電車に乗車中など，複数端末の接続状況がいっせいに変化しやすい環境においては通信タイミングが集中する事象が発生していると考えられる．一方で提案手法では，接続状態が変化したタイミングに同期した他律型アプリの動作を遅延させているため，状態変化に同期した通信も発生しなかった．

5.1.2　自律型アプリが動作時に引き起こす状態変化

自律型アプリとしてアラームアプリを用い，設定時刻において画面オン状態への状態変化を複数端末で同一の時刻に発生させることで評価を行った．画面オン時の起動回数および通信有無を表 5 に示す．既存機構では画面オンのタイミングに同期してアプリが動作することで通信が発生しており，同一時刻にアラームを設定する端末数が増えることで問題となる可能性がある．一方で提案手法では，画面オン状態への遷移による状態変化に同期した他律型アプリの動作を遅延させているため，状態変化に同期した通信も発生しなかった．遅延して画面オンの通知が配送されると，その時点でアプリが動作して通信が発生した．本提案手法では，既存機構に比べ通知までの時間が長くなるため，たとえば前回通信時からの経過時間が閾値を超えたら通信を行うなどの判定処理を行っているアプリでは，通知を配送した時点で通信が発生する可能性は少なからず増えるものの，処理タイミングが端末間で分散されるよう遅延を行っているため，問題ないと考える．

5.1.3　サーバからの通知を受信するアプリが引き起こす状態変化

サーバからの通知を受信するアプリとしてメッセンジャーアプリを用い，他の端末から評価対象の複数端末に対して同一のタイミングでメッセージを送信し，画面オン状態への状態変化を発生させることで評価を行った．画面オン時の起動回数および通信有無を表 5 に示す．5.1.2 項と同様に，既存機構では画面オンのタイミングに同期してアプリが動作することで通信が発生している一方で，提案手法では，画面オン状態への遷移による状態変化に同期した他律型アプリの動作を遅延させているため，状態変化に同期した通信も発生しなかった．また，遅延して画面オンの通知が配送されると，その時点でアプリが動作して通信が発生した．

5.2　通信集中の回避

5.1 節での評価より，同期干渉の発生が回避されていることが確認できた．本提案手法では，遅延させる時間を端末間で異ならせることにより通信タイミングを端末間で分散させることで通信集中の削減を図っているが，実際に通信集中を削減できているかを検証する．具体的には，同時に複数の端末で状態変化が発生した際に通信が発生する端末数を測定した．ただし，基地局側から端末への通信を制御することは端末側での改善だけでは難しく，本提案手法で対象としているのは端末側契機での上り通信セッションの開始タイミングとなるため，評価においても上り通信のみを測定対象とした．

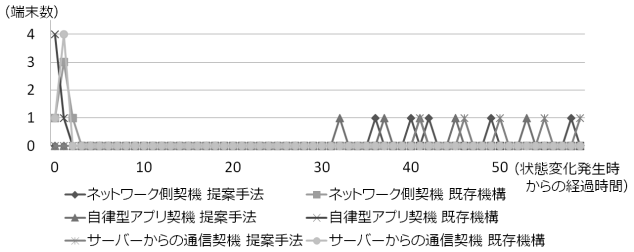


図 6　上り通信セッション確立端末数
Fig. 6　Count of devices that establish up-link connection.

5 台の評価端末における上り通信セッションの確立端末数を**図 6** に示す．既存機構では状態変化が発生するとただちに他律型アプリが動作し，その際に通信が発生するため状態変化発生時点から 2〜3 秒以内に通信確立が集中しており，5 台中 4 台の端末において通信が発生していた．一方で，提案手法では遅延時間を 30〜60 秒までの範囲内で端末のブート時刻に基づいて決定されるため，状態変化発生時点から 30〜60 秒の範囲に分散して通信確立が発生しており，同時に通信確立している端末数は最大でも 1 台であった．この結果より，通信集中の削減が図れていることが分かる．

5.3　既存機構動作の維持

ここまでで述べてきたとおり，ユーザ操作や大域非同期型に分類される状態変化による端末状態変化に同期してアプリが動作する挙動は，アプリ開発者の意図した同期動作であると考えられる．画面オフ状態から画面オン状態への状態変化を 5 回発生させ，その際に他律型アプリが起動した回数，および，通信の有無を確認することで，既存機構と同様の通知動作が維持されていることを検証した．

5.3.1　大域非同期型に分類される状態変化

大域非同期型に分類される状態変化を発生させるため，Alarm Manager への Alarm 登録において Exact 指定ではなく Inexact 指定により動作時刻を設定し，動作時刻において画面オン状態への状態変化を発生させる評価用アプリを作成した．この評価用アプリにより画面オン状態への状態変化を発生させることで評価を行った．画面オン時の起動回数および通信有無を**表 6** に示す．既存機構と同様に，提案手法においても画面オン時のタイミングに同期してアプリが動作しており，その際に通信が発生している．

大域非同期型に分類される状態変化は，発生タイミングが各端末によって異なるため，画面オン時に通信が発生しても問題はない．

5.3.2　ユーザ操作による状態変化

ユーザ操作により画面オン状態への状態変化を発生させることで評価を行った．画面オン時の起動回数および通信有無を表 6 に示す．既存機構と同様に，提案手法においても画面オン時のタイミングに同期してアプリが動作し通信が発生している．

ユーザ操作による状態変化は，発生タイミングが各端末によって異なるため，画面オン時に通信が発生しても問題はない．

表 6　大域非同期な状態変化時の起動回数
Table 6　Count of invocations when state is changed by a non-globally synchronized app or a user operation.

	状態変化時起動回数	状態変化時通信有無
大域非同期型アプリによる画面オン（既存機構）	5 回	有
大域非同期型アプリによる画面オン（提案手法）	5 回	有
ユーザ操作による画面オン（既存機構）	5 回	有
ユーザ操作による画面オン（提案手法）	5 回	有

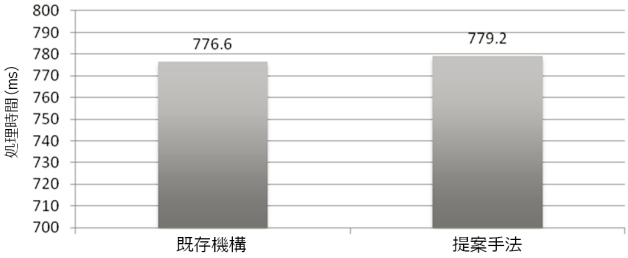


図 7　通知動作完了までの処理時間
Fig. 7　Elapsed time of finishing notification.

5.4　提案手法による状態変化通知動作へのオーバヘッド

提案手法では，状態変化の発生要因を取得する処理や，大域同期していると判定したアプリのリストに対し状態変化が発生した際の指示要求元情報が含まれるか判定する処理が必要となる．そこで，既存機構に対して提案手法の実装を追加したことによるオーバヘッドが十分に小さいことを検証した．検証に際しては，画面オン状態への遷移を意味する ACTION_SCREEN_ON をアクション名に持つ Broadcast Intent を対象とし，Power Manager Service 内で画面オン状態への遷移が発生した時点から，通知対象であるアプリすべてへの通知が完了するまでの処理時間を測定した．なお，通知対象となるアプリは同一とし，アプリリストに含まれるアプリの個数は 3 つとした．これは評価条件におけるインストールアプリのうち同時刻に動作するアプリ数がたかだか 3 つであり，アプリリストに保持されるアプリ数もおおよそ 3 つ以下となるためである．

処理時間の測定結果を**図 7** に示す．この結果より，既存機構と比較した提案手法のオーバヘッドは約 0.3%であり，全体の処理時間に対する影響はほとんど無視できるといえる．

6. おわりに

Android OS における通信集中の発生要因として，状態変化の通知動作による同期干渉の発生にともない大域同期起動した他律型アプリが通信を発生することがあげられる．同期干渉により大域同期起動してしまうことはアプリ開発者の意図していない動作であり，この挙動を意識した実装をすべての開発者に求めることは難しい．

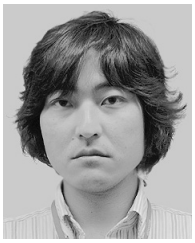
本研究では，意図しない大域同期起動を引き起こす同期干渉を抑制するために，大域同期起動しやすい状態変化を大域同期型の状態変化として分類し，大域同期性を持っている場合は，通知の配送タイミングを遅延するよう制御した．提案手法を AOSP 版の Android OS に実装し，大域同期が発生しうるネットワーク側における状態変化，自律型アプリが動作時に引き起こす状態変化，サーバからの通知を受信するアプリが引き起こす状態変化のいずれにおいても，通知タイミングが遅延され，それにともない他律型アプリが発生する通信タイミングも分散されることを実端末による実験により確認した．

参考文献

[1] Google Inc.: Android - Discover Android, available from ⟨http://www.android.com/about/⟩ (accessed 2013-07-23).
[2] Apple Inc.: アップル - iOS 6, available from ⟨http://www.apple.com/jp/ios/⟩ (accessed 2013-07-23).
[3] Microsoft Corporation : Windows Phone, available from ⟨http://www.microsoft.com/ja-jp/windowsphone/⟩ (accessed 2013-07-23).
[4] Telefonaktiebolaget L.M. Ericsson: Traffic and Market Report (June 2012).
[5] Maier, G., Schneider, F. and Feldmann, A.: A first look at mobile hand-held device traffic, *Passive and Active Measurement*, pp.161–170 (2010).
[6] Hossein, F. et al.: Diversity in smartphone usage, *Proc. 8th international conference on Mobile systems, applications, and services*, pp.179–194 (2010).
[7] Xu, Q. et al.: Identifying diverse usage behaviors of smartphone apps, *Proc. 2011 ACM SIGCOMM conference on Internet measurement conference*, pp.329–344 (2011).
[8] Apple Inc.: iOS App Programming Guide (2013).
[9] Microsoft Corporation: Background agents for Windows Phone, available from ⟨http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202942%28v=vs.105%29.aspx⟩ (accessed 2013-07-23).
[10] Google Inc.: Multitasking the Android Way, available from ⟨http://android-developers.blogspot.jp/2010/04/multitasking-android-way.html⟩ (accessed 2013-07-23).
[11] Qian, F. et al.: Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization, *Proc. 21st international conference on World Wide Web*, pp.51–60 (2012).
[12] Google Inc.: Alarm Manager, available from ⟨http://developer.android.com/reference/android/app/AlarmManager.html⟩ (accessed 2012-12-05).
[13] Google Inc.: Broadcast Intent, available from ⟨http://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)⟩ (accessed 2012-12-05).
[14] Jacobson, V. and Karels, M.J.: Congestion Avoidance and Control, *ACM SIGCOMM Computer Communication Review*, Vol.18, No.4, pp.314–329 (1988).
[15] Floyd, S. and Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Trans. Networking*, Vol.1, No.4, pp.397–413 (1993).
[16] Floyd, S., Gummadi, R. and Shenker, S.: Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, Technical report, ACIRI (2001).
[17] Abbasov, B. and Korukoglu, S.: Effective RED: An algorithm to improve RED's performance by reducing packet loss rate, *Journal of Network and Computer Applications*, Vol.32, No.3, pp.703–709 (2009).
[18] 小西哲平，神山　剛，川崎仁嗣，稲村　浩：Android 端末のための画面オフ状態におけるバックグラウンドタスク実行タイミング制御手法の検討，情報処理学会マルチメディア通信と分散処理ワークショップ論文集，Vol.2012, No.4, pp.249–256 (2012).
[19] Kashibuchi, K. et al.: A new smooth handoff scheme for mobile multimedia streaming using RTP dummy packets and RTCP explicit handoff notification, *Wireless Communications and Networking Conference*, pp.2162–2167 (2006).
[20] Google Inc.: Google Cloud Messaging for Android, available from ⟨http://developer.android.com/google/gcm/index.html⟩ (accessed 2013-06-26).
[21] Open Mobile Alliance Ltd.: Push Architectural Overview, available from ⟨http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-250-pusharchoverview-20010703-a.pdf⟩ (accessed 2013-06-26).
[22] Google Inc.: Android NDK, available from ⟨http://developer.android.com/tools/sdk/ndk/index.html⟩ (accessed 2012-12-05).

川崎 仁嗣　（正会員）

株式会社 NTT ドコモ先進技術研究所勤務．平成 20 年筑波大学大学院システム情報工学研究科博士前期課程修了．同年（株）NTT ドコモ入社．モバイルコンピューティング，端末セキュリティ，分散システムに関する研究に従事．



神山 剛　（正会員）

株式会社 NTT ドコモ先進技術研究所勤務．平成 18 年東京大学大学院新領域創成科学研究科修士課程修了．同年（株）NTT ドコモ入社．モバイルコンピューティング，ソフトウェア省電力化，分散システムに関する研究に従事．

小西 哲平　（正会員）

株式会社 NTT ドコモ先進技術研究所勤務．平成 22 年大阪大学大学院基礎工学研究科博士前期課程修了．同年（株）NTT ドコモ入社．ソフトウェア省電力化に関する研究に従事．

大久保 信三

株式会社 NTT ドコモ先進技術研究所勤務．平成 3 年電気通信大学電子情報学科卒業．平成 5 年同大学院電子情報学専攻博士前期課程修了．同年 NTT 移動通信網（株）（現（株）NTT ドコモ）入社．以来，高度無線呼出方式，次世代移動通信システム，近距離無線方式の研究開発に従事．電子情報通信学会会員．

太田 賢　（正会員）

株式会社 NTT ドコモ先進技術研究所勤務．平成 10 年静岡大学大学院博士課程修了．平成 11 年 NTT 移動通信網（株）入社．モバイルコンピューティング，端末セキュリティ，分散システムに関する研究に従事．訳書「コンピュータネットワーク第 5 版」等．電子情報通信学会会員．博士（工学）．

稲村 浩　（正会員）

株式会社 NTT ドコモ先進技術研究所勤務．平成 2 年慶應義塾大学大学院理工学研究科修士課程修了．同年日本電信電話（株）入社．平成 6 年から 7 年にカーネギーメロン大学計算機科学科にて訪問研究員．平成 10 年より NTT ドコモ．モバイル環境におけるシステムソフトウェア，トランスポートプロトコルに関する研究開発に従事．電子通信情報学会，ACM，IEEE 各会員．博士（工学）．