通信処理を考慮したバックグラウンドタスクの一括実行スケジューリングによる Android 端末の省電力化

川西　将輝[†a)]　　森野　博章[†]

Energy Aware Batch-based Background Task Scheduling in Android OS Considering Communication Processes

Masaki KAWANISHI[†a)] and Hiroaki MORINO[†]

あらまし　SNS を始めとしてユーザへリアルタイム通知を行うスマートフォンアプリケーションが広く利用されている．これらはユーザが端末を操作しない時間帯でもバックグラウンドで定期的にサーバとの間で通信を行うため通知を行うアプリケーションが端末内で複数動作すると消費電力が増加する．本論文では Android OS を対象としてこの問題を緩和するため，複数のバックグラウンド処理を一括実行するバッチ方式に着目し，事前の調査に基づいて高い確率で通信を行うことが予想できるバックグラウンド処理を可能な限り一つのバッチで一括実行するようにスケジューリングを行う手法を提案する．本方式を実機に実装し模擬アプリケーションを実行させて性能評価を行い，実行予定時刻の早い順に一括実行する既存のバッチ方式と比較して消費電力が低減されることを示す．

キーワード　Android OS，バックグラウンド処理，通信，消費電力，アラーム，バッチ

1. まえがき

Facebook や Twitter を始めとしてユーザへのリアルタイム通知機能を備えた SNS アプリケーションがスマートフォンを中心にして広く利用されている．これらはユーザが端末を操作しない時間帯でも定期的にサーバへ接続する，あるいはサーバへの通信コネクションを維持するといった処理をバックグラウンドで行う必要があるため，端末内で通知を行うアプリケーションが複数動作すると消費電力が増加する．バックグラウンドで動作するアプリケーションが端末の消費電力に与える影響については盛んに研究されている [1], [2]．例えば [1] ではバックグラウンドで通信を行うアプリケーションが複数動作するとそれらが全く動作しない状態と比較して消費電力が最大で 3.6 倍に増加すると報告されている．

バックグラウンドで行われる通信に伴う消費電力の低減を図るアプローチの一つとして，ユーザの操作頻度が少ないと判断されるアプリケーションの通信を制限する手法が検討されてきたが，SNS の場合は同様の頻度で複数のアプリケーションを利用するユーザも多いと考えられる．また，例えばメールやファイル同期など，ユーザの操作頻度が少なくとも利用する際には直ぐにデータを取得したいようなアプリケーションの場合，頻度のみに基づいて通信が制限されると同期完了までの遅延増大等の品質低下を招くため，アプリケーションの個別の動作特性に応じて制限の有無を決定するなどを行う必要があり，ユーザに新たな操作コストが加わる可能性がある．

こうした問題を回避し，アプリケーションの品質を低下させることなく通信に伴う消費電力を低減する手法として，本論文では Android OS において複数の処理を一括実行するバッチ方式に着目し，無線通信処理を考慮してバックグラウンド処理の実行時刻をスケジューリングする手法を提案する．本方式ではアプリケーションの通信特性を事前に調査し，OS に予約実行登録されるバックグラウンド処理を通信を伴うもの

† 芝浦工業大学大学院理工学研究科，東京都
Graduate School of Engineering and Science, Shibaura Institute of Technology, 3–7–5 Toyosu, Kouto-ku, Tokyo, 135–8548 Japan
a) E-mail: ma15028@shibaura-it.ac.jp
DOI:10.14923/transcomj.2017MOP0008

とそうでないものに分類し，通信を伴う複数のタスクが一括実行されるようにスケジューリングすることで省電力化を図る．提案方式を AOSP (Android Open Source Project) 版 Android OS [3] に適用して実機により評価を行い，通信を考慮せずに一括実行する既存のバッチ方式と比較して消費電力が低減されることを示す．

2. 研究背景

2.1 リアルタイム通知を行うアプリケーションとサーバとの接続形態

端末のアプリケーションにおいてサーバがもつユーザへの通知情報をリアルタイムで取得する手法には大きく分けてプル型とプッシュ型がある．プル型ではアプリケーションが定期的にサーバへ接続し情報を取得し，プッシュ型ではサーバとアプリケーションの間で TCP を始めとするトランスポートプロトコルのコネクションを常に維持しておき，サーバで情報が発生すると直ちに端末に送信される．プル型でも接続遅延を低減するためトランスポートコネクションを維持する形態もある．

トランスポートコネクションは一定時間パケットの送受信が行われないと切断されるためコネクションを維持するためには定期的に制御パケット，例えば TCP の場合は KeepAlive パケットをアプリケーションからサーバへ送信する必要がある．したがってプル型，プッシュ型とも何らかの形で定期的にサーバへパケットを送信している．ユーザが操作しているフォアグラウンドのアプリケーション，あるいはバックグラウンドで継続して動作するアプリケーション（音楽再生など）がない状況で上述のサーバへの定期的な接続・接続維持処理のみが行われる場合は，端末は通常はスリープ状態でパケットの送受信が行われる時間帯のみアクティブ状態になることを繰り返すことになる．

2.2 LTE 通信における端末の状態遷移と電力消費

一般にセルラー端末では通信を行っていない時間帯に通信機能を停止させることで消費電力を低減する機能をもっている．LTE では端末の Radio Resource Control (RRC) の状態として基地局に接続してパケットの送受信を行う `RRC_CONNECTED` と，ページング (基地局からの呼び出し) の受信のみを行うことができる `RRC_IDLE` があり，`RRC_IDLE` では一定周期でページングチャネルをモニタする他は大部分の時間で通信機能がオフとなる．`RRC_CONNECTED` では通信機能が常にオンになっている状態 (以下，Active モードと呼ぶ) で DRX inactivity timer が切れるまでの間にパケットの送受信が行われないとある一定周期で通信機能のオンとオフを繰り返して消費電力を低減する C-DRX (Connected mode Discontinuous Reception) モードに入る (図 1) [4], [5]．同様に Active モードでセットされた Inactivity timer が切れるまでに通信が行われないと，`RRC_IDLE` に遷移する．

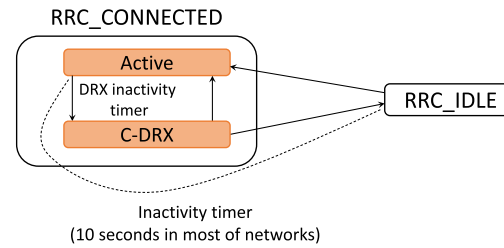

図 1　端末における RRC の状態遷移
Fig. 1　State transision of RRC in a user equipment.

すなわち，`RRC_IDLE` の状態にある端末が基地局と接続してパケットの送受信を行うと，状態は `RRC_IDLE` → `RRC_CONNECTED` (Active モード) → `RRC_CONNECTED` (C-DRX モード) → `RRC_IDLE` と遷移し，通信モジュールの消費電力は `RRC_CONNECTED` (Active モード)，`RRC_CONNECTED` (C-DRX モード)，`RRC_IDLE` の順で大きい．

`RRC_CONNECTED` (Active モード) から `RRC_IDLE` へ移行する Inactivity timer の値は一般には 10 秒程度に設定されており，**2.1** で述べた，リアルタイム通知を行うアプリケーションが定期的にサーバに接続してサイズの小さいパケットを送受信する場合では，送受信が終了しても一定時間は通信機能がオンである時間帯が存在していることになる．

2.3 Android OS におけるバックグラウンド処理の実行機構

Android OS 上でアプリケーションの処理を実装する際に選択できる主な Java クラスとしてアクティビティ，サービス，ブロードキャストレシーバがあり，それぞれプロセスの実行形態が異なるが，中でもサービス (インテントサービスも含む) とブロードキャストレシーバとして実装されたプロセスは画面の状態によらず実行が継続されるため，バックグラウンドで動作させたい処理に用いられる．サービスやブロードキャストレシーバとして実装したプログラムを時刻を指定して実行する，あるいは一定周期で繰り返し実行する場

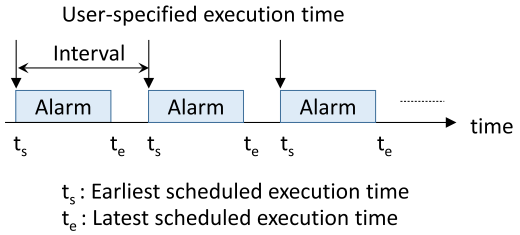

図 2　繰り返し実行アラームに設定される実行時刻範囲の例
Fig. 2　An example of execution time range for repeatedly executed alarms.

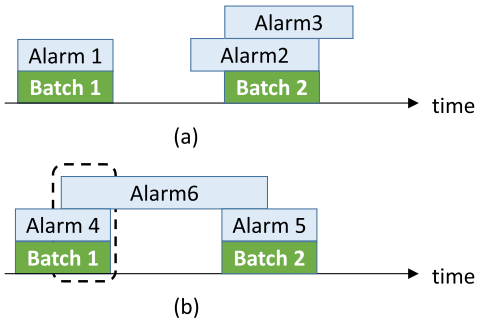

図 3　バッチ設定の動作
Fig. 3　Operation of batch configuration.

合には当該プログラムを希望時刻・希望実行形態の情報とともに AlarmManager と呼ばれる管理機構に登録する．本論文ではこれらのプログラムを主な議論の対象とし，登録される各プログラムをアラームと呼ぶ．

AlarmManager にアラームを登録するメソッドには `set()`,`setExact()`,`setRepeating()` などがあり，`set()`,`setExact()` は一回のみ実行する場合，`setRepeating()` は一定周期で繰り返し実行する場合に用いられる．一回のみ実行する場合の実行時刻，繰り返し実行の場合の初回時刻及び繰り返し時間間隔は引数で指定する．前述のリアルタイム通知を行う既存のアプリケーションではサーバへのパケット送信処理の登録に `setRepeating()` が多用されている．

AOSP 版 Android OS の Ver.4.4 以降では AlarmManager に登録された各アラームに対して実行時刻範囲 $r = [t_s, t_e]$ (t_s: 最早実行時刻，t_e: 最遅実行時刻) が設定される．実行時刻範囲の大きさはアラームを登録する際に用いられたメソッドによって異なり，`set()`,`setRepeating()` ではユーザが指定した実行時刻を t_s とし t_e は繰り返し実行間隔・登録時の時刻等から決定されるのに対して `setExact()` では $t_s = t_e$ となりユーザが指定した時刻に必ず実行される．図 2 は `setRepeating()` で登録されたアラームの実行時刻範囲を示しており，ユーザが指定した初回実行時刻，実行間隔を t_0，t_{int} とすると n 回目の繰り返し実行の時刻範囲 $[t_s(n), t_e(n)]$ は $t_s(n) = t_0 + (n-1) * t_{int}$，$t_e(n) = t_s(n) + 0.75 * t_{int}$ と設定される．

このように範囲を設定した後，範囲に重なりがある複数のアラームをバッチと呼ばれる処理単位で一括実行することで端末がアクティブ状態である時間を低減し省電力化を図っている．

各々のバッチはそれがカバーするアラームの時刻範囲 $r' = [t'_s, t'_e]$ をもっており，新たに登録されるアラーム (新規アラーム) がどのバッチで実行されるかは以下の手順で決定される．

（1） 新規アラームと時刻範囲が重なる既存のバッチが存在しない場合は，そのアラームと同じ時刻範囲をもつバッチが新たに生成される．新規アラームの時刻範囲の一部が既存のバッチの時刻範囲と重なる場合は，そのバッチで新規アラームを一緒に実行するよう設定され，バッチの時刻範囲は実行する全てのアラームを最小の時刻範囲でカバーするように変更される．図 3 (a) の Batch 1 は Alarm 1 に対して新たに生成されるバッチ，Batch 2 は当初 Alarm 2 に対して設定されたバッチであるが新規の Alarm 3 も含むことで範囲が変更されたケースである．

（2） 新規アラームの時刻範囲の一部が複数の既存のバッチの時刻範囲に含まれる場合は最早実行時刻 t_s が最も早いバッチに設定される．図 3 (b) の Alarm 6 の場合は Batch 1 と Batch 2 に設定されうるが，重なる時刻範囲のより早い Batch 1 で実行されるよう設定され，Batch 1 の時刻範囲は Alarm 4 と Alarm 6 の範囲の共通部分 (破線で囲まれた部分) に変更される．

2.4　通信の電力消費の観点で見た既存バッチ方式の問題点

前節で述べた AlarmManager における既存のバッチ方式は端末の CPU がスリープ状態にある時間を増加させて消費電力を低減すること主目的として設計されている反面，無線通信の消費電力低減については十分考慮されていない．すなわち，一般に AlarmManager に登録されるアラームには通信を行うものとそうでないものがあるが，既存方式は新たに登録されるアラームに対して重なる時刻範囲が時間的に最も早い既登録アラームとまとめるようにバッチが設定されるため，通信を行う複数のアラームが一括実行されるとは限ら

ない．例えば通信を行うアラームを比較的長い間隔で実行するアプリケーションが複数と，通信を行わないアラームを短い間隔で実行するアプリケーションがそれぞれ動作しているような場合では，既存のバッチ方式では通信を行うアラーム同士ではなく，通信を行わないアラームと通信を行うアラームでバッチが設定される可能性が高くなり，無線通信の消費電力削減効果は限定的である．

3. 通信処理を考慮したアラームスケジューリング方式の提案

以上に述べた既存技術の問題を解決するため，AlarmManager において通信を伴うアラームが一括実行されるようにスケジュールすることで，アラームの実行時刻を大きく遅延させることなく無線通信の消費電力を低減するバッチ設定方式を提案する．

3.1 方式の動作

本方式は，アプリケーション特性の事前調査に基づいて新たに登録されるアラームが通信を行うアラームかどうかを判定し，通信を行うアラームである場合に以下の手順でバッチを設定する．

（1） 新規アラームへ設定する実行時刻範囲内に通信を行う既登録のアラームが存在する場合，その既登録アラームと新規アラームでバッチを設定する．

（2） 新規アラームの実行時刻範囲内に通信を行う既登録アラームが存在しない場合，従来と同じ方式でバッチを設定する．

図 4 に動作例を示す．Alarm 1, 2, 3 が既に設定されており，新規アラームとして Alarm 4, 5 を設定する場合の例である．ここで Alarm 2, 4, 5 は通信を行うアラーム，Alarm 1, 3 は通信を行わないアラームとなっている．Alarm 4 は Batch 1 と Batch 2 のどちらにも設定されうるが，提案方式では通信を行うアラームが含まれている Batch 2 で実行するように設定する．既存方式では最早実行時刻がより早いアラーム同士を組み合わせるため Alarm 4 は Batch 1 に設定され，従って Batch 1 と Batch 2 でそれぞれ通信が行われるのに対して，提案方式では Batch 1 では通信が行われないため，通信モジュールがアイドル状態である時間が増加し，消費電力の低減が図られる．

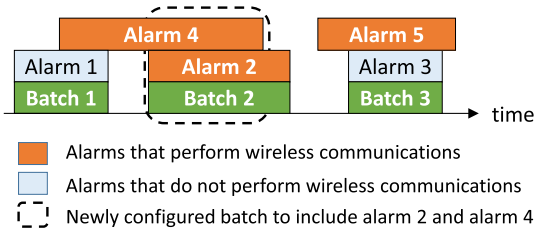

図 4 提案方式の動作例
Fig. 4 Example of the proposed alarm scheduling scheme.

Alarm 5 については選択可能な Batch が一つのみであることから Batch 3 に設定される..

本方式が有効に動作するには，バッチが設定されるアラームの特性について以下の条件が必要となる．

（1） アラームが実行されて直ちに (おおむね数秒以内に) 通信が行われること．

アラームが実行されてからそのプログラムで通信が行われるまでの時間はアプリケーションによって異なり，後述するように同じアプリケーションでもアラームごとに数秒から数十秒の間で変動するものも存在する．それらに本方式を適用しても有効性は発揮されにくい．アラーム実行後直ぐに通信が行われることが分かっているアプリケーションのアラーム同士を一括実行する必要がある．

（2） アラームの実行時刻範囲の t_e と t_s の差（以下，区間幅と呼ぶ）が十分大きいこと．

この条件はバッチを柔軟に構成するために必要である．

3.2 アラームの実行に関する計測

前述した，提案方式の実行に必要なアラームの特性の条件を満たすアプリケーションを調査する目的で，利用者数の多い幾つかのアプリケーションに関してアラームの実行時刻と通信処理の特性を計測する実験を行った．端末には Nexus 5 (LG-D821)，モバイル回線には NTT Docomo の MVNO 回線，OS には AOSP 版の Android 4.4 を用いた．AOSP 版の Android はそのままではアプリケーションの一部機能が利用できないため，Gapps [6] をインストールしユーザの実環境と同じ設定で測定を行った．評価対象のアプリケーションは，Facebook, Instagram, Twitter, LINE (以上 SNS)，Avast (アンチウィルス)，LifeBear (カレンダー)，Google Fitness, Lifelog (以上フィットネス) とした (括弧内はアプリケーションのカテゴリーを示す)．

アラームが実行された後に通信が開始されるまでの時間の特性を表 1 に，アラームの実行間隔と区間幅に関する結果を表 2 にそれぞれ示す．

表 1 に示すように，Facebook，Instagram，Avast

では全てのアラームが，Twitter では全体の 86%のアラームが，それぞれ実行後 5 秒以内に通信を行っている．したがってこれら三つのアプリケーションのアラームを一括実行することで通信の消費電力が低減する可能性が高い．一方，LINE では実行後 5 秒以内に通信を行うアラームは全体の 64%で，34%は 20 秒経過した後に通信を行うか，そもそも通信を行わないアラームであり，他の通信を行うアプリケーションのアラームと一括実行することで見込める効果はやや低いことが分かる．

次に，表 2 の結果から，Facebook, Instagram, Twitter はいずれもアラーム実行間隔が 15 分で，区間幅は 11 分 15 秒と比較邸長いため，これらのアラームを一括実行することは容易である．LINE はアラームの実行間隔が変動することから各々のアラームが実行される時間範囲に他の通信を行うアラームが存在するとは限らないこと，また Avast は通信の頻度が低いことからそれぞれ一括実行による効果が得られにくい．通信を行わない LifeBear, Google fitness, Lifelog の中で特徴的なのは Lifelog であり，アラーム実行間隔が短く，区間幅が 0 秒となっている．Lifelog と通信を行う複数のアプリケーションが動作している場合には，既存のバッチ方式を適用すると後者のアラームは Lifelog のアラームと同じバッチで実行される可能性が高く，通信を行うアラーム同士で一括実行されることは少ないことから，提案方式を適用することで得られる効果が高いと考えられる．

表 1　アラーム実行と通信開始の時間差特性
Table 1　Time difference between the start of alarm execution and wireless communication.

アプリケーション	0〜5 秒	5〜20 秒	20 秒〜	通信処理なし
Facebook Instagram	100%	0%	0%	0%
Twitter	86%	0%	12%	1%
LINE	64%	2%	27%	7%
Avast	100%	0%	0%	0%
LifeBear	0%	0%	0%	100%
Google fitness	0%	0%	1%	99%
Lifelog	0%	0%	0%	100%

表 2　アラームの実行間隔と区間幅の特性
Table 2　Characteristics on execution time interval and schedule time range width of alarms.

アプリケーション	アラーム登録メソッド	アラーム実行間隔	区間幅
Facebook Instagram Twitter	`setRepeating()`	15 分	11 分 15 秒
LINE	`set()`	10〜20 分	10〜20 分
Avast	`set()`	240 分	1〜6 分
LifeBear	`setRepeating()`	30 分	22 分 30 秒
Google fitness	`seRepeating()`	15 分	11 分 15 秒
Lifelog	`setExact()`	1 分	0 秒

4. 性能評価

3.2 の計測結果を踏まえ，提案方式を実機に実装して，既存のバッチ方式を比較対象として特性を評価する．

4.1　模擬アプリケーションによる消費電力評価

本節においては **3.2** の計測で評価対象としたアプリケーションの動作を模擬するように新たに作成したプログラム (以下，模擬アプリケーションと呼ぶ) を評価対象として用いた．予備実験を通して，この方法は実際のアプリケーションで評価を行う場合に比較し試行回ごとの結果のばらつきが小さいことが明らかになったことから，アラームの実行に関する消費電力の評価に重点を置くという観点でこの方法を採用した．

4.1.1　評価環境及び評価対象アプリケーション

実験に用いた端末．モバイル回線，OS は **3.2** の計測と同一とした．評価する模擬アプリケーションの動作仕様を表 3 に示す．

通信アプリ A, B はそれぞれ Facebook と Twitter の動作を，位置情報取得アプリは Lifelog の動作をそれぞれ一部模擬するように設定している．通信アプリ A, B ではアラームが実行されると実験用に用意したサーバとの間で TCP によるデータパケットの送受信を行う．TCP コネクションはアラームが初回に実行される際に確立し，以降は同じコネクションを継続して使用する．両アプリとも，送受信するパケット長は表に示す固定値とする．位置情報アプリはアラームが実行されると GPS 情報を取得する．各々において，アラームの登録メソッド，アラーム実行間隔の値は表 2 で示した模擬対象のアプリケーションと同一に設定

表 3　模擬アプリケーションの仕様
Table 3　Specification of imitated applications.

アプリケーション	通信アプリ A	通信アプリ B	位置情報取得アプリ
模擬対象	Fackbook	Twitter	Lifelog
アラーム実行時の動作	サーバへのパケット送信	サーバへのパケット送信	GPS 情報の取得
パケット長	送信：1kB 受信：1kB	送信：1kB 受信：4kB	——
初回起動時刻	OS 起動の 2 分後	OS 起動の 4 分後	OS 起動の 3 分後

した．

この評価では，通信アプリ A, B のアラーム実行間隔が 15 分，位置情報取得アプリのアラーム実行間隔が 1 分であり，提案方式の効果は A と B の起動時刻の差 d の値に依存する．例えば $d = 0$ の場合，特に制御を行わなくとも通信アプリ A, B のアラームが常に一括実行されるため提案方式の効果は現れない．d が位置情報取得アプリのアラーム実行間隔である 1 分以上である場合は，従来方式では A と B のうち初回実行時刻の早い方のアプリのアラームが常に位置情報取得アプリのアラームと一括実行され初回実行時刻の遅い方のアプリのアラームが単独実行されるのに対して提案方式では A と B のアラームが常に一括実行されるため，消費電力低減効果が必ず現れる．この場合，消費電力低減の度合いは d の値によらず一定である．一方，d が 1 分未満である場合は，A と B のアラーム設定時刻の間の時刻に位置情報取得アプリのアラームが設定される状況で提案方式を実行すれば消費電力低減効果が現れるが，この状況が必ず生じるとは限らず，生じる 9 確率は d の値に応じて小さくなり $d = 0$ では上述のように効果は全く現れない．

事前の予備実験において d の値を種々変化させて評価を行い上記の特性を確認した．これを踏まえ，以下の評価においては全て d を 1 分以上の値に設定した．実際の状況では，d の値は端末起動時の各アプリの起動時刻の差によって決定されることになる．したがって提案方式により効果が現れる確率は OS 及びアプリの起動負荷と端末処理能力との関係で決定され，その定量評価を行うことは今後の課題である．

消費電力の測定では，バッテリーを外してバッテリー接続端子に直流電源を接続し，0.1 秒ごとに消費電力を測定した．端末起動時から一定時間経過した時刻を 0 として 6 時間計測を行って 1 回の試行とし，5 回の試行を行った．

4.1.2 評価結果

端末の消費エネルギー特性を図 5 に示す．

プロットは 5 回の試行の平均値であり，参考値として標準偏差をエラーバーで示している．また各方式について平均値の線形近似を破線で示している．

線形近似の相関係数はいずれも 0.9 以上であり近似は妥当であると判断できる．近似式における経過時間 T の係数で表される消費電力は従来方式が約 0.252[W]，提案方式は約 0.224[W] で，約 12.4%の消費電力低減効果が確認された．

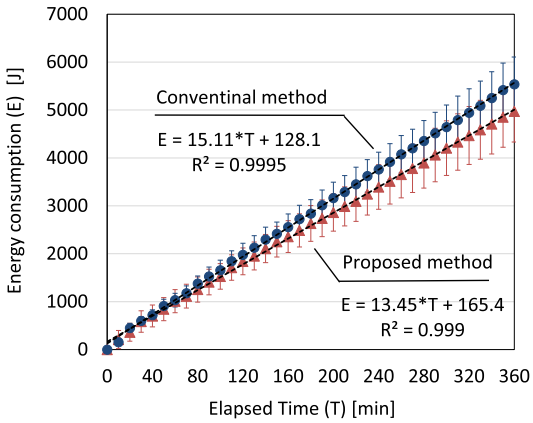

図 5 経過時間に対する端末の消費エネルギー特性
Fig. 5 Terminal energy consumption versus elapsed time.

表 4 IdlePeriod の時間長特性
Table 4 Characteristics of IdlePeriod time length.

	従来方式 a	提案方式 b	比率 b/a
1 回の IdlePeriod の平均時間長 [sec]	317	440	1.4
IdlePeriod の総時間長 [sec]	21532	21550	1.0

また，パケットの送受信が行われないひとまとまりの時間を IdlePeriod と定義し，IdlePeriod の総時間長と，1 回の IdlePeriod の平均時間長を計測した結果を表 4 に示す．提案方式は従来方式と比較し IdlePeriod の総時間長には変化がないが一回の IdlePeriod の時間長を約 40%増加させている．IdlePeriod の時間長は通信モジュールがアイドル状態である時間とほぼ対応している．

表 5 には各方式において一つのバッチで実行されたアラーム数の特性を，そのバッチに含まれる通信を行うアラーム及び通信を行わないアラームの各々の数に応じて分類して示している．「総バッチ数」は分類に当てはまるバッチの数を 1 回の試行あたりの平均値 (すなわち 5 回の試行の合計値を 5 で除した値) で示しており，「一括実行アラーム数」は一つのバッチで実行されるアラーム数の平均値を示している．提案方式では，通信を行うアラーム数が二つ以上一括実行されたバッチの数が従来方式の約 3.2 倍である一方で，通信を行うアラーム一つと通信を行わないアラームが一つ以上がまとめて実行されたバッチの数は約 1.8%にまで減少している．

以上より，提案方式は通信を行う複数のアラームを

表 5　一括実行されるアラーム数で分類したバッチの特性
Table 5　Characteristics of batches classified by the number of executed alarms.

バッチの分類			従来方式 a	提案方式 b	比率 b/a
	$N_a \geq 2$	総バッチ数	7.40	23.8	3.2
		一括実行アラーム数	4.11	3.60	0.87
	$N_a = 1$ and $N_b \geq 1$	総バッチ数	33.4	0.600	0.018
		一括実行アラーム数	3.27	2.33	0.71
	$N_b \geq 2$	総バッチ数	166	185	1.1
		一括実行アラーム数	2.26	2.33	1.0
	$N_a = 1$ or $N_b = 1$	総バッチ数	587	588	1.0

N_a：そのバッチに含まれる，通信を行うアラーム数
N_b：そのバッチに含まれる，通信を行わないアラーム数

表 6　実アプリケーション実行時の IdlePeriod 時間長特性
Table 6　Characteristics of IdlePeriod time length while running real applications.

	従来方式 a	提案方式 b	比率 b/a
1 回の IdlePeriod の平均時間長 [sec]	61.1	75.6	1.2
IdlePeriod の総時間長 [sec]	7076	7103	1.0

一括実行するようにバッチを設定することで，IdlePeriod を大きく増加させ，消費電力の低減をもたらすことが確認された．

4.2　実アプリケーションを用いた性能評価

本節では，**4.1.2** で示した評価結果の中で，IdlePeriod の時間長の特性に関して実際のアプリケーションを対象に評価を行った．評価対象のアプリケーションは Facebook，Twitter，Lifelog とし，**4.1.1** と同一の仕様で構成した端末にこれらのアプリケーションをインストールして **4.1.1** で述べた模擬アプリケーションの実験と同一の方法で計測を行った．ただし試行回数は 1 回とした．評価結果を表 6 に示す．

模擬アプリケーションの結果に比べ，両方式とも 1 回の IdlePeriod の平均時間長は約 1/5 に，IdlePeriod の総時間長が約 1/3 に，それぞれ減少している．これは Facebook と Twitter において模擬アプリケーションでは設定していないサーバからのプッシュ通信があり，その都度通信モジュールがスリープからアクティブに状態遷移したことが主な理由である．両方式の比較においては，提案方式では 1 回の IdlePeriod の時間長が従来方式よりも約 20%増加しており，模擬アプリケーションでの結果に比べて改善幅は小さいものの，消費エネルギーが削減されることが示された．

5.　関連研究及び関連技術

LTE/3G において通信が終了した後も通信モジュールが一定期間オンのまま継続する現象は “radio tail” と呼ばれ，バックグラウンド処理の通信を制御することによる消費電力低減の研究が行われてきた．[7], [8] では過去のパケット送受信パターンからアプリケーションの通信タイミングを予測し，通信モジュールがアイドル状態へ遷移するタイマ値を標準より短く設定する Fast Dormancy と呼ばれる機能を用いて消費電力を低減する手法を提案している．[9] では画面がオンである状態でアプリケーションがフォアグラウンドになる頻度からそのアプリケーションの重要度を判定し，重要でないと判定されたアプリケーションについては画面がオフの状態で行われるバックグラウンドでの通信を遮断することで消費電力を低減する手法を提案している．[10] では，アプリケーションの起動頻度を元に「不要アプリ」と「必要アプリ」を判別し，不要アプリの通信を選択的に遮断する一方で，必要アプリに対しては上述と同様に Fast Dormancy の機能を利用して消費電力を低減するとともに制御信号数を削減する手法を提案し，実機で実際のアプリケーションを動作させる実験により有効性を示している．Fast Dormancy は消費電力削減効果が高いが，動作には端末と網側の両方が対応している必要がある．これに対して提案手法の実施には端末の OS ソフトウェアの改修が必要であるものの，網側では構成を変更する必要がない．また 3GPP のリリース 8 では Fast Dormancy は端末の必須機能と規定されているが網側の対応はオプションとなっており，現時点では Fast Dormancy を用いた方式が有効性を発揮する環境は限定されると考えられる．更に，アプリケーションの通信を遮断するとユーザ品質に影響が及ぶ可能性があり，その実行の可否は

アプリケーション個別の機能を見て判断する必要がある．[10] ではユーザの利便性を低下させうる例として利用頻度の低いメーラやニュースリーダ等が通信制御の対象となりプッシュ通知を受信できないケースを挙げ，その対策として通信を遮断されたアプリの名前を画面の通知領域にリアルタイムに表示してユーザが遮断の対象から外すことができるようにしている．これに対し本論文で提案する手法は通信の遮断等の制御を行わないためユーザに新たな操作を要求しない点で優れている．

[11] では上りリンクの複数のパケットの送信タイミングを調整し集約することで通信モジュールのアイドル状態である時間を増加させる手法を提案している．コンセプトは本論文の提案方式と一部共通しているが，送信バッファに入ったパケットの送信タイミングを調整する手法であり，その前段で行われる，パケットを生成し送信バッファに入力する処理は各タスクごとに行われ集約されない．これに対し提案方式はパケットが生成され実際に送信されるまでの一連の処理をバッチを用いて集約することから，CPU がアイドルまたはスリープの状態にある継続時間を増加させる効果があると考えられる．また，この文献の性能評価では電力消費モデルを用い実トラヒックデータを入力とする計算で有効性を示しているのに対して，本論文では実機で評価を行っている点が異なる．

本論文に関連する既存技術としては Android 6.0 から導入された Doze モードがある．Doze モードではセンサーやタイマーなどを用いてユーザが端末を利用していないと判断すると端末がディープスリープ状態となる [12]．この状態ではアラームの実行はいったん中断され，一定時間後にアクティブ状態となって中断されたアラームを一括処理することで省電力化を図る．Doze モードではユーザがあらかじめホワイトリストに登録したアプリケーションのアラームは中断されず，また Google が管理するプッシュ通知機構である GCM (Google Cloud Messaging) を利用するアプリケーションに対してサーバからメッセージの通知がある場合も Doze モードを抜けて受信が行われる．この手法は通知を行う全てのアプリケーションが GCM を利用すれば，通信の消費電力低減の観点で理想的な特性を示す．しかしながら本論文で着目している利用者の多い SNS アプリケーションは現状では GCM を利用しておらず，その場合 Doze モードではアラームの実行が延期されるため，消費電力の低減と引き替えに通知の遅延が発生する．またホワイトリストに追加されたアプリケーションでは既存の仕組みで通知の受信が行われるため，**2. 2** で述べた消費電力の問題が残る．

6. むすび

本論文では，Android OS のアラームと呼ばれるバックグラウンド処理の予約実行において，高い確率で通信処理を行うことが予想できるアラーム同士を可能な限り同じバッチで一括実行することで通信モジュールのアイドル状態の継続時間を増加させ消費電力の低減を図る手法を提案した．実機を用い模擬アプリケーションと実アプリケーションのそれぞれで評価を行った結果，いずれの条件でも通信モジュールのアイドル状態に対応する IdlePeriod の継続時間が本方式により増加すること，また模擬アプリケーションによる評価において，端末の消費電力が 12%低減することを明らかにした．

今後は，対象となるアプリケーションが登録するアラームの通信処理の有無を監視し，バッチにより一括実行するアラームの対象かどうかを動的に判断する機構を組み込んで評価することが課題である．

文　献

[1] A. Aucinas, N. Vallina-Rodriguez, Y. Grunenberger, V. Erramilli, K. Papagiannaki, J. Crowcroft, and D. Wetheral, “Staying online while mobile: The hidden costs,” Proc. ACM CoNEXT, pp.315–320, Santa Barbara, USA, Dec. 2013.

[2] X. Chen, N. Ding, A. Jindal, Y.C. Hu, M. Gupta, and R. Vannithamby, “Smartphone energy drain in the wild: Analysis and implications,” Proc. ACM SIGMETRICS, pp.151–164, Portland, USA, June 2015.

[3] Android Open Source Project, https://source.android.com/ (参照 2017-08-06)

[4] 3GPP TS 36.304 V8.2.0, May 2008.

[5] 3GPP TS 36.321 V8.0.0, Dec. 2007.

[6] The Open GApps Project, http://opengapps.org/ (参照 2017-02-08)

[7] F. Yu, G. Xue, H. Zhu, Z. Hu, M. Li, and G. Zhang, “Cutting without pain: Mitigating 3G radio tail effect on smartphones,” Proc. IEEE INFOCOM pp.440–444, Turin, Italy, April 2013.

[8] P.K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V.N. Padmanabhan, and G. Varghese, “RadioJockey: Mining program execution to optimize cellular radio usage,” Proc. ACM MobiCom, pp.101–112, Istanbul, Turkey, Aug. 2012.

[9] X. Chen, A. Jindal, N. Ding, Y.C. Hu, M. Gupta, and R. Vannithamby, “Smartphone background activities in the wild: Origin, energy drain, and optimization,”

Proc. ACM MobiCom, pp.40–53, Sept. 2015.
[10] 高木　雅，川原圭博，浅見　徹，“アプリごとのトラフィックとユーザの利用状況を考慮したスマートフォンの通信制御方法,” 情処学論，vol.56, no.3, pp.1121–1131, March 2015.
[11] S. Pradhan, S.K. Dandapat, N. Ganguly, B. Mitra, and P. De, “Aggregating inter-app traffic to optimize cellular radio energy consumption on smartphones,” Proc. IEEE COMSNETS, pp.1–8, Jan. 2015.
[12] Android 6.0 Changes, https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html (参照 2016-07-07)

（平成 29 年 8 月 7 日受付，12 月 15 日再受付，
30 年 1 月 30 日早期公開）

川西　将輝

平 27 芝浦工大・工・通信卒．平 29 同大大学院修士課程了．現在，NTT コミュニケーションズ (株) に勤務．在学中，Android OS のタスクスケジューリングに関する研究に従事．

森野　博章　（正員：シニア会員）

平 6 東大・工・電子卒．平 11 同大大学院博士課程了．平 11 同工学系研究科リサーチアソシエイト，平 13 中央大学研究開発機構・機構助教授，平 17 芝浦工大・工・専任講師，平 20 同准教授，現在に至る．博士（工学）．無線メッシュネットワーク，車々間通信システム，P2P ネットワークなどの研究に従事．博士 (工学)．2013 年国際会議 IS-CANDAR Best Paper Award 受賞．IEEE，情処学会各会員．