修士論文

アプリケーションの性質を考慮した スマートフォンの通信制御 および LTE網のパディングを活用した センサデータ送信手法

2015年2月5日

指導教員　浅見　徹　教授

東京大学大学院情報理工学系研究科
電子情報学専攻　48-136422

高木　雅

内容梗概

音声通話・メール・ブラウジング・カメラ・音楽・ゲーム・決済などの多彩な機能を，手のひらサイズで実現したスマートフォンは，まさに ICT 技術の結晶である．ここ数年でスマートフォンは著しい進化を遂げ，数年前のパソコンに匹敵するほどの高い演算能力と高速な通信環境，また，高機能で多彩なセンサデバイスを手に入れた．このような進化を支えたのは，誰もが自由にハードウェア・OS・アプリを開発できるオープンな環境と，クラウドサービス事業者（Over The Top：OTT）の参入を促した高速な無線ネットワークの存在であった．しかし，OTT の参入によりサービスやコンテンツを掌握できなくなった携帯電話事業者は収益性が低下し，コンテンツの進化に見合うだけのネットワーク設備の増強を続けることが困難になりつつある．

一方で，近年 MVNO（仮想移動体通信事業者）が安価な携帯電話回線を提供するようになったことで，個々のデバイスをインターネットに接続するコストが大幅に低下し，センサネットワークの在り方は大きく変わろうとしている．すなわち，信頼性の乏しいマルチホップ型のローカルネットワークから脱却し，個々のセンサノードをインターネットに直結して，クラウド上にセンサデータを集積することが現実的なコストで実現可能となった．特に，高性能かつ多彩なセンサデバイスを搭載し，国内だけで 6000 万台以上が稼働するスマートフォンは，センサノードとしての利用価値が非常に高いが，現在は十分に活用されないままとなっている．

そこで，本研究では，スマートフォンと携帯電話網が大規模センサネットワークとして活用される時代の到来を見据え，(1) 携帯電話網をスマートフォン上のアプリケーションに合わせて最適化し，ネットワーク設備の利用効率の向上を図ること，(2) 従来の参加型センシングにおいてユーザに参加を躊躇わせる原因となっていた，データ送信時のスマートフォンの消費電力および通信量の増加を，ゼロないし無視できる水準に抑えること，(3) スマートフォンのセンサを活用したセンシング技術を確立し，センサデバイスとしての有用性を示すこと，の 3 つの課題に取り組んできた．

課題 (1) では，スマートフォン利用者 1,800 人の協力を得て，延べ 78,200 アプリの通信挙動の分析を行い，ユーザやアプリによって通信間隔の分布が大きく異なることを明らかにした．これを受け，起動中のアプリ一覧と各アプリの通信間隔の分布データから，時々刻々と変化する通信の発生確率をリアルタイムで予測する手法を提案し，さらに，制御信号と消費電力のバランスを評価する指標として「アイドルタイマー効率」を定義して，これが最大となるアイドルタイマーを選択することで，制御信号と消費電力を効果的に削減する手法を提案する．シミュレーション評価では，制御信号を最大 86.1%，通信モジュールの消費電力を最大 37.6%，同時に削減できることを確認した．

課題 (2) では，LTE 網の無線区間に最小割り当て単位が存在することに着目し，MAC レイヤのパディング部分にセンサデータを埋め込んで送信することで，ユーザ端末の消費電力や通信量を増加させることなく，センサデータを送信する手法を提案する．108 ユーザ，81.09GB 分の実通信データに基づく評価では，全通信量の 1.13%にあたるセンサデータを付加できることを確認した．従って，毎月 7GB のデータ通信を行うユーザの場合，毎月 79.1MB のセンサデータを付加できる．

課題 (3) では，高い静粛性ゆえに交通事故が多発している電気自動車 (EV) とハイブリッド車 (HV) について，車両の接近をスマートフォンで事前に検知する技術を提案する．本手法では，モータのスイッチング雑音に着目し，機械学習を用いて EV を 92.8%，HV を 82.2%の精度で検知する．

目 次

第 1 章 序論 **1**
1.1 本研究の背景 2
1.1.1 携帯電話事業者の疲弊と MVNO の台頭 2
1.1.2 高機能センサとしてのスマートフォン 2
1.2 本研究の目的 2
1.3 本論文の構成 3

第 2 章 アプリケーションの性質を考慮したスマートフォンの通信制御手法 **5**
2.1 はじめに 6
2.2 スマートフォンの実際の利用状況 7
2.2.1 調査方法 7
2.2.2 トラヒック 7
2.2.3 通信時の消費電力 11
2.3 アプリ単位の通信制御 12
2.3.1 不要アプリの制御 12
2.3.2 必要アプリの制御 13
2.3.3 不要アプリの制御機構 15
2.3.4 必要アプリの制御機構 16
2.4 制御信号と消費電力の削減量の評価 17
2.4.1 不要アプリに対する性能 17
2.4.2 必要アプリに対する性能 17
2.5 関連研究 20
2.6 まとめ 21

第 3 章 LTE 網のパディングを活用したセンサデータ送信手法 **22**
3.1 はじめに 23
3.2 LTE ネットワークの仕様 23
3.2.1 データリンク層（レイヤ 2） 24
3.2.2 物理層（レイヤ 1） 25
3.3 利用可能な MAC 層パディングの量 26
3.3.1 実態の調査 26
3.3.2 パディング量の推定 27
3.4 データの埋め込み方式 27
3.5 応用例 28
3.6 関連研究 28

第 4 章　スマートフォン・センシングの活用例：電気自動車の接近検知　29
4.1　はじめに 30
4.2　モータユニットのスイッチング雑音 31
4.2.1　スイッチング雑音の発生原理 31
4.2.2　スイッチング雑音の分析 32
4.3　検知手法 33
4.3.1　特徴量ベクトル 33
4.3.2　ラベリング 35
4.3.3　教師あり学習 35
4.4　評価 36
4.4.1　EV の検知性能 36
4.4.2　EV と HV の検知性能比較 37
4.5　他の検討事項 38
4.5.1　スマートフォンの位置と検知精度 38
4.5.2　False Negative の低減 39
4.5.3　フレーム長と検知精度，消費電力の関係 39
4.6　Android 端末と Google Glass への実装 40
4.7　関連研究 41

第 5 章　結論　42
5.1　本論文のまとめ 43
5.2　本研究の主たる貢献 43
5.3　今後の課題 44

謝辞　46

付録　52

図 目 次

1.1 従来のセンサネットワークと本研究で提案するセンサネットワーク 3

2.1 スマートフォンにインストールされているアプリ数の分布 8
2.2 上位 16 アプリの通信間隔の累積分布関数 (CDF) 9
2.3 スマートフォンの稼働時間に対する画面点灯時間の割合のヒストグラム 10
2.4 トラヒックのうち画面点灯中に発生したものの割合のヒストグラム 10
2.5 ユーザ操作と通信発生確率の関係 10
2.6 米国の携帯電話キャリアにおける RRC ステートマシンの一例 [11] 11
2.7 携帯電話網を通して通信を行った際の GALAXY Nexus (SC-04D) の消費電力 . . 11
2.8 通信間隔分布 (CDF) からのアイドルタイマーの決定手順 14
2.9 従来方式を 100%とした，画面点灯時の提案手法の性能評価 19
2.10 従来方式を 100%とした，画面消灯時の提案手法の性能評価 19
2.11 アイドルタイマー効率の基準値を変化させた時の性能評価 20
2.12 ユーザ操作のみを考慮した場合の提案手法の性能評価 20

3.1 LTE における送信パケットの処理フロー（右上部分が提案手法） 23
3.2 LTE の無線フレーム構成 [24] 24
3.3 サブフレーム構成と RB の配置 [24] 24
3.4 送出パケットサイズの分布 27
3.5 無線区間 TB サイズの分布 27

4.1 提案するシステムのコンセプト 30
4.2 ハイブリッド車とガソリン車の走行音量の比較 31
4.3 計測実験を行った環境 32
4.4 走行音の大きさとモータの回転速度の関係 33
4.5 EV（日産リーフ）のスペクトログラム 34
4.6 HV（トヨタ プリウス PHV）のスペクトログラム 34
4.7 検知アルゴリズムのフローチャート 35
4.8 さまざまな車速の EV の録音データでの判定精度 36
4.9 EV 接近前に警告通知が出される累積確率 36
4.10 EV・HV 接近前に警告通知が出される累積確率の比較 37
4.11 計測中のピンマイクの装着位置 38
4.12 車両接近前に警告通知が出される累積確率 38
4.13 False Negative 低減時の判定精度 39
4.14 フレーム長と判定精度の関係 39
4.15 スマートフォンへの実装 40

5.1 VpnService API の標準的な利用法の概要図 54
5.2 VpnService API を活用したパケットキャプチャの概要図 55

表 目 次

2.1 コネクション数の上位 16 アプリ 8

3.1 変調方式と TB サイズの関係（上りリンク） 25
3.2 RSSNR と変調方式 (MCS) の対応関係 26

4.1 EV の 20kHz 周辺の f_0 32
4.2 周波数セグメント 35
4.3 さまざまな車速の EV と HV の録音データでの判定精度 37

5.1 機種別の LTE 通信パラメータ取得可否 53

第1章

序論

1.1　本研究の背景

1.1.1　携帯電話事業者の疲弊と MVNO の台頭

ここ数年，スマートフォンの普及に伴って携帯電話網を流れるデータトラヒックは指数関数的な増加を続けてきた．しかしながら，携帯電話事業者はパケット定額制料金を採用してきたため，次第に収益性が低下し，トラヒックの急増に見合うだけのネットワーク設備の増強を続けることが困難になりつつある．2012 年には国内最大手の携帯電話事業者である NTT ドコモのネットワークで，2013 年には国内 2 位の携帯電話事業者である KDDI のネットワークで，大規模な通信障害が発生するに至っている．さらに，2014 年に導入された携帯電話大手 3 社の新料金プランではパケット定額制料金が廃止されるなど，携帯電話事業者は疲弊の色を隠し切れない．

一方で，携帯電話事業者から回線設備を借り受けて移動体通信サービスを提供する MVNO（仮想移動体通信事業者）は，回線の卸価格が低下したこともあり，近年の台頭が目立つ．MVNO が月額 500 円程度でデータ通信回線を提供するようになったことで，LTE/3G 通信モジュールの運用コストは大幅に低下し，これを利用した新たなサービスが出現した．例えば，カーシェアリングサービスでは，貸し出し車両 1 台 1 台に 3G 通信モジュールを搭載し，ドアロックの解錠から，ガソリン残量の管理，返却状況の確認までをオンラインシステムから遠隔操作で実施する．また，2014 年に東京電力が本格導入を開始したスマートメータは，一部地域において 3G 通信モジュールを搭載しており，30 分毎の電力使用量を収集する [1]．将来的には，電力供給量に合わせて家電の稼働状況を制御する機能も搭載される予定である．

このようなサービスは LTE/3G 通信モジュールの運用コストが高価だった時代には考えられなかったものであり，携帯電話網を取り巻く環境はここ数年で大きく変化している．携帯電話網をより安定的に，効率よく運用するためには，アプリケーションやサービスの変化に応じて最適化を図る必要がある．

1.1.2　高機能センサとしてのスマートフォン

近年のスマートフォンには，さまざまな入出力デバイスが搭載されている．入力デバイスとしては，カメラ，マイク，タッチパネル，GPS，磁気センサ，加速度センサ，ジャイロセンサ，温度センサ，湿度センサ，気圧センサ，照度センサ，近接センサ，指紋センサ，脈拍センサが挙げられる．出力デバイスとしては，ディスプレイ，スピーカー，バイブレータが，通信デバイスとしては，Wi-Fi，Bluetooth，NFC，赤外線通信，3G/LTE/WiMAX，デジタル TV，FM ラジオ／トランスミッタが搭載されている．これらのデバイスは十分に高性能であり，センサノードとして利用価値が非常に高いが，現在は十分に活用されないままとなっている．スマートフォンは国内だけで 6000 万台以上が稼働しており，日本全国に広く分布するなど，センサデバイスとして活用しやすい特徴を持つ．これらの端末を，既に接続している携帯電話網を通して，あるいは，MVNO の安価なデータ通信回線と組み合わせることで，超大規模なセンサネットワークを構築することが可能である．

1.2　本研究の目的

本研究の最終的な目標は，世界中のスマートフォンと携帯電話網を取り込んで世界規模のセンサネットワークを構築し，クラウドサービスから現実世界を「知覚」するためのインフラを提供する

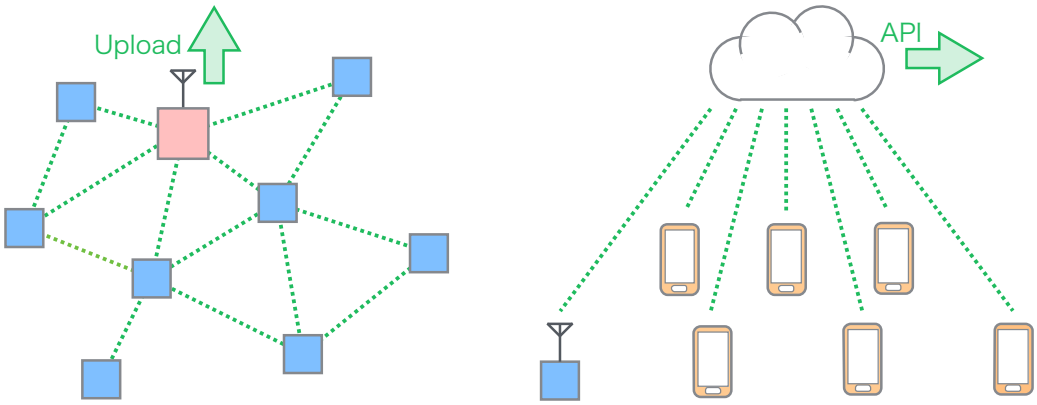

図 1.1: 従来のセンサネットワークと本研究で提案するセンサネットワーク

ことにある．センサネットワークを通して収集したセンサデータを，API を通して様々なクラウドサービスに提供することで，クラウドサービスのより一層の発展を促す．

従来のセンサネットワークでは，ローカルエリアのマルチホップネットワークを構築し，代表ノードにセンサデータを集約してから送信するのが一般的である（図 1.1 左）．この方式では，代表ノードが単一障害点（Single Point of Failure：SPOF）となるばかりでなく，中継ノードにトラブルが生じるとそれ以下の末端ノードが利用不能になるなど，信頼性に乏しい．

一方，本研究で提案するセンサネットワークは，スマートフォンをセンサノードとして採用し，1 台 1 台が携帯電話網を通してインターネットに直結し，個別にセンサデータを送信する（図 1.1 右）．この方式では，単一障害点は存在せず，各ノードの側にユーザが居るため，異常が発生した場合でも対処しやすい．また，既存のデバイスおよびネットワークを利用するため，センサネットワークの構築コストが低いという特徴もある．さらに，ユーザの移動により形状がひとりでに変化し，適当なインセンティブを与えることで所望の形状にすることも可能である．

本研究では，スマートフォンと携帯電話網が世界規模のセンサネットワークとして活用される時代の到来を見据え，(1) 携帯電話網をスマートフォン上のアプリケーションに合わせて最適化し，ネットワーク設備の利用効率の向上を図ること，(2) 従来の参加型センシングにおいてユーザに参加を躊躇わせる原因となっていた，データ送信時のスマートフォンの消費電力および通信量の増加を，ゼロないし無視できる水準に抑えること，(3) スマートフォンのセンサを活用したセンシング技術を確立し，センサデバイスとしての有用性を示すこと，の 3 つの課題に取り組む．本稿では，各課題に対して解決策を提示し，実環境で取得したデータを用いて評価を行う．

1.3　本論文の構成

本論文は以下の各章により構成される．

第 1 章 序論

第 2 章 アプリケーションの性質を考慮したスマートフォンの通信制御手法

第 3 章 LTE 網のパディングを活用したセンサデータ送信手法

第 4 章 スマートフォン・センシングの活用例：電気自動車の接近検知

第 5 章 結論

2 章では，1800 ユーザ分の通信統計を分析し，ユーザやアプリによる特徴の差異を明らかにした上で，起動中のアプリ一覧と各アプリの通信間隔の分布データから，時々刻々と変化するトラヒックの性質をリアルタイムで予測し，アイドルタイマーに反映する手法を紹介する．3 章では，LTE 網の無線区間に最小割り当て単位が存在することに着目し，MAC レイヤのパディング部分にセンサデータを埋め込んで送信することで，ユーザ端末の消費電力や通信量を増加させることなく，センサデータを送信する手法を紹介する．4 章では，スマートフォンのセンサデバイスとしての有用性を示すため，高い静粛性ゆえに交通事故が多発して社会問題となっている EV と HV について，車両の接近をスマートフォンを用いて検知する技術を紹介する．最後に，5 章で本論文の内容をまとめる．

第2章

アプリケーションの性質を考慮したスマートフォンの通信制御手法

2.1　はじめに

近年の携帯電話システムは，大きく 3 つの問題に直面している．1 つ目の問題は，スマートフォンの普及に伴ってデータトラヒックが急増し，都市部を中心に実効スループットが低下していることである．スマートフォンのユーザはフィーチャーフォンのユーザと比較して，平均で 10 倍以上のデータトラヒックを発生させると言われており，エリクソンの報告によると，ここ数年間，携帯電話網を流れるデータトラヒックは年間 2 倍のペースで指数関数的に増加を続けてきた [2]．今後もデータトラヒックは同様のペースで増加を続けると予想されており，2020 年までに 2013 年実績の 100 倍に達する計算になる．実効スループットの低下はユーザの体感品質を大きく低下させるため，携帯電話事業者にとって実効スループットの改善は急務である．対策として，基地局の小セル化による単位面積当たりの収容能力の向上や，Wi-Fi スポット整備による固定網へのオフロードを促進しているものの，基地局の新設には計画から運用開始まで半年から 1 年という長い時間が必要であり，Wi-Fi でのオフロードにはカバーできるエリアや効果のあるユーザ層が限定されるという課題がある．

2 つ目の問題は，データトラヒックの急増に伴って制御信号が急増し，コアネットワークの処理能力を瞬間的に上回る事態が発生していることである．常時のインターネット接続を前提とするスマートフォンは，フィーチャーフォンより高頻度で通信を行い，より多くの制御信号を発生させることが知られている．実際に，スマートフォンの普及に伴って，2009 年から 2012 年までの 3 年間で制御信号は 5 倍に増加している [3]．2012 年には，国内最大手の携帯電話事業者である NTT ドコモのネットワークにおいて，制御信号の量がコアネットワークの処理能力を上回ったことによる大規模な通信障害が発生している [4]．ひとたび通信障害が発生すると，数百万人のユーザが携帯電話サービスを利用できなくなるため，早急な対策が求められる．

3 つ目の問題は，スマートフォンのバッテリ駆動時間がフィーチャーフォンと比べて著しく短いことである．フィーチャーフォンが 1 回の充電で 1 週間使えることも珍しくないのに対し，ほとんどのスマートフォンは 2 日に 1 度以上の頻度で充電する必要がある．さらに，フィーチャーフォンの平均的なバッテリ容量が 1000mAh 未満，スマートフォンの平均的なバッテリ容量が 2000mAh 台であることを踏まえると，消費電力の差には 10 倍近い開きがある．Carroll らによると，スマートフォンの消費電力の約 85%は，ディスプレイ，通信モジュール（LTE，3G，Wi-Fi），CPU の 3 点が占めており，通信モジュールの消費電力はトラヒックや制御信号の量と密接な関係にある [5]．

本章では，スマートフォン上で利用可能なさまざまな情報を活用し，スマートフォン上で，ユーザに不要な通信を間引いたり，アイドルタイマーを動的に設定したりすることで，携帯電話網を流れるデータトラヒックと制御信号，端末の消費電力を効果的に削減する手法を提案する．具体的には，アプリの起動頻度，通知の頻度，ウィジェットの利用状況，画面の点灯状態，実行中のアプリ，各アプリの通信間隔の分布，ユーザ操作からの経過時間，といった情報を活用して LTE/3G 接続時に適切な通信制御を行う．

本章の構成は以下のようになっている．まず，2.2 節では実際に稼働中の Android 端末から収集したトラヒックの統計的な性質や，通信中のスマートフォンの消費電力の性質を紹介する．次に，2.3 節で本章の提案手法であるアプリ単位での通信制御について詳しく述べた後，2.4 節で提案手法の性能評価を行う．さらに，2.5 節で関連研究について触れ，2.6 節で本章の内容をまとめる．

2.2　スマートフォンの実際の利用状況

2.2.1　調査方法

スマートフォンが生み出すトラヒックの性質は，インストールされているアプリの組み合わせに大きく依存する．2014 年現在，Google PLAY では 100 万本以上のアプリが公開されており，それぞれ発生させるトラヒックの量や頻度が大きく異なると考えられる．先行研究でも，実ユーザのトラヒックを調査した例は存在するが，我々が着目している，アプリ毎のトラヒックの性質の違いを明らかにするような公開データの存在は確認できなかった [6, 7, 8, 9]．そこで我々は，その実態を調査するため，トラヒックの統計データを収集する Android アプリ「通信量お知らせアプリ」を製作し，2013 年 4 月より Google PLAY で公開して協力者を募ってきた [10]．2014 年 6 月現在，1,800 ユーザ，のべ 78,200 アプリ分のデータを収集しており，ユーザ数は現在も増加を続けている．この「通信量お知らせアプリ」は，

- インストール済みアプリの名称とパッケージ名
- 各アプリの累計通信量（送受信，画面状態別）
- 各アプリの累計通信回数（送受信，画面状態別）
- 各アプリの通信間隔の分布（送受信，画面状態別）
- 通信発生時の各アプリの実行状態

といった統計情報を記録し，週に 1 度，我々のサーバへ送信する．なお，データの重複管理に必要となる端末固有 ID，機種名，OS バージョンの情報を除いては，ユーザに関する情報は一切収集していない．それゆえ，ユーザの年齢層や性別などは不明である．ただし，Google PLAY の統計情報によると，ユーザの 96.87%は日本国内におり，通信キャリア別では，NTT ドコモが 58.9%，au が 17.7%，SoftBank が 7.5%を占めている [10]．この比率は，Google PLAY における各通信キャリアのシェア比率（4.4%，1.4%，0.5%）とほぼ一致している．

2.2.2　トラヒック

全体の概要

図 2.1 は，各ユーザのスマートフォンにインストールされていたアプリ数の分布である．1,800 ユーザでの平均は 58 個，最大は 629 個であり，一般的なユーザのスマートフォンには，40〜50 個のアプリがインストールされていると考えて差し支えない．

表 2.1 は，収集した全ユーザの合計のトラヒックにおいて，コネクション数の多い上位 16 アプリとそのユーザ数を示した表である．ここでは「コネクション」という単語を便宜上「途中に 1 秒以上の間隔のない一続きの通信」という意味で用いており，TCP における「コネクション」とは定義が異なる．表 2.1 において特筆すべきは，100 万本以上あるアプリの中で，コネクション数の分布が特定のアプリ群に著しく偏っており，上位 10 アプリが全コネクション数の 50%を占めていることである．特に，1 位の「LINE」は単独で全コネクション数の約 6 分の 1 を占めており，携帯電話網を流れるトラヒックの性質に大きな影響を与えている．「LINE」は，主にテキストチャットを行うアプリであり，少量の通信を高頻度で行うため，制御信号数やユーザ端末の消費電力に影響を及ぼしやすい．2 位の「Google 連絡帳の同期」は，Android 標準の連絡帳アプリのデータ同期

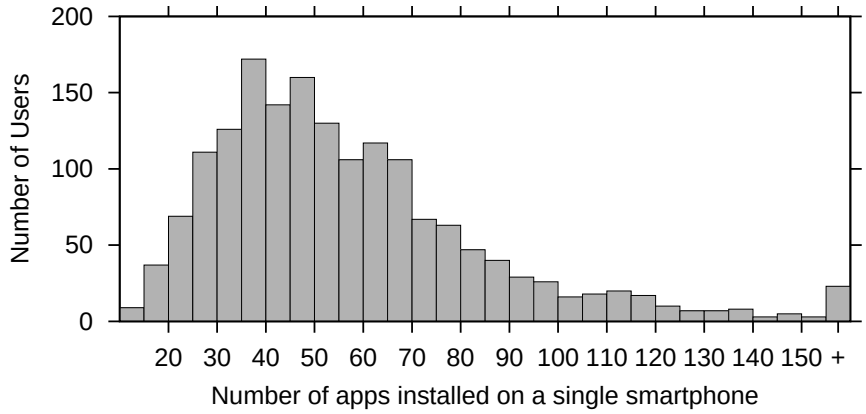

図 2.1: スマートフォンにインストールされているアプリ数の分布

表 2.1: コネクション数の上位 16 アプリ

App Name	Connection#		User#
LINE	10,880,076	16.3%	1,116
Google Sync.	6,034,994	9.0%	1,471
Browser	4,317,009	6.4%	1,235
Facebook	3,333,681	5.0%	956
Twitter	2,740,469	4.1%	649
Google Search	2,149,407	3.2%	1,471
Simeji	1,334,526	2.0%	224
YouTube	1,220,967	1.8%	1,483
Google PLAY Store	1,180,899	1.8%	1,539
Chrome	1,017,815	1.5%	979
Yahoo!	1,012,369	1.5%	388
Google Map	848,033	1.3%	1,248
Google+	707,741	1.1%	1,087
Popacon(LINE)	599,103	0.9%	212
Messenger(Facebook)	525,953	0.8%	140
Virus Buster	500,287	0.7%	174

を司るシステムアプリで，1 位の「LINE」と合わせると，全コネクション数の 25%を占める．他には,「Twitter」や「Facebook」といった SNS アプリ,「Chrome」や「Google Map」といった情報閲覧系アプリが，多くのユーザを獲得して上位にランクインしている．一方で,「Simeji」や「ウイルスバスター」といったアプリは，ユーザ数の少ない割にコネクション数が多い．従って，トラヒックをユーザ単位で考える際には，これらのアプリの影響が相対的に大きくなる．

各アプリの利用状況

図 2.2 は，合計コネクション数の多い上位 16 アプリについて，通信間隔の累積分布関数を示したものである．通信間隔の中央値は,「ブラウザ」や「Twitter」などのアプリでは 10 秒前後であるが,「Google+」や「Gmail」などのアプリでは 1,000 秒を超えている．現在の LTE/3G ネットワークにおいて標準的なアイドルタイマー長である 16 秒以内での通信発生確率を見ても，10%〜60%とアプリによって大きく異なる．従って，起動しているアプリの組み合わせによって，スマー

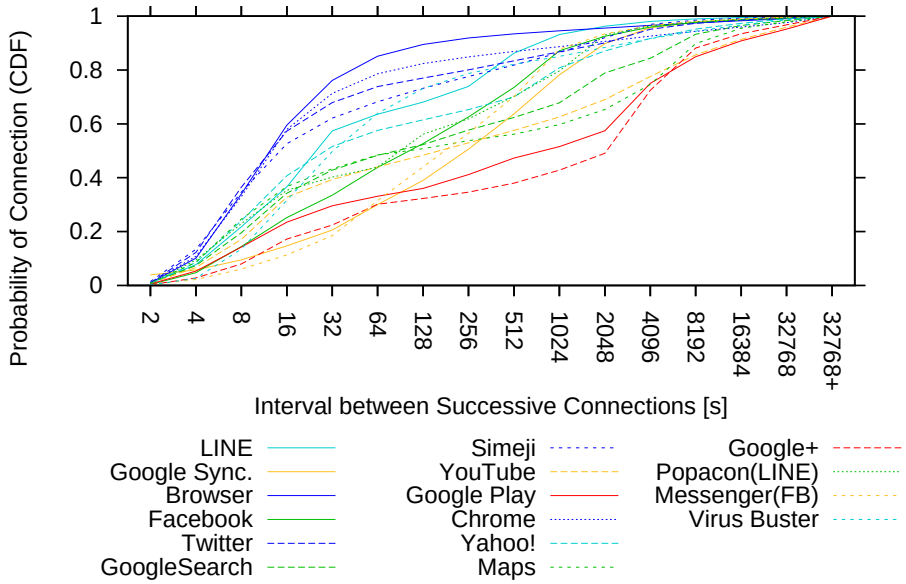

図 2.2: 上位 16 アプリの通信間隔の累積分布関数 (CDF)

トフォンの生み出すトラヒックの性質は大きく変化し，通信制御の戦略にも影響を与える．

画面の点灯状況との関係

前述の「通信量お知らせアプリ」では，各トラヒックの発生時に画面の点灯状況も記録している．図 2.3 は，スマートフォンの稼働時間に対する画面点灯時間の割合の分布である．1,800 ユーザの平均値は 34.0%，中央値は 26.8%であり，一般的なスマートフォンは稼働時間の約 20%しか画面が点灯していない．一方，図 2.4 は，各ユーザのトラヒックのうち画面点灯中に発生したものの割合の分布である．1,800 ユーザの平均値は 84.8%，中央値は 93.6%であり，一般的なスマートフォンではトラヒックの約 95%が画面点灯中に集中していると言える．従って，画面点灯中と画面消灯中では，トラヒックの発生確率に 80 倍近い差があることになる．従って，画面の点灯状態を合わせて考慮することで，性能の向上を期待できる．

ユーザ操作との関係

一般に，ユーザがスマートフォンなどの情報通信機器を操作する場合，タッチパネルやボタンといった入力デバイスの操作タイミングとトラヒックの発生タイミングの間には，強い関連性があると考えられる．この仮説を検証するため，3 人の実験協力者を募り，合計 93 時間のトラヒック計測実験を行った．計測では，GALAXY Nexus SC-04D (Android 4.2.2) と NTT ドコモの 3G 回線を使用し，実験協力者にはなるべく普段通りにスマートフォンを利用してもらった．なお，この実験を前述の「通信量お知らせアプリ」に統合しなかったのは，Google PLAY を通して配布されるアプリの権限では，タッチパネルやボタンの操作状況を常時監視することが技術的に困難なためである．

図 2.5 は，ユーザ操作からの経過時間を横軸に，累計のトラヒック発生確率を縦軸に取ってプロットしたグラフである．あるユーザ操作があった時，1 秒以内に通信が発生する確率は 15%，2

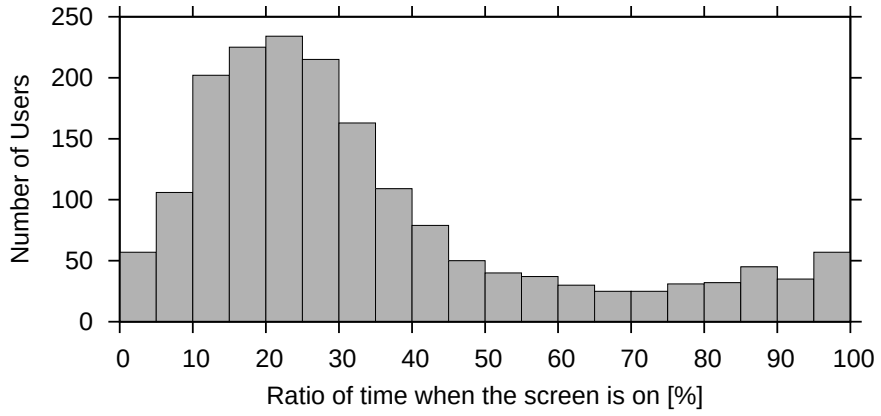

図 2.3: スマートフォンの稼働時間に対する画面点灯時間の割合のヒストグラム

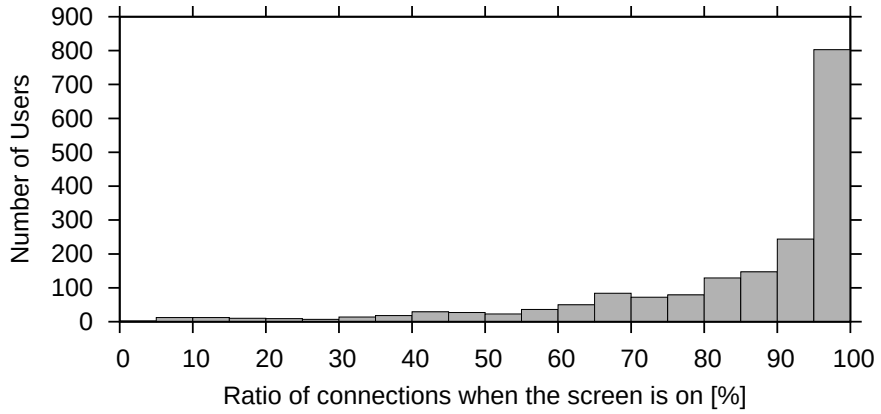

図 2.4: トラヒックのうち画面点灯中に発生したものの割合のヒストグラム

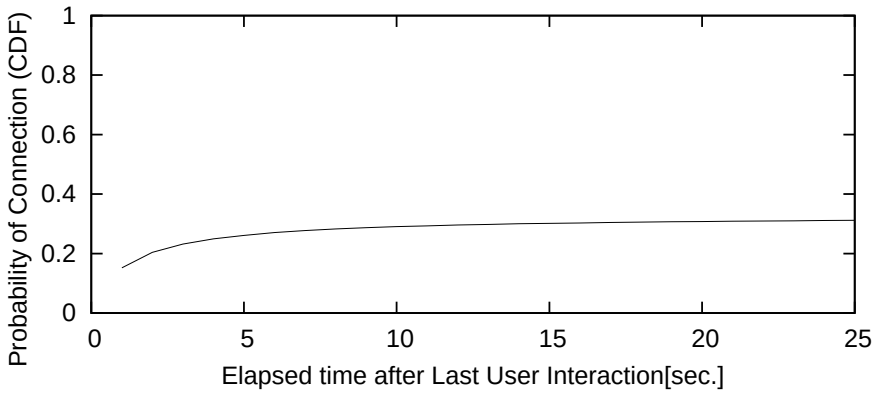

図 2.5: ユーザ操作と通信発生確率の関係

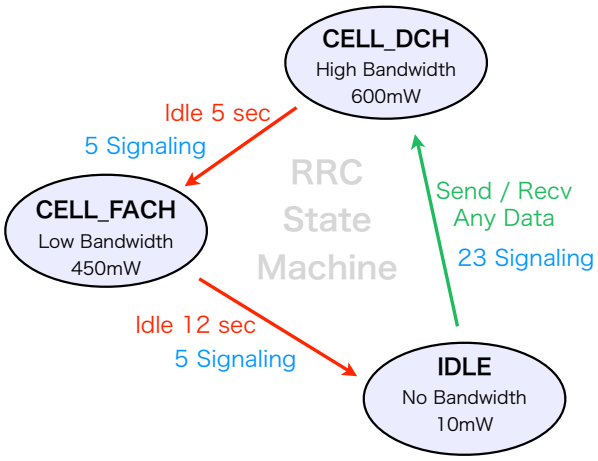

図 2.6: 米国の携帯電話キャリアにおける RRC ステートマシンの一例 [11]

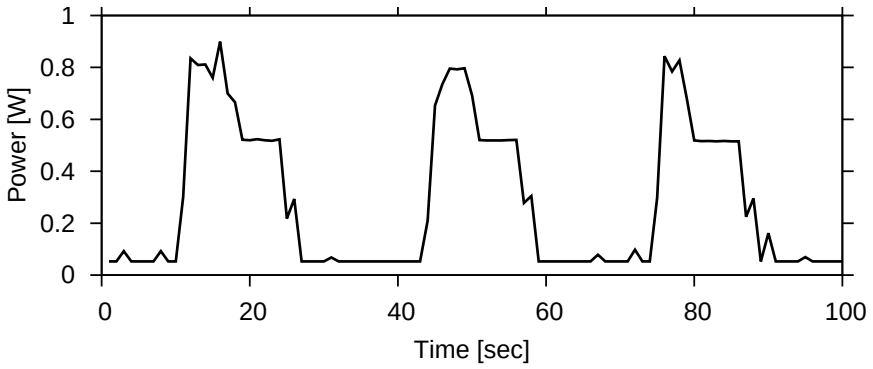

図 2.7: 携帯電話網を通して通信を行った際の GALAXY Nexus (SC-04D) の消費電力

秒以内で 20%，5 秒以内で 26%となっており，ユーザ操作の有無が通信の発生確率を大きく左右する要因であることが確かめられた．従って，ユーザ操作のタイミングを考慮することで，さらなる性能の向上を期待できる．

2.2.3　通信時の消費電力

スマートフォンのモバイルデータ通信による消費電力は，Radio Resource Control(RRC) における接続状態に応じて大きく変化する．RRC では，3G 端末の接続状態として *CELL_DCH*, *CELL_FACH*, *IDLE* の 3 状態が，LTE 端末の接続状態として *CONNECTED*, *IDLE* の 2 状態が定義されており，図 2.6 に示すように通信状態に応じて状態遷移することになっている [12]．*IDLE* 状態への遷移は，無通信状態が一定時間続くことが条件となっており，この制御機構はアイドルタイマーと呼ばれる．このタイマー時間の設定は，携帯電話事業者やユーザ端末の製造メーカに委ねられており，16 秒前後であることが多い．スマートフォンの通信時に電力計測を行うと，通信終了後に消費電力の大きい状態が尾を引く様から，この 16 秒間を “Radio Tail” と呼ぶ．一般に Radio Tail を長くすると，コネクションを再利用できる確率が高まるため，コネクション制御に関わる制御信号数を抑えられるものの，ユーザ端末の消費電力は増大する．反対に，Radio Tail を短くす

ると，ユーザ端末の消費電力は抑えられるものの，コネクションの切断と再確立が増加するため，制御信号数が増加しネットワークへの負荷が増大する．

2.2.2 節で述べた，アプリ毎のトラヒックの性質の違いを踏まえると，どのアプリが起動しているかによって，最適な通信制御の戦略が変わる．通信発生確率が高い状況では，アイドルタイマーを延長して制御信号の削減を狙い，通信発生確率の低い状況では，アイドルタイマーを短縮して消費電力の削減を狙う戦略が有効である．

我々は，実際の端末に設定されているタイマー時間と消費電力を確かめるため，電力計測実験を行った．この実験では，Android のリファレンス端末であり標準的な動作を期待できる GALAXY Nexus SC-04D（Android 4.1.1）と 0.1 Ω抵抗，デジタルマルチメータ Agilent 34410A を用いて消費電力を計測した．図 2.7 は，画面を消灯した状態で，30 秒ごとに数 KB のファイルを送受信した時の消費電力の推移である．図 2.7 では，消費電力は通信開始直後に 0.8W まで急上昇し，約 5 秒後に 0.5W に低下し，さらに約 8 秒後に元の 0.05W まで急落する様子が読み取れる．これらの 3 状態はそれぞれ，RRC の *CELL_DCH*，*CELL_FACH*，*IDLE* に対応しており，この端末のアイドルタイマーは合計 13 秒であることが分かる．本章では，これらの値を用いて提案手法の制御信号および消費電力の削減性能を評価する．

2.3　アプリ単位の通信制御

2.3.1　不要アプリの制御

本章の冒頭で述べたように，スマートフォンにインストールされているアプリの中には，ユーザが必ずしも常には必要としていないものが存在する．具体的には，Android OS に含まれている Google 系サービスのアプリ，メーカや通信事業者がプリインストールしたアプリ，ユーザ自身がインストールしたものの既に使わなくなったアプリ，季節により利用頻度が大きく変化するアプリ，などが該当する．我々の実験では，未使用状態の「Google+」が約 1 時間周期で 100KB 前後の定期通信を行うことが確認されている．

このような不要アプリのトラヒックを選択的に遮断することで，データトラヒックや制御信号，端末の消費電力を効果的に削減できる．ただし，誤ってユーザが利用中のアプリの通信を遮断してしまうとユーザへの影響が大きいため，本手法では以下の条件で随時，不要アプリの自動判定を行い，ユーザがアプリを利用し始めると自動的に制御対象から外れるようにする．利用状況の判定期間は，特定の曜日にだけ利用するアプリの存在を考慮して標準で 1 週間としたが，可変とすることでユーザが省電力と利便性のバランスを調整できるようになる．

- 当該アプリの起動回数が直近 1 週間で 0 回であること
- 通知バーへの通知回数が直近 1 週間で 0 回であること
- 当該アプリのウィジェットを設置していないこと
- ウェアラブル端末での利用がないこと
- Android OS のシステムアプリでないこと

ここでは，ユーザへのインタラクションの有無に焦点を当て，ユーザに何らかの情報提示をすることなく，バックグラウンドで通信するだけのアプリは不要と判定する．ただし，システム領域（`/system/app/`）にインストールされているアプリについては，OS の安定稼働のため制御対象外

とする．これにより，2.3.3 節で述べるように，通信制御中であっても GCM によるプッシュ通知が利用可能となる．

しかしながら，上記の判定条件では，利用頻度の低いメーラやニュースリーダ，ウイルス対策アプリといったアプリが通信制御の対象となる可能性が残る．適切に実装されたアプリであれば，新着情報はプッシュ通知をトリガとして実行し，通信失敗時にはその旨の通知が表示されるため，大きな問題は発生しないが，そうでないアプリも数多く存在する．そこで，通信を遮断されたアプリの名前を通知領域にリアルタイムで表示し，そこから当該アプリを起動して一時的に通信制御の対象から外したり，通信制御の除外リストに追加したりできるようにする．また，直近 10 件ほどの遮断履歴も合わせて表示する．これにより，利用頻度の低いアプリには一時的な通信許可を，利用頻度の高いアプリには永続的な通信許可を，ユーザが手軽に与えることができる．

2.3.2　必要アプリの制御

不要アプリ以外のアプリは，通信を阻害するとユーザへの影響が大きいため，データトラヒックの制御は行わず，アイドルタイマーを状況に応じて変化させることで，制御信号と消費電力のみを削減する．本手法では，アプリ毎の通信間隔の分布と，画面の点灯状況，タッチパネルやボタンへのユーザ操作の有無に基いて，アイドルタイマーを動的に決定する．なお，これらの情報は Linux カーネル上で管理されており，`dumpsys` コマンド等で取得可能である．

必要アプリに対する制御は LTE/3G 接続中のみを対象とし，コネクション管理に制御信号を必要としない Wi-Fi 接続中は制御を行わない．また，Wi-Fi 接続中はアプリの挙動が変化する場合があるため，後述する各アプリのトラヒックの性質調査も停止する．ユーザによっては，Wi-Fi 接続と LTE/3G 接続を頻繁に行き来することがあるが，トラヒックの性質調査に時間が掛かることを除けば，性能面への大きな影響はない．将来的には，Wi-Fi 接続中のアプリの挙動変化や Wi-Fi モジュールの消費電力も考慮し，LTE/3G 接続と Wi-Fi 接続の自動切り替え制御を盛り込んで，制御信号と消費電力のさらなる最適化を検討する．

アプリ毎の通信間隔の分布に基づく制御

本手法では，まず，事前に収集したアプリ毎の通信間隔の分布データ（図 2.2）と実行中のアプリの一覧を用いて，その時点での端末全体のトラヒックの性質を予測する．実行中のアプリ一覧は Android SDK の ActivityManager API で取得し，この一覧に含まれる各アプリの通信間隔の累積分布関数（$CDF_{App\#1}$，$CDF_{App\#2}$，…，$CDF_{App\#N}$）から，以下の計算式を用いて端末全体のトラヒックの通信間隔の累積分布関数 CDF_{Total} を算出する．

$$CDF_{Total}(t) = 1 - \prod_{n=1}^{N} \{1 - CDF_{App\#n}(t)\} \tag{2.1}$$

ここで，t は直近の通信からの経過時間である．なお，各アプリの通信頻度の情報は，分布データに含まれるため，ここで重み付けは必要ない．

次に，得られた端末全体の通信間隔の分布データから，最適なアイドルタイマー長を決定する．アイドルタイマーの調整では，制御信号数と消費電力がトレードオフの関係となるため，なるべく短いアイドルタイマーで消費電力を抑えつつ，Radio Tail 部分でなるべく多くのトラヒックをカバーして制御信号を削減することが望ましい．ここで，単位消費電力あたりの制御信号削減量を「アイドルタイマー効率」と定義すると，それはすなわち，単位アイドルタイマー長あたりの後続

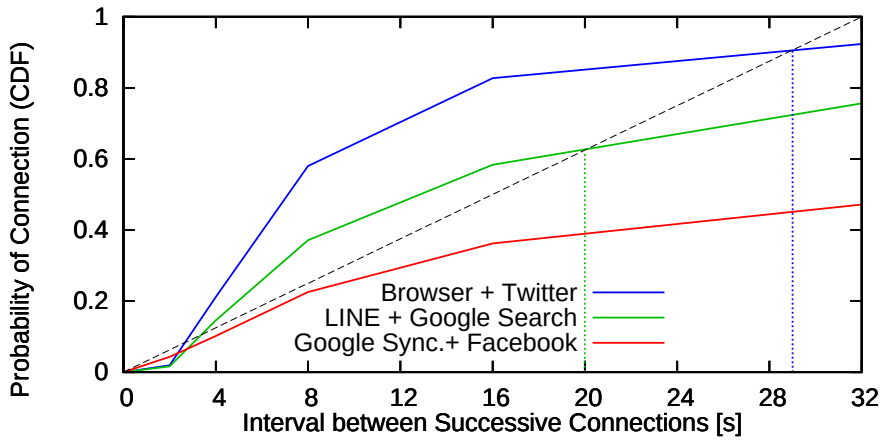

図 2.8: 通信間隔分布 (CDF) からのアイドルタイマーの決定手順

トラヒックカバー率であり，カバー率が 100%となる点において，許容されるアイドルタイマー長の上限が定まる．また，通信間隔の CDF のグラフ（図 2.8）では，時間軸を線形軸とした場合の原点からの傾きにあたる．この傾きが基準値より大きくなる区間のうち，アイドルタイマーが最長となる点を選ぶことで，所望のアイドルタイマー効率を維持しつつ，トラヒックに応じたアイドルタイマー長が得られる．ただし，通信の発生頻度が低い場合には，この条件を満たす点が存在しないため，3G ネットワークの RTT（数百 ms）とコンテンツサーバの応答時間を考慮し，1 回の通信が分断されることのないよう，アイドルタイマー長は 2 秒とする．

図 2.8 は，アイドルタイマー長の上限を 32 秒と設定した場合，すなわち，アイドルタイマー効率の基準値を 100%/32sec=3.125(%/sec) と設定した場合の例である．この基準値では，「ブラウザ」と「Twitter」の合成トラヒック（青線）に最適なアイドルタイマーは 29 秒となり，「LINE」「Google 検索」の合成トラヒック（緑線）では 20 秒となる．ただし，「Google 連絡先の同期」と「Facebook」の合成トラヒック（赤線）では，アイドルタイマー効率が 3.125(%/sec) 以上となる点が存在しないため，アイドルタイマーは 2 秒とする．なお，このアイドルタイマー効率の基準値は，制御信号数と消費電力のどちらを優先したいかに応じて適宜変更できる．

ユーザ操作に基づく制御

ユーザ操作の影響については，図 2.5 に示した通信発生確率の $CDF_{operation}$ を，以下の式を用いて，全アプリの合成トラヒックの CDF に反映する．すなわち，ユーザ操作から τ 秒経過後に発生するトラヒックの割合を乗算することで，ユーザ操作の影響を反映する．

$$CDF(t) = CDF_{Total}(t)\{1 - CDF_{operation}(\tau)\} \tag{2.2}$$

ここで，t は直近の通信からの経過時間，τ は直近のユーザ操作からの経過時間である．t と τ は必ずしも一致しないことに注意が必要である．

画面の点灯状況に基づく制御

2.2.2 節で確認したように，画面の点灯状況によって通信の発生頻度は大きく変化する．これに対応するため，2.3.2 節の制御では，画面点灯中と消灯中に分けて通信間隔の分布を集計し，画面の点

灯状況に応じた分布データを用いて，端末全体のトラヒックの通信間隔の累積分布関数 CDF_{Total} を算出し，最適なアイドルタイマー長を決定する．

以上の手順で算出したアイドルタイマー長は，以下のタイミングで再計算し適用する．

- 端末上のいずれかのアプリが起動または終了した時
- 端末上のいずれかのアプリが通信を行った時
- 端末の画面が点灯または消灯した時
- 端末の電源が投入された時

2.3.3　不要アプリの制御機構

概要

本手法では，Android SDK でサポートされていない，アプリ単位での通信制御を実現するため，Linux カーネルの `iptables` コマンドを活用する．

`iptables` のパケットフィルタには，INPUT ／ FORWARD ／ OUTPUT の 3 つのルールチェインがあり，それぞれ，その端末に到着／通過／送出されるパケットに対する処理を管理している．各ルールでは，送信元 IP アドレスや宛先ポート番号などを指定して処理対象とするパケットを規定し，通過／拒否／破棄などの処理を指定できる．本手法では，OUTPUT チェインのみを使用する．これは，INPUT チェインで受信パケットを遮断したとしても，そのパケットは既に携帯電話ネットワークを通して配送されたものであり，コネクションの確立を阻止できないためである．なお，OUTPUT チェインでは，送信元プロセスの UID を指定して，処理対象とするパケットを規定できる．

一方，Android OS では，アプリは各々に割り当てられた Dalvik 仮想マシン上で実行される仕組みになっている．それぞれの Dalvik 仮想マシンは，アプリに割り当てられた固有の UID で実行されており，ファイル IO やネットワーク通信などカーネルレベルの命令を実行する場合には，その UID と Linux カーネルのユーザ管理機能を用いて，アプリ間の独立性を確保している．それゆえ，どのアプリがどのファイルにアクセスしたか，あるいは，どのアプリがどれだけネットワーク通信を行ったか，を Android OS では簡単に調べることができる．

そこで，本手法では，`iptables` の UID 指定機能を用いて，アプリ単位での通信制御を実現する．なお，`iptables` の利用には root 権限が必要となるため，本手法は OS ベンダまたは端末メーカによる実装を想定している．

プッシュ通知のサポート

本研究では，SMS や GCM（Google Cloud Messaging）によるプッシュ通知がユーザビリティの維持に不可欠であると考え，通信制御中であってもプッシュ通知が利用可能となるよう配慮している．制御プレーンを通して配送される SMS は，我々の通信制御による影響を全く受けないため，ここでは GCM によるプッシュ通知を利用可能とするために必要な要件について論じる．

まず，GCM の仕組みについて簡単に説明する．端末上で GCM のプッシュ通知を利用するアプリが起動されると，Android OS は GCM サーバにアクセスしてユーザトークンを登録し，GCM サーバとの間に TCP コネクションを確立する．その後は，定期的にパケットをやり取りするこ

とでコネクションを維持し，何らかの理由でコネクションが切断された場合には自動的にコネクションを張り直す．具体的には，Wi-Fi 接続中は 15 分に一度，LTE/3G 接続中は 53 分に一度，mtalk.google.com の 5,228〜5,230 番ポートと通信することが知られている．

次に，GCM サーバにメッセージの配信依頼が届くと，宛先のユーザトークンから端末を特定し，その端末との間のコネクションを調べる．もし，コネクションが存在すれば，そのコネクションを通じて端末にメッセージを送信する．もし，コネクションが切断されていた場合には，端末がオフライン状態にあると判断し，次にコネクションが確立された時にメッセージを送信する．

端末側では，サーバからメッセージが届くと，Android OS が GCM サーバに ACK を返し，Broadcast Intent を用いて当該アプリにメッセージが伝えられる．GCM の通信部分は全て Android OS によって管理されており，アプリはユーザトークン登録の API を 1 度コールすれば，後は Broadcast Intent を待つだけで良い仕組みになっている．それゆえ，本提案手法のように，特定のアプリの UID を指定して通信を遮断した場合でも，Android OS のシステムを司る UID を制御対象としない限りは，当該アプリは正常にプッシュ通知を受信することができる．ただし，プッシュ通知をトリガとして，当該アプリがメールの取得やデータの同期などの通信を行う場合には，通信制御の対象となってしまうため，別途配慮が必要である．

一方，root，system，shell など，Android OS のシステムを司る UID を指定して通信を遮断した場合や，UID を指定せずに全通信を遮断した場合には，GCM サーバと端末の間のコネクションを維持できなくなるため，プッシュ通知の配信は不可能となる．この場合，GCM サーバは端末がオフライン状態になったと判断し，次回のコネクション確立までメッセージの送信を待機する．

本提案手法では，GCM への影響を防ぐため，システム領域（/system/app/）のアプリを制御対象から除外し，ユーザ領域（/data/app/）のアプリのみを制御対象とする．

他の実装方式の検討

Android 端末上で通信制御を行う手法は，iptables 以外にも存在する．root 権限を必要とせず，Android SDK の範囲内でアプリ毎の通信制御を実現する手段としては，VpnService API が挙げられる．しかし，この方式では制御用の Android アプリが常駐するため，Linux カーネル上で動作する iptables 方式と比較して端末の CPU 負荷が大きい．一方，root 権限が利用可能であれば，端末上にプロキシサーバを設置する方式や，hosts ファイルを編集する方式が考えられる．しかし，前者は，アプリ毎の制御を実現するためにパケット解析が必要で CPU 負荷が大きく．後者は，ホスト名での制御しかできない．他に，外部サーバを経由させる方式も考えられるが，ユーザ数に比例してサーバの負荷が増大し，サービスとしてスケールしない．

2.3.4　必要アプリの制御機構

アイドルタイマー長の制御方法に関しては，各端末のチップセットや実装に依存するため，汎用的な実装方法に関する言及は避ける．ここでは一例として，ソニーモバイルの Xperia シリーズにおけるアイドルタイマーの設定方法を紹介する．

Xperia シリーズでは，省電力化のためにアイドルタイマー長を短縮する実装がなされており，そのコンポーネントが Linux カーネル上に散見される．Xperia Z1 f (SO-02F) の場合，RRC コネクション管理のためのコマンド（/system/bin/fast-dormancy）と設定ファイル（/system/etc/fast-dormancy/fd_int_conf.txt）が存在する．設定ファイルには waitTime という項目があり，初期値は “5” 秒となっている．この項目を編集することで，アイドルタイマー長を変更できる．なお，Samsung,

LG 電子，HTC，モトローラ，シャープ，富士通製の端末についても調査を行ったが，root 権限を取得できない端末も多く，同様のコンポーネントを発見することができなかった．

なお，上記コマンドの実行や，上記ファイルの編集には root 権限が必要となるため，2.3.3 節で述べた不要アプリに対する制御と同じく，OS ベンダまたは端末メーカによる実装を想定している．

2.4　制御信号と消費電力の削減量の評価

2.4.1　不要アプリに対する性能

評価方法

我々は，前述の「通信量お知らせアプリ」を用いて，通信発生時の各アプリの実行状況を統計的に調査した．具体的には，Android SDK の ActivityManager.RunningAppProcessInfo API を用い，通信発生時に 10 回に 1 回の割合で，各アプリによるユーザへのインタラクションの有無を以下の 6 段階で取得した．前半の 3 段階はユーザにとって認知可能な状態，後半の 3 段階は認知不能な状態である．

- 100：フォアグラウンドで動作中
- 130：BGM 再生などユーザが認知可能な状態
- 200：ウィジェットなどが可視な状態
- 300：サービスとしてバックグラウンドで動作中
- 400：何らかのコードがバックグラウンドで動作中
- 500：休止中

ここでは，直近 1ヶ月間のデータにおいて少なくとも 1 回以上，ユーザにとって認知可能な状態になったことのあるアプリを必要アプリとし，それ以外を不要アプリとして，コネクション数の削減量を評価した．なお，ユーザ数が少ないのは，この機能を追加した時期が遅かったためである．

コネクション数の削減効果

調査対象となった 419 ユーザの全コネクションのうち，13.1%が不要アプリによるものであった．従って，必要アプリのトラヒックとの兼ね合いのため一概には言えないが，不要アプリのトラヒック遮断により最大で 13.1%の制御信号と消費電力を削減できる．また，ユーザ別に見ると，不要アプリのコネクションが全体の 38.5%に達するユーザもおり，削減効果はユーザの利用状況によって大きく異なる．

2.4.2　必要アプリに対する性能

評価方法

我々は，2.2 節で収集したトラヒックの統計データを用いて仮想的なトラヒックを生成し，提案手法を適用した際の制御信号と消費電力の削減量を評価した．ただし，ユーザ操作のタイミングに

ついては,「通信量お知らせアプリ」で収集したデータには含まれないため，2.4.2 節で別途評価を行うこととし，ここではユーザ操作のない状況を想定した．また，制御信号数はコネクション数に依存して決まり通信データ量には依存しないこと，データの送受信に必要な電力は必要経費であることから，ここではデータ量は考慮せず Radio Tail 部分の消費電力に焦点を当てている．また，ディスプレイや CPU の消費電力は考慮していない．

まず，表 2.1 に示したコネクション数での上位 16 アプリから，1〜16 個を無作為に選定し，2.3.2 節の式 (2.1) を用いて，合成トラヒックの通信間隔の累積分布関数を得た．次に，この分布に基いて，10,000 コネクションの仮想的なトラヒック間隔を生成した．この仮想トラヒックに対し，2.2 節で確認した実際の消費電力（Radio Tail 部分：0.517W，アイドル時：0.056W）とアイドルタイマー長（13 秒）を用いて，本手法の動的アイドルタイマー適用時と，従来の固定長アイドルタイマー適用時の制御信号数と消費電力をそれぞれシミュレーション評価した．シミュレーションは，画面点灯中・消灯中の各データに対して，アプリ数を 1〜16 個に変化させながら，各 100 回ずつ行った．なお，図 2.9,2.10 の縦軸は，従来方式を 100%とする相対値であり，エラーバーは標準偏差を表す．

実行中のアプリ数と削減性能

図 2.9 は，許容するアイドルタイマー長の上限を 32 秒と設定した場合における，画面点灯時の制御信号および消費電力の削減量である．各シミュレーションで選択されたアプリの組み合わせによって，性能に大きなバラつきがあるものの，制御信号を大幅に削減できる．実行中のアプリ数が多くトラヒックの発生頻度が高いほど削減性能は向上し，消費電力への影響もアプリ数 3 をピークに減少する．アプリが 16 個の場合には，制御信号を平均 91.3%削減しつつ，消費電力を平均 0.4%削減できる．

一方，画面消灯時の制御信号および消費電力の削減量は図 2.10 の通りである．画面点灯中と比較すると削減幅は小さいものの，消費電力を削減できる．実行中のアプリ数が多いほど性能が向上し，アプリが 16 個の場合には，消費電力を平均 46.9%削減しつつ，制御信号の増加を平均 12.8%に抑えられる．

2.2.2 節で指摘したように，スマートフォンは稼働時間の約 20%の時間しか画面が点灯した状態にないが，トラヒックの 95%は画面点灯中に発生する．このことを踏まえて，点灯中と消灯中を合わせた全体での性能を評価する．消費電力は時間に依存するため，全体では $-0.4\times0.20-46.9\times0.80 = -37.6\%$ の消費電力を削減できることになる．一方，制御信号数はトラヒック量に依存するため，全体では $-91.3 \times 0.95 + 12.8 \times 0.05 = -86.1\%$ の制御信号を削減できることになる．

アイドルタイマー効率の基準値と削減性能

図 2.11 は，実行中のアプリ数が 10 個の場合において，アイドルタイマー効率の基準値，すなわち，許容されるアイドルタイマー長を変化させた時の制御信号および消費電力の削減量である．当然ながら，許容するアイドルタイマー長を長くするほど，制御信号の削減幅は大きくなるが，消費電力の削減幅は次第に小さくなっていく．反対に，許容するアイドルタイマー長を短くすると，制御信号が急激に増加する．制御信号の削減幅は 32 秒付近から伸びが鈍くなっているため，アイドルタイマー効率の基準値を 100%/32sec=3.125(%/sec) とするのが良い．ただし，この基準値は，制御信号数と消費電力のどちらを優先したいかに応じて適宜変更できる．

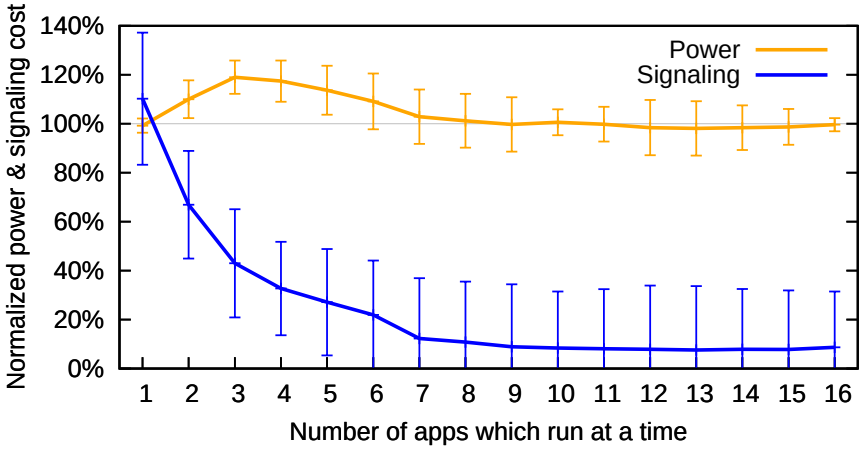

図 2.9: 従来方式を 100%とした，画面点灯時の提案手法の性能評価

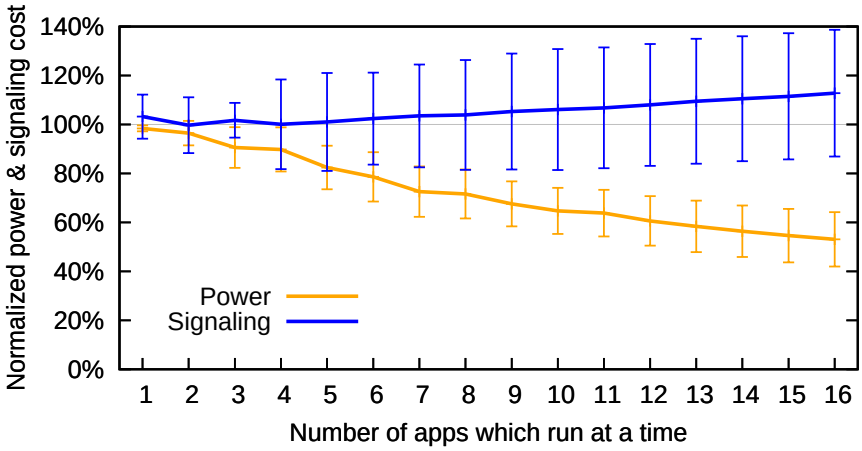

図 2.10: 従来方式を 100%とした，画面消灯時の提案手法の性能評価

ユーザ操作の考慮と削減性能

我々は，2.2.2 節で収集した，ユーザ操作およびトラヒックの発生間隔に関する 3 ユーザ分の統計データを用いてシミュレーション評価を行った．なお，評価を独立させたのは，「通信量お知らせアプリ」でユーザ操作に関する情報を収集できなかった関係で，シミュレーションに用いる統計データが異なるためである．

まず我々は，統計データに基づいて 86,400 秒分の仮想的なタッチ操作およびトラヒック発生の時系列を生成した．なお，この統計データでは，アプリ毎の通信は区別されておらず，端末全体のトラヒックを一括りにしている．次に，この仮想トラヒックに対し，2.2 節で確認した実際の消費電力（Radio Tail 部分：0.517W，アイドル時：0.056W）とアイドルタイマー長（13 秒）を用いて，制御信号と消費電力の削減量を評価した．シミュレーションは，アイドルタイマー効率の基準値を変化させながら，各 10 回ずつ行い，グラフには平均と標準偏差を示す．2.4.2 節と同様，通信データ量は考慮せず，Radio Tail 部分の消費電力に焦点を当てている．

図 2.12 は，アイドルタイマー効率の基準値を変化させた時の制御信号および消費電力の削減量である．設定された基準値に応じて，消費電力または制御信号数を削減できる．2.4.2 節で述べたよ

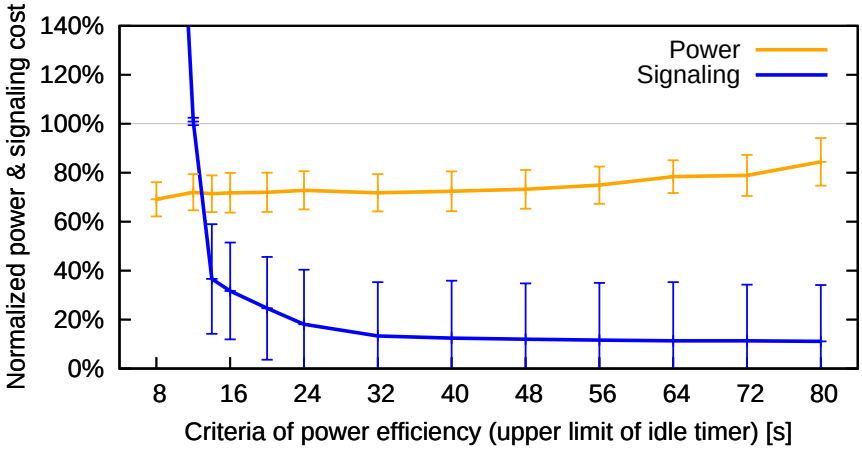

図 2.11: アイドルタイマー効率の基準値を変化させた時の性能評価

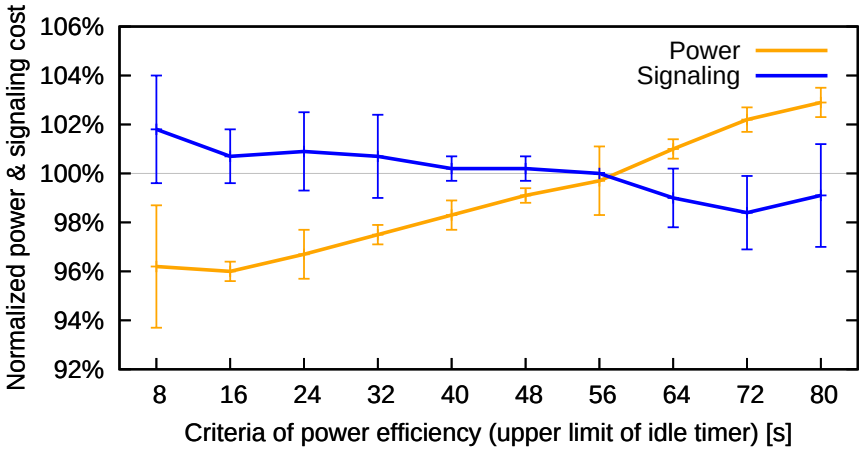

図 2.12: ユーザ操作のみを考慮した場合の提案手法の性能評価

うに許容するアイドルタイマー長を 32 秒とした場合には，平均 0.7%の制御信号増で平均 2.5%の消費電力を削減可能である．2.4.2 節までの評価結果とは，シミュレーションの元となった統計データが異なるため単純な掛け算はできないが，ユーザ操作を考慮することで，消費電力と制御信号をより良く削減できると考えられる．

2.5　関連研究

スマートフォンの通信制御と省電力化に関しては，様々な研究調査が盛んに行われてきた [13, 14, 15, 16]．しかし，我々の知る限りでは，1 つ 1 つのアプリについて個別にトラヒックの性質を分析し，通信間隔の分布に着目して制御に反映させることを検討する研究はなかった．Chakraborty らは，スマートフォンのバックグラウンド通信が，同じ基地局に接続している他の端末の通信と重ならないよう，各端末が分散的にスケジューリングを行うことで，端末の消費電力を削減しつつ平均スループットを向上させる画期的な手法を提案している [17]．Athivarapu らは，スマートフォン上のアプリが発行するシステムコールに着目し，その発行パターンから通信タイミングを予測し，

ごくわずかの制御信号増と引き換えに，ユーザ端末の消費電力を 20〜40%削減する手法を提案した [18]．この手法は消費電力の削減に重きを置いており，制御信号数の削減については詳しく考慮されていない．Huang らは，スマートフォンの画面消灯中に行われる通信の多くが，画面点灯中の通信と比べて，高い遅延耐性を持つことを発見し，スリープ中の通信をスケジューリングすることで，消費電力を約 60%削減する手法を提案した [19]．この研究では，20 ユーザ・5ヶ月分の通信記録しか解析しておらず，トラヒックの内容に偏りがあった可能性がある．Qian らは，スマートフォン上のアプリが，自身の行う通信の開始タイミングや終了タイミングを正確に予測できることに着目し，アプリが OS に対して通信タイミングを通知する仕組みを構築することで，Radio Tail の長さを最適化する手法を提案した [20]．しかしながら，この手法ではアプリ側での対応が必要となり，100 万本を超える既存のアプリ資産をそのまま活用することができないという欠点がある．特定の種類の通信に特化した手法としては，Han らによるテザリング通信に特化した省電力化手法 [21] や，Xu らによるメール同期タイミングの最適化手法 [22]，Nath らによるモバイルアプリ向け広告の通信最適化手法 [23] などが挙げられる．これらの手法は，いずれも我々の提案手法と組み合わせることが可能であり，さらなる性能向上を期待できる．

2.6　まとめ

本章では，スマートフォン上の様々なアプリに対して，各アプリの性質を反映した通信制御を個別に実施することで，ユーザビリティを維持しつつ，携帯電話網を流れるデータトラヒックと制御信号，端末の消費電力を効果的に削減する手法を提案した．

本手法では，アプリ起動やウィジェット利用の有無などの情報に基いて不要アプリを判別し，それらの通信を選択的に遮断することで，効果的にトラヒックを削減する．また，他のアプリについては，各アプリのトラヒックの統計的な性質に基きアイドルタイマーを動的に変更することで，制御信号量と消費電力を削減する．また，画面の点灯状態やユーザ操作からの経過時間も考慮する．

本手法では，アプリ単位での通信制御が鍵となるが，Android SDK にはそれを実現する API が存在しないため，我々は，Android が Linux カーネル上で動作するミドルウェアであることに着目し，iptables コマンドを用いて制御機構部分を実装した．

評価実験では，実際に稼働中の Android 端末から収集したトラヒックの統計データを用いて仮想トラヒックを生成し，本提案手法を適用した場合のトラヒックをシミュレーションして，制御信号と消費電力の削減量を評価した．結果として，制御信号を最大 86.1%，消費電力を最大 37.6%，同時に削減できることを確認した．

第3章

LTE網のパディングを活用したセンサデータ送信手法

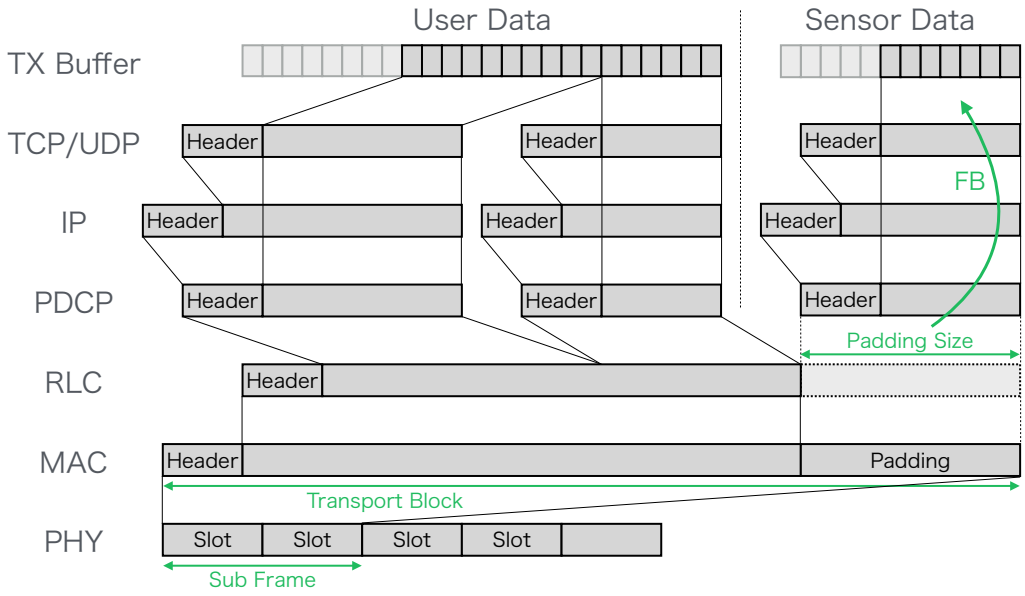

図 3.1: LTE における送信パケットの処理フロー（右上部分が提案手法）

3.1　はじめに

スマートフォンを用いた参加型センシングでは，センシング時の消費電力の増加，データ送信時の消費電力および通信量の増加が，ユーザに参加を躊躇わせる要因となり，ユーザ数を確保することが難しいという課題があった．本稿では，LTE 網の無線区間に最小割り当て単位が存在することに着目し，MAC レイヤのパディング部分にセンサデータを埋め込んで送信することで，ユーザ端末の消費電力や通信量を増加させることなく，センサデータを送信する手法を提案する．これまでにも，センサデータをユーザトラヒックの末尾に付加して送信することで消費電力の削減を狙う Piggybacking 手法は提案されているが [15]，LTE 網において MAC レイヤのパディングを活用する手法は，筆者の知る限りでは存在しない．

3.2　LTE ネットワークの仕様

LTE は 2009 年 3 月に 3GPP によって標準化された第 3.9 世代移動通信システムであり，第 3 世代の W-CDMA 方式や CDMA2000 方式と比較して，高速大容量かつ低遅延という特徴を持つ [24, 25] ．LTE では，無線区間の多重アクセス方式として，下りに OFDMA（Orthogonal Frequency Division Multiple Access）方式，上りに SC-FDMA（Single Carrier FDMA）方式を採用することで，高速大容量な下りリンクと省電力な上りリンクを実現している．全二重化モードに周波数分割多重を採用する FDD-LTE 方式と，時分割多重を採用する TD-LTE 方式が存在するが，他の仕様に大きな違いはない．なお，WiMAX は Release 2.1 から TD-LTE との互換性を持つようになったため，本章の内容は LTE 網だけでなく WiMAX 網にも適用可能である．

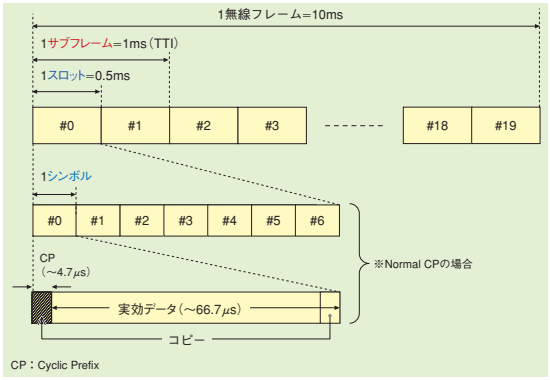

図 3.2: LTE の無線フレーム構成 [24]

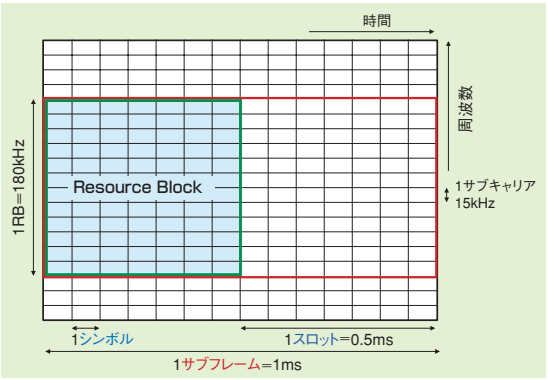

図 3.3: サブフレーム構成と RB の配置 [24]

3.2.1　データリンク層（レイヤ 2）

LTE のデータリンク層は，PDCP サブレイヤ，RLC サブレイヤ，MAC サブレイヤの 3 つのサブレイヤから構成され，ネットワーク層（レイヤ 3）から受け取った IP パケットを加工して，物理層（レイヤ 1）のフレームサイズに収まるよう調整する役割を担う（図 3.1）．以下，各サブレイヤの役割を解説する．

PDCP サブレイヤ

PDCP（Packet Data Convergence Protocol）サブレイヤでは，冗長になりがちな IP ヘッダ部分の圧縮を行い，無線区間のリソース利用効率の向上を図る [26]．具体的には，最低でも 20 バイトある IP ヘッダを除去して，1〜4 バイトのトークンを付加する．これにより，1500 バイト長の IP パケットであれば，通信量の 1%強を節約できる．

RLC サブレイヤ

RLC（Radio Link Conrtol）サブレイヤでは，通信路の状況によって時々刻々と変化する物理層のフレームサイズ（TB サイズ）に合わせて，PDCP サブレイヤから受け取った IP パケットを分割・結合し，1 バイトまたは 2 バイトのヘッダを付加する [27]．

MAC サブレイヤ

MAC（Medium Access Control）サブレイヤでは，RLC サブレイヤから受け取ったデータチャンクに 1 バイトまたは 2 バイトのヘッダを付加した上で，データ長が TB サイズに満たない場合にパディング処理を行い，データ長を調整する [28]．パディング領域は通常 0 埋めされるが，仕様上は内容に関する制約がないため，この部分に直接データを埋め込むことが可能である．ただし，独自形式でデータを埋め込んだ場合には，基地局側にデータを回収する機構が必要となる．

表 3.1: 変調方式と TB サイズの関係（上りリンク）

MCS Value	Modulation	Code Rate	TB Size[bit]
0	QPSK	0.106	32
1	QPSK	0.152	56
2	QPSK	0.182	72
3	QPSK	0.242	104
4	QPSK	0.273	120
5	QPSK	0.318	144
6	QPSK	0.379	176
7	QPSK	0.470	224
8	QPSK	0.530	256
9	QPSK	0.606	296
10	16QAM	0.333	328
11	16QAM	0.333	328
12	16QAM	0.379	376
13	16QAM	0.439	440
14	16QAM	0.485	488
15	16QAM	0.545	552
16	16QAM	0.591	600
17	16QAM	0.621	632
18	16QAM	0.682	696
19	16QAM	0.758	776
20	16QAM	0.818	840
21	16QAM	0.818	840
22	16QAM	0.879	904
23	16QAM	0.970	1000
24	16QAM	1.030	1064
25	16QAM	1.091	1128
26	16QAM	1.152	1192
27	16QAM	1.212	1256
28	16QAM	1.424	1480

3.2.2　物理層（レイヤ 1）

LTE の物理層では，周波数軸方向に 12 個のサブキャリア（180kHz），時間軸方向に 7 シンボル（0.5ms，スロット）を最小単位（Resource Block：RB）とし，時間軸方向に 2RB 単位（1ms，サブフレーム）でリソース割り当てを行う（図 3.2，図 3.3）．この最小割り当て単位で送信できるデータ量（TB サイズ）は，変調方式や符号化率に依存し，変調方式や符号化率は通信路の状況に応じて決定される．変調方式は，QPSK，16QAM，64QAM の 3 種類（上りの 64QAM はオプション）が用意され，符号化率と合わせて 29 通りのパラメータ（Modulation and Coding Scheme：MCS）が定義されている．MCS に応じて，TB サイズは最小で 32 ビットから最大で 1480 ビットまで変化する（表 3.1）．

表 3.2: RSSNR と変調方式 (MCS) の対応関係

RSSNR[dB]	MCS Value
＜ 0	0
＜ 3	2
＜ 5	4
＜ 7	6
＜ 9	8
＜ 10	10
＜ 12	12
＜ 14	14
＜ 15	16
＜ 17	18
＜ 19	20
＜ 21	22
＜ 23	24
＜ 25	26
≧ 25	28

3.3　利用可能な MAC 層パディングの量

3.3.1　実態の調査

ユーザの利用状況や変調方式に依存するパディング量の実態を調査するため，我々は「通信量お知らせアプリ with パケットフィルタ」を製作し，Google PLAY で一般公開して協力者を募ってきた [29]．このアプリでは，パディング量の推定に必要なパラメータとして，(1)IP パケット長の分布，(2)IP パケットの生起間隔の分布，(3) 無線区間の変調方式，を収集する．IP パケット長と生起間隔については，Android SDK の VpnService API を活用して，端末上の全ての IP パケットを傍受し記録した．また，無線区間の変調方式については，PhoneStateListener API を用いて，信号強度，RSRP，RSRQ，RSSNR，CQI の 5 つのパラメータを取得し，ns-3 の LTE モジュールで用いられる RSSNR と変調方式の対応表に基づいて，変調方式を推定した（表 3.2）[30]．なお，LTE パラメータの取得可否には機種依存があり，例えば RSSNR は原則 Android 4.2 以上の端末で取得可能だが，Nexus5（Android 5.0）では取得できなかった．本稿の内容は，2014 年 12 月 14 日から 2015 年 1 月 31 日までに回収した 108 ユーザ，81.09 GB 分の通信データに基づいている．

IP パケット長の分布を図 3.4 に示す．全パケットのうち 34.5%は 40 バイト長であるが，これは Keep-Alive のための空の TCP パケットであると推測される．また，576 バイト長，1500 バイト長のパケットがそれぞれ 3.2%，50.7%を占めているが，これは IPv4 で規定される最小の MTU サイズおよびイーサネット標準の MTU サイズに対応しており，分割して送信されたデータパケットであると推測される．なお，パケット長の平均値は 837 バイト，中央値は 1500 バイトであった．一方，TB サイズの分布は図 3.5 に示す通りで，全ユーザの平均値は 550 ビット，中央値は 488 ビットであった．しかし，機種やユーザによる差異が大きく，平均 148 ビットのユーザから平均 1477 ビットのユーザまで幅広く分布する．これは，ユーザが主に滞在していた場所の電波状況の良し悪しと，その機種の電波受信感度の良し悪しを反映しているものと思われる．

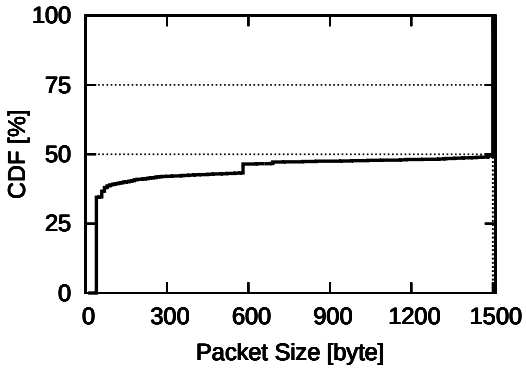

図 3.4: 送出パケットサイズの分布

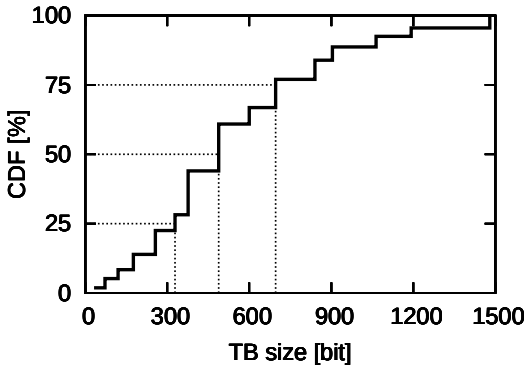

図 3.5: 無線区間 TB サイズの分布

3.3.2　パディング量の推定

今回，我々は，IP パケット長と生起間隔，TB サイズの分布データを基に，シミュレーションによるパディング量の推定を行った．まず，IP パケット長と生起間隔の分布データに従って，1 万パケット分の仮想トラヒックを生成した．次に，TB サイズの分布に従って，各パケット送信時の TB サイズを決定し，図 3.1 の手順に従ってフレームに格納した際のパディング量を求めた．なお，10 ミリ秒以内の間隔で連続して生起するパケットは一括でスケジューリングするものとし，レイヤ 2 で付加される PDCP ヘッダ，RLC ヘッダ，MAC ヘッダは，それぞれ 2，2，3 バイトとした．シミュレーションを 100 回行ったところ，平均のパディング長は 39 バイトであり，全通信量の 1.19%がパディングとなっていることが明らかとなった．これは，毎月 7GB のデータ通信を行うユーザの場合，毎月 83.3MB に相当する．

ユーザによる差異

ユーザ別の統計データを用いて，3.3.2 節と同様のシミュレーション評価を行ったところ，全通信量に対するパディング量の割合は，ユーザによって 0.27%〜51.85%と大きく異なった．ユーザによる差異が生じた原因としては，スマートフォンの利用頻度の違いと電波状況の違いが考えられる．スマートフォンの利用頻度が低いユーザは，Keep-Alive のための空の TCP パケットが相対的に多くなるため，通信量に対するパディング量の割合が大きくなる．また，電波状況の良いエリアに居るユーザは，TB サイズが大きくなるため，パディングが生じやすくなる．

3.4　データの埋め込み方式

限られたパディング領域を最大限に活用するためには，センサデータをそのままパディング領域に埋め込むことが好ましいが，この場合には基地局設備にセンサデータを回収する機構を新設する必要がある．本稿では，携帯電話事業者の保有設備に変更を加えることなく，ユーザ端末側の変更だけで実装可能となるよう，データを UDP/IP パケットに格納し，ユーザデータと同様のレイヤ 2 処理を施して付加する方式を提案する．具体的には，ユーザデータ用とは別にセンサデータ用の送信バッファを設置し，PDCP ヘッダ部分を含めてパディング量と一致するよう，2 バイトを差し引いた分量のセンサデータを付加する（図 3.1 右）．なお，TCP ではなく UDP を採用したのは，

ヘッダとコネクション管理による通信帯域のロスを最小限に抑え，限られたパディング領域をなるべく有効に活用するためである．

上記のシミュレーションにおいて，3 バイト以上のパディングに対して，PDCP ヘッダ分を差し引いてデータ付加可能量を計算したところ，全通信量の 1.13% となった．従って，毎月 7GB のデータ通信を行うユーザの場合，毎月 79.1MB のセンサデータを付加できる．

3.5　応用例

これは，1 日あたり 2.64MB に相当し，時刻・緯度・経度・三軸加速度の情報を毎秒 1 回のペースで収集することに相当する．センサデータを ZIP 形式などで可逆圧縮すれば，実質的なデータ付加可能量を 2〜3 倍にすることが可能であり，また，用途によっては圧縮センシングの技術を適用することで，さらに多くのセンサデータを送信できる．

そこで，自転車やバイクの乗車中にスマートフォンをマウントすれば，舗装の損傷や積雪といった路面状況のセンシングや，交通渋滞の検知などに応用できる．また，地震などの災害発生時には，ユーザの移動経路の情報を収集することで，通行可能な道路を割り出し，救援物資の迅速な配送を支援できる．他にも，スマートフォンの TV チューナ機能で UHF 帯の利用状況を調査し，位置情報と共に収集することで，広範囲のホワイトスペース地図を低コストで作成できる．さらに，センシング間隔を数分まで延長すれば，画像や音声データを送信することも可能であり，自動車のダッシュボードに設置したスマートフォンで，道路の交通量を計測することも考えられる．

3.6　関連研究

スマートフォンを用いた参加型センシングについては，様々な研究が行われてきた [15, 31]．Ra らは，人手による補助が必要なクラウドセンシングをサポートするプラットフォームを設計・実装し，複数のユーザのデータを織り交ぜることでプライバシーに問題に対処した [32]．Das らは，センシング用のバイナリをプッシュ通知を用いて迅速に配布し，大規模なセンシングを補助するプラットフォーム PRISM を製作した [33]．PRISM はセキュリティ面に配慮し，サンドボックス機構を持つ．これらのプラットフォームは，我々の手法と併用が可能である．

センシング時の消費電力を低減する手法としては，Lane らが Piggyback CrowdSensing (PCS) を提案している [34]．PCS では，ユーザがスマートフォンを操作するタイミングを学習して，操作中にのみセンシングを行うことで，スリープ状態からの復帰回数を減らし，また，ユーザが行う通信の合間にセンサデータを送信することで，Radio Tail 部分の消費電力を削減する．ただし，この方式では MAC 層パディングの活用までは考慮されていない．

また，スマートフォンの通信をレイヤ別に分析し，リソースの非効率な利用を発見するツール ARO を Qian らが提案している [35]．ただし，彼らが主に検討対象としているのは RLC サブレイヤおよび RRC ステートマシン部分の制御であり，MAC サブレイヤのパディング領域については検討がなされていない．

第4章

スマートフォン・センシングの活用例：電気自動車の接近検知

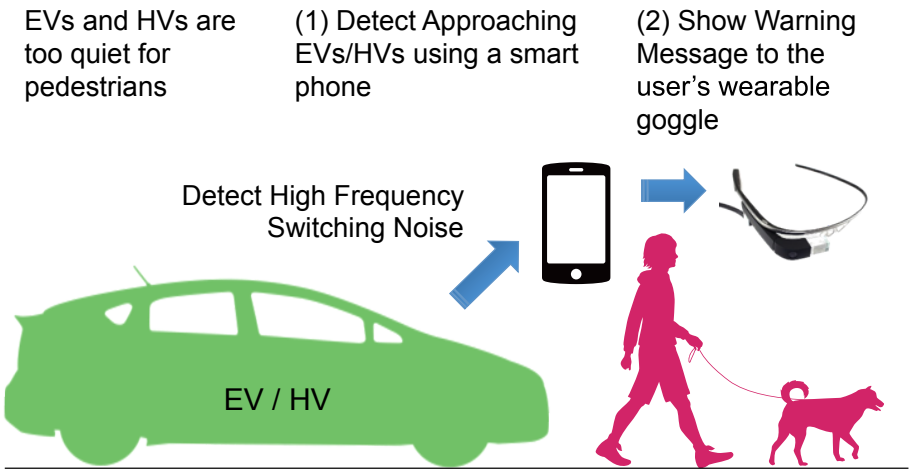

図 4.1: 提案するシステムのコンセプト

4.1　はじめに

ハイブリッド車 (HV) や電気自動車 (EV) は近年，急速な性能向上と販売価格の低下により，消費者にとって手ごろな選択肢となりつつある．特に，HV は，日本における新車販売台数 (2012 年) の 29.7%を占めており，本格的な普及が始まっている [36]．HV や EV の販売価格は，まだガソリン車と比較して高い傾向にあるが，高い燃費性能と走行時の静音性がユーザの支持を集めている．特に，30km/h 未満での低速走行時における静音性は著しく優れており，このことが新たな社会問題を引き起こした [37, 38]（図 4.2）．従来のガソリン車が歩行者の死角から接近する場合，歩行者はそのエンジン音によって車両の接近を知覚できたが，低速走行時の HV や EV はエンジン音がなく走行音が小さいため接近に気づきにくい．この静音性を悪用し，HV で歩行者の後方から近づき，カバンを引ったくる事件も発生している [39]．また，アメリカ運輸省・道路交通安全局 (NHTSA) の統計によると，HV の駐車時や後退時における対歩行者事故の発生率は，ガソリン車の 2 倍に達しており，対自転車事故の増加も報告されている [40]．今後も，HV や EV の普及が進むと見られることから，事故対策は急務である．

このような問題に対処するため，国内で販売されている HV や EV には「車両接近通報装置」が搭載されている，この装置は，車外に備え付けられたスピーカーから電子音を流すことで，周囲の歩行者に車両の接近を知らせるものである．しかし，この装置は HV や EV の長所である静音性を損なうものであり，また，運転者の操作でスイッチを切ることができるため，安全性の観点からは問題がある．他には，車々間通信システムや歩車間通信システムが，多くの研究者から提案されている [41, 42]．これらのシステムは，狭域通信（DSRC）や携帯電話網を用いて，車両と歩行者が互いの位置情報を交換し，互いの接近を事前に通知する，というものである．車両側と歩行者側の双方に専用設備が搭載されていれば，このようなシステムは非常に効果的であるが，現状では普及に至っていない．車両側のインフラ整備を待たずとも，ユーザの持ち歩くデバイスだけで車両の接近をできるなら，その方が好ましい．

本章では，EV や HV の発する高周波音を利用して，スマートフォンで車両の接近を検知する手法を提案する．ほぼ全ての EV や HV は，搭載されているモータユニットの内部機構ゆえに，5kHz，10kHz，20kHz 周辺にスイッチング雑音を発することが知られている．このスイッチング雑音には，車種や車速といった有用な情報が含まれている．例えば，モータユニットのスイッチング周波数が

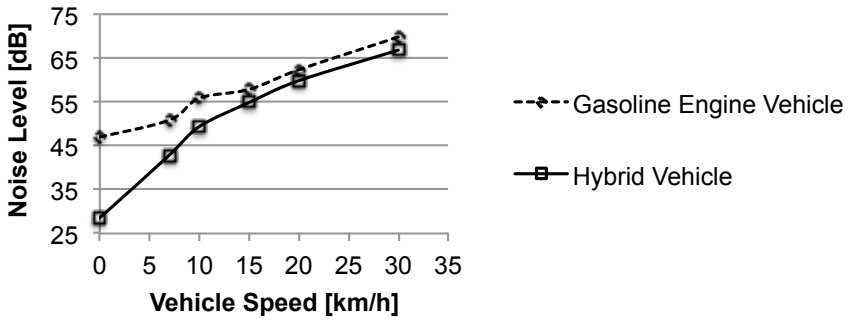

図 4.2: ハイブリッド車とガソリン車の走行音量の比較

既知であれば，ドップラー効果を考慮することで，歩行者と車両の相対速度を割り出す事ができる．また，車種によって，各周波数帯域の音量比が異なることから，車種も判別できる．

我々の実験では，大通りや住宅街の環境雑音下においても，これらのスイッチング雑音が十分に検出可能であることが判明した．さらに，人間の聴覚が苦手とするこれらの高周波音を，スマートフォンなどの機器で容易に検出できることを見出した．また，その検出ロジックはスマートフォンでリアルタイム処理できるほど軽量である．

本章は以下のような構成となっている．まず，4.2 節で EV と HV のスイッチング雑音の録音データを紹介し，データの分析によってどのような情報が利用可能であるかを論じる．次に，4.3 節で提案する EV と HV の検知システムの詳細を述べ，4.4 節と 4.5 節で性能評価を示す．さらに，4.6 節で Android 端末および Google Glass への実装方法を紹介した後，4.7 節で関連研究を紹介して本章をまとめる．

4.2　モータユニットのスイッチング雑音

4.2.1　スイッチング雑音の発生原理

EV や HV を駆動するモータは，タイヤの中に収まるほど小型・軽量で，かつ，エネルギー効率に優れている必要があるため，永久磁石同期電動機（IPM モータ）が採用されることが多い．IPM モータの回転速度は，電源の交流周波数によって制御され，大くの場合，パルス幅変調（PWM）を用いて交流周波数を調整している．PWM とは，短いパルス波のデューティー比を変化させることで，擬似的に任意の周波数の正弦波を出力する機構である．PWM に必要なパルス波を高い変換効率で得るため，HV や EV 絶縁ゲートバイポーラトランジスタ（IGBT）が用いられる．IGBT の特性により，そのスイッチング周波数は 5kHz か 10kHz であることが多い．この高周波電流がモータ内の導線を流れ，モータの筐体を振動させることで，スイッチング雑音が生じる．矩形波には必ず高調波が含まれるため，このスイッチング雑音には 15kHz や 20kHz の倍音も含まれる．

EV や HV から生じるスイッチング雑音を分析するため，我々は HV と EV それぞれ 1 台ずつを用意し，図 4.3 の 3ヶ所で走行音の録音実験を行った．実験車両は販売台数を考慮し，EV として日産リーフ，HV としてトヨタプリウス PHV を採用した．プリウス PHV には，エンジンを全く使わずに走行する「EV モード」が搭載されており，本実験は「EV モード」の状態で実施した．実験場所は，対歩行者事故の多い，駐車場，住宅街，大通りの 3ヶ所である．車速は 5km/h，10km/h，20km/h，30km/h の 4 通りとした．30km/h を上限としたのは，車速が増すほどタイヤの転がり

図 4.3: 計測実験を行った環境

	10km/h	20km/h	30km/h
f_{High}	2.010×10^4Hz	2.018×10^4Hz	2.023×10^4Hz
f_{Low}	1.993×10^4Hz	1.985×10^4Hz	1.978×10^4Hz
$f_{High}-f_{Low}$	170Hz	330Hz	450Hz

表 4.1: EV の 20kHz 周辺の f_0

音やボディの風切音が大きくなり，30km/h 以上ではエンジン車と同等の走行音が発生するためである [43].

実験では，測定者から 1.5m 離れた地点を車両で通過した際の走行音を，腰の前の位置に保持した iPod Touch（第 4 世代）の内蔵マイクで録音した．サンプリング周波数は 48kHz，録音形式は 16bit のリニア PCM である．計測回数は，各実験場所・各車速において最低 5 回以上とした．ただし，大通りでの計測は，交通の妨げとならないよう，30km/h で 3 回だけ実施した．なお，実験中は車両の「車両接近通報装置」を無効とした．なお，騒音計（Rion NA-28）を用いて計測した各実験場所の環境雑音レベルは，駐車場 37.6dB，住宅街 40.6dB，大通り，58.7dB であった．

4.2.2　スイッチング雑音の分析

録音データから最接近の前後数秒間を抜粋したスペクトラムを図 4.5 ，図 4.6 に示す．日産リーフの走行音には，10kHz 周辺と 20kHz 周辺にスイッチング雑音の明瞭なピークが見られる．一方，トヨタプリウス PHV の走行音では，10kHz 周辺や 20kHz 周辺より，5kHz 周辺に強いピークが見られる．人間の可聴域は概ね 20Hz〜20kHz であるが，個人差が大きく，加齢により高周波音に対する感度が低下することが知られている．それゆえ，EV や HV の走行音は高齢者には聞き取りづらい．また，図 4.6 には，各 2,3 個の S 字カーブが見られる．これはドップラー効果によって，見かけの音の高さが変化したためであり，元の周波数 f_0 と観測される周波数 f の関係は以下の式で与えられる．

$$f = \frac{c^2 f_0}{c^2 - v^2}\left\{1 - \frac{v^2 t}{\sqrt{c^2 v^2 t^2 + l^2(c^2 - v^2)}}\right\} \tag{4.1}$$

ここで，c は音速，v は車両の移動速度，l は最接近時の車両と観測者の距離である．t は時刻であり，車両と観測者の最接近時に $t = 0$ となる．この関係式と gnuplot のフィッティング機能を用いて f_0 を計算した（表 4.1）．f_{High} と f_{Low} は，それぞれ図 4.5(20kHz) の高音側・低音側の 2 本

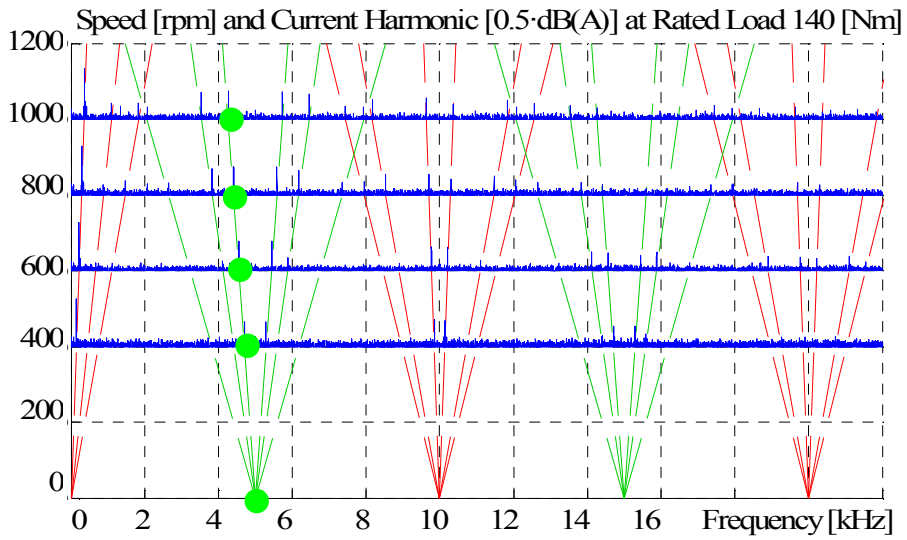

図 4.4: 走行音の大きさとモータの回転速度の関係

の S 字カーブの f_0 を表している．f_{High} と f_{Low} の差は，車速の増加に伴って広がっていく．これは，電源入力のパスル幅変調によって回転速度を調整する，PWM モータに特有な特徴である．図 4.4 に示すように，PWM モータのスイッチング雑音は多数の側波帯に分かれることが知られている [44]．それゆえ，図 4.5 にはそれぞれ 2,3 個の S 字カーブが見られるのである．これらの側波帯の周波数差の情報を活用することで，車両の速度を推定できる．

4.3　検知手法

スマートフォン上で動作する，EV や HV の接近検知ロジックは，単純かつ軽量である必要がある．さらに，環境雑音に対してロバストでなくてはならず，特定のメーカーや車種ばかりに特化してもいけない．また，スイッチング雑音に含まれている，車両の対地速度や観測者との相対速度といった情報も有効活用すべきである．これらの要件を満たすべく，我々は機械学習によるアプローチを選択した．我々の検知スキームのフローチャートを図 4.7 に示す．

4.3.1　特徴量ベクトル

初めに，4.2 節で取得した録音データから，車種・車速毎の特徴量ベクトルを抽出する．ここで，32,768 サンプル（0.683 秒分）の波形データを「1 フレーム」と定義し，録音データをフレーム単位に切り刻んで以下の処理を施す．まず，各フレームの波形データを，高速フーリエ変換 (FFT) によって周波数領域のスペクトルデータに変換する．次に，4.2 節で紹介した，モータユニットのスイッチング雑音の特徴を考慮し，5kHz，10kHz，20kHz 周辺に重点を置いて，表 4.2 に示した周波数帯域のデータを抜粋する．さらに，各周波数帯域について最大音量と平均音量を計算し，全周波数帯域での平均音量で正規化したものを特徴量とする．この正規化処理は，ほとんどのスマートフォンに搭載されている，録音音量の自動調整機構に対処するために必要なものである．この最大音量と平均音量の組み合わせにより，その周波数帯域に鋭いスペクトルが存在するかどうかを判定する．これら 2 つの特徴量を，全 48 帯域について取得し，96 次元の特徴量ベクトルを得る．

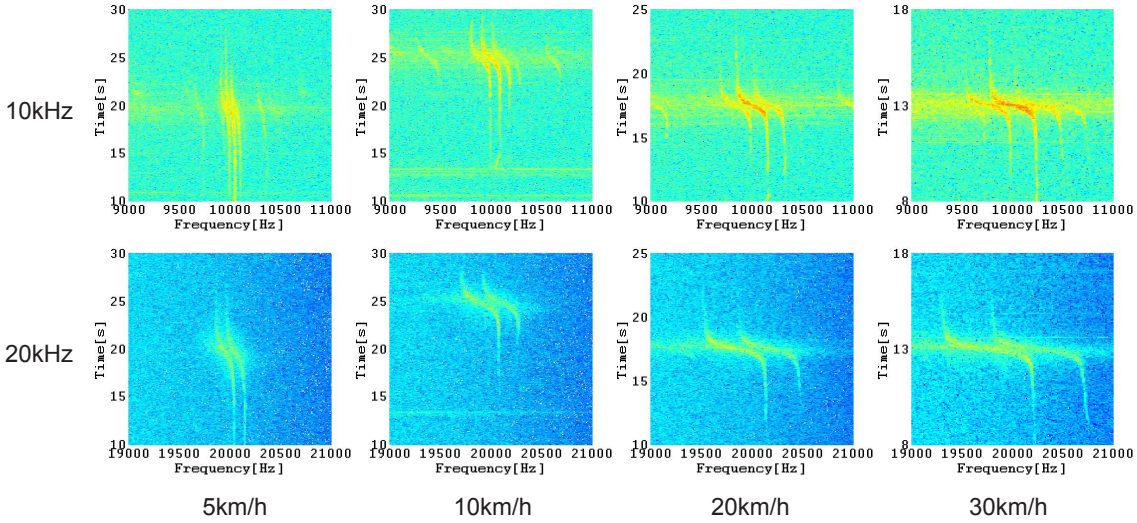

図 4.5: EV（日産リーフ）のスペクトログラム

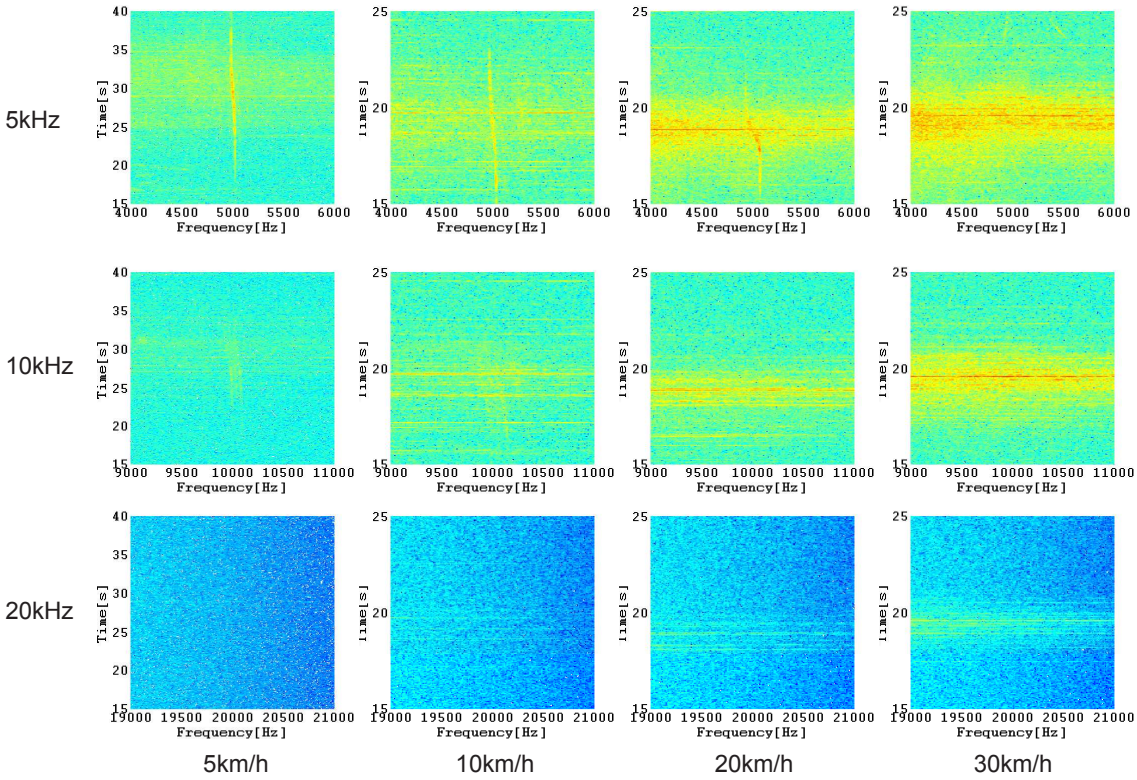

図 4.6: HV（トヨタ プリウス PHV）のスペクトログラム

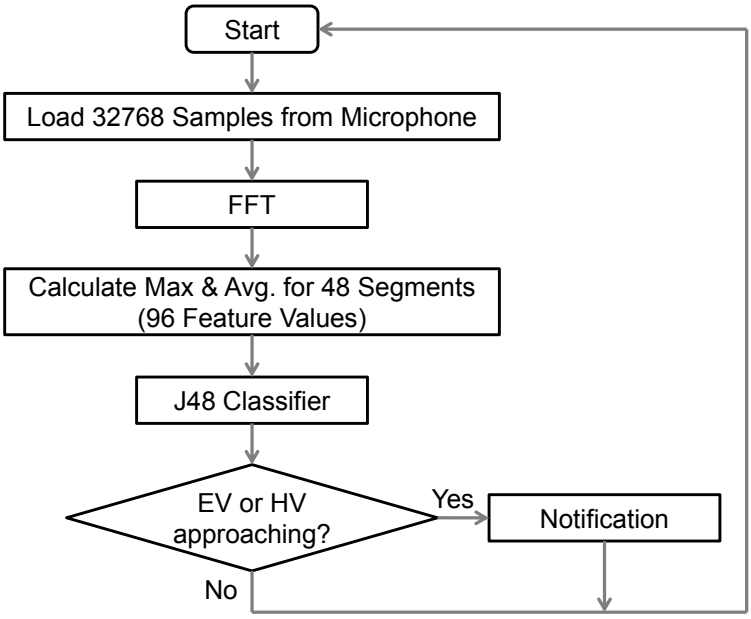

図 4.7: 検知アルゴリズムのフローチャート

Band	Beginning and end	Interval
5kHz	4600Hz – 5400Hz	50Hz (16 segments)
10kHz	9200Hz – 10800Hz	100Hz (16 segments)
20kHz	18400Hz – 21600Hz	200Hz (16 segments)

表 4.2: 周波数セグメント

4.3.2　ラベリング

録音データから抽出した特徴量ベクトルには，1 つ 1 つ手作業で正解ラベルを付与した．このラベルには，接近車両の有無だけでなく，車種や車速の情報も含まれる．ただし，歩行者から離れて行く車両は，歩行者に害を及ぼさないため，車両が測定者から遠ざかる部分の録音データには「車両なし」のラベルを付与した．まとめると，車両接近中の録音データには，車種と車速の情報を含めた「ev5」「ev10」「ev20」「ev30」「hv5」「hv10」「hv20」「hv30」の 8 種類のラベルを，それ以外の録音データには「no (vehicle)」のラベルを，それぞれ付与した．この正解ラベルを用いて，次節で説明する決定木を構築した．

4.3.3　教師あり学習

WEKA はデータマイニングの分野で広く用いられている，強力な分析ツールである．我々は，C4.5 アルゴリズムの Java 実装版である J48 決定木を分類器として採用した．学習データは，駐車場，住宅街，大通りの 3ヶ所で録音したものである．EV と HV の計測は個別に行い，車速は 5km/h，10km/h，20km/h，30km/h の 4 通りとした．それぞれの条件について，最低 6 回ずつ計測を実施した．学習データは，合計で 1552 フレーム分であった．

@Parking Lot		Recognition Result				
		no	ev5	ev10	ev20	ev30
Ground Truth	no	98.81%	0.00%	0.60%	0.30%	0.30%
	ev5(km/h)	2.99%	88.06%	7.46%	0.00%	1.49%
	ev10(km/h)	3.45%	8.62%	84.48%	3.45%	0.00%
	ev20(km/h)	0.00%	2.08%	4.17%	89.58%	4.17%
	ev30(km/h)	14.29%	0.00%	0.00%	4.08%	81.63%

図 4.8: さまざまな車速の EV の録音データでの判定精度

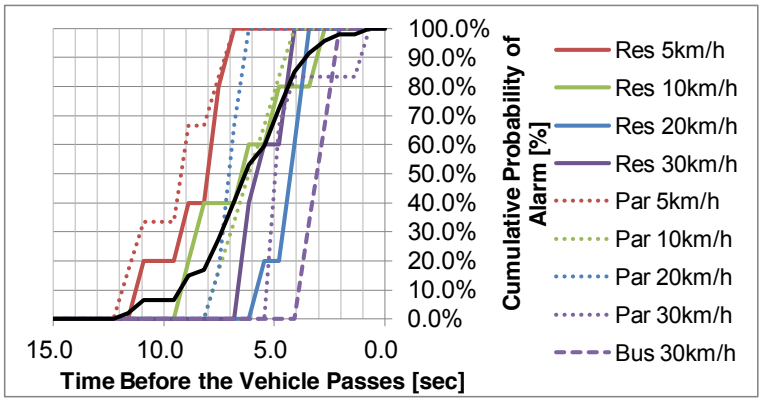

図 4.9: EV 接近前に警告通知が出される累積確率

4.4　評価

4.4.1　EV の検知性能

まず，環境雑音や車速の違いによる，EV の検知精度への影響の評価である．ここでは，前述の 1552 フレーム分の録音データのうち，HV のデータを除く 1350 フレーム分を用いて，10 分割交差検証を行い，検知精度を評価した．

評価結果を表 4.8 に示す．いくつかのデータで車速の誤判定が見受けられるものの，歩行者に車両接近を通知する本アプリケーションの機能としては大きな問題ではない．ここで,「実際には車両接近中でないのに，車両接近中である (ev5,ev10,ev20,ev30) と判定した場合」を False Positive ,「実際には車両接近中であるのに，車両接近中でない (no) と判定した場合」を False Negative と定義する．つまり,「誤検知」を False Positive,「見落とし」を False Negative とすると，False Positive の発生確率は 1.2%，False Negative の発生確率は 4.95%である．

図 4.9 は，検知開始時間の累積分布である．図中の “Res” は住宅街 (Residential Area) を，“Par” は駐車場 (Parking Lot) を，“Bus” は大通り (Busy Street) を表している．図より，最接近 6.4 秒前には 50%の確率で，最接近 3.4 秒前には 90%の確率で，車両の接近を検知できる．また，車速の増加に従って検知開始時間が遅くなる傾向も見られる．音波は距離の 2 乗に比例して減衰するため，走行音が車速の 2 乗に比例して大きくなるなら検知開始時間は一定となるが，実際には 2 乗よりは小さい伸びであると分かる．さらに，環境雑音による検知開始時間の変化も読み取れる．前述の通り，駐車場，住宅街，大通りの環境雑音レベルは，37.6dB，40.6dB，58.7dB であり，環境雑音が小さいほど早く接近車両を検知できる傾向が見られる．今回の実験で，検知開始が最も早かっ

@Parking Lot		Recognition Result								
		no	ev5	ev10	ev20	ev30	hv5	hv10	hv20	hv30
Ground Truth	no	95.92%	0.09%	0.00%	0.09%	0.09%	0.53%	0.18%	0.89%	1.24%
	ev5	0.00%	97.06%	2.94%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	ev10	6.45%	6.45%	83.87%	3.23%	0.00%	0.00%	0.00%	0.00%	0.00%
	ev20	10.34%	0.00%	3.45%	86.21%	0.00%	0.00%	0.00%	0.00%	0.00%
	ev30	11.76%	0.00%	0.00%	5.88%	76.47%	0.00%	0.00%	0.00%	5.88%
	hv5	6.41%	0.00%	1.28%	1.28%	0.00%	70.51%	19.23%	1.28%	0.00%
	hv10	9.76%	0.00%	2.44%	0.00%	0.00%	31.71%	56.10%	0.00%	0.00%
	hv20	27.66%	0.00%	0.00%	0.00%	0.00%	0.00%	2.13%	57.45%	12.77%
	hv30	30.56%	0.00%	0.00%	0.00%	0.00%	2.78%	0.00%	16.67%	50.00%

表 4.3: さまざまな車速の EV と HV の録音データでの判定精度

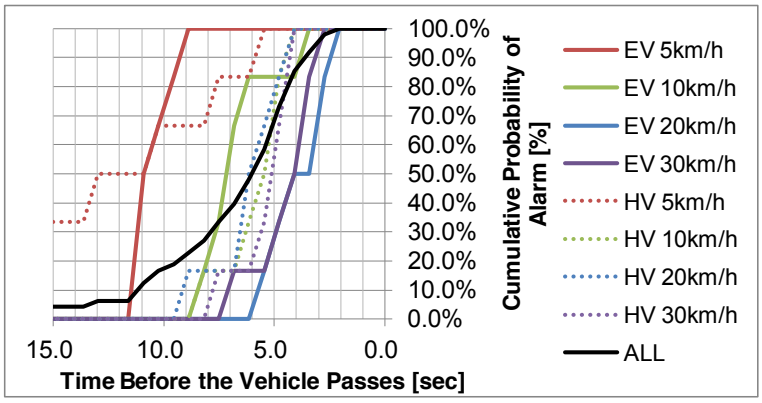

図 4.10: EV・HV 接近前に警告通知が出される累積確率の比較

たのは，住宅街・5km/h のデータにおける 11.6 秒前であり，大通り・30km/h のデータに限ると，3.4 秒前である．

4.4.2　EV と HV の検知性能比較

次に，EV の録音データと HV の録音データが混在する場合の評価である．ここでは，駐車場での録音データのみを用いた．10 分割交差検証による評価結果を図 4.10 に示す．False Positive（誤検知）の発生確率は 4.08%，False Negative（見落とし）の発生確率は 4.87%である．この結果から EV と HV は分類器で明確に区別できることが分かる．これは，モータユニットのスイッチング雑音が，一定の周波数帯に特徴音を持ちながらも，その分布バランスが車種によって異なるためである．

HV では，特に 20km/h，30km/h において，False Negative（見落とし）の発生確率が高い．これは，図 4.5 と図 4.6 を比較した時に，EV は S 字カーブがハッキリ見えるのに対し，HV は S 字カーブがほとんど見えないことに起因している．ただし，図 4.9 に示した検知開始時間の累積分布では，EV と HV で大きな違いは見られず，むしろ，環境雑音や車速による影響の方が大きい．

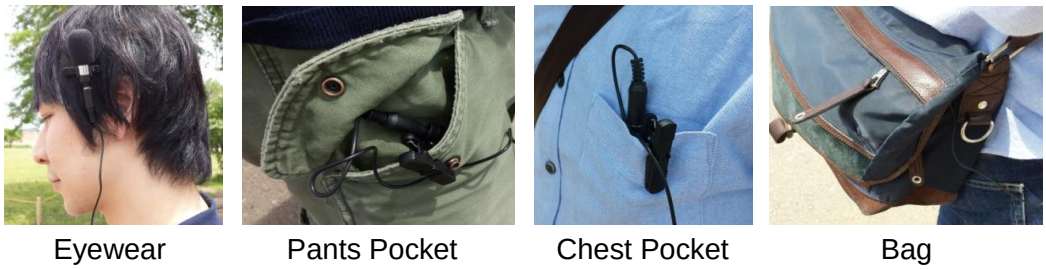

図 4.11: 計測中のピンマイクの装着位置

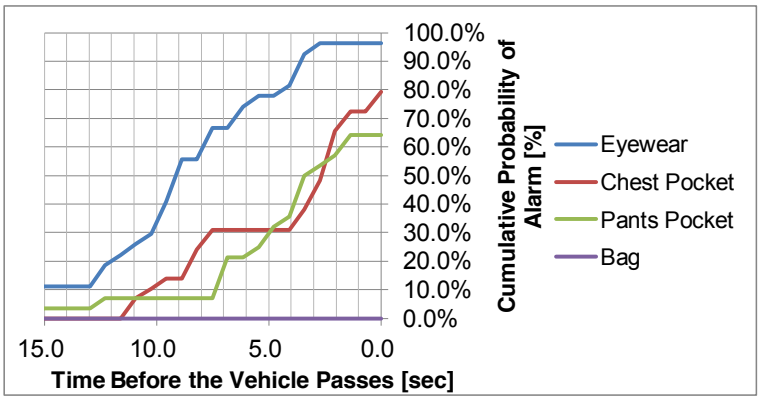

図 4.12: 車両接近前に警告通知が出される累積確率

4.5　他の検討事項

4.5.1　スマートフォンの位置と検知精度

上記の実験では，腰の前の位置に iPod Touch を保持して計測を行ったが，ここでは計測位置が検知精度に及ぼす影響を調べるため，4 台の iPod Touch とピンマイクを用いて，4ヶ所同時に計測を行った．ピンマイクはそれぞれ，眼鏡のツルの位置（Glass 型デバイスやヘッドフォンの装着位置を想定），胸ポケットの中，ズボンのポケットの中，肩掛け鞄の中，の 4ヶ所に配置した（図 4.11）．評価指標としては，4.3 節で構築した決定木に録音データを投入した場合の検知開始時間を採用した．なお，この実験では EV のみを評価対象としている．

図 4.12 は，上記 4ヶ所で録音したデータに対して，本提案手法を適用した際の検知開始時間の累積分布である．眼鏡のツルの位置では，腰の前の位置と同等の精度で判定できる．一方，ポケットの中では，検知開始が遅れる場合や，最接近までに一度も検知できない場合がある．また，鞄の中は音が遮断されており，録音データを人間の耳で聞いても車両の走行音を聞き取ることができず，検知率も 0%となった．

以上の結果から，たとえスマートフォンがポケットや鞄の中にある場合でも，イヤホンマイクを接続することで，スマートフォンを腰の前の位置に保持した場合と遜色ない精度で車両の接近を検知できることが示された．

@Parking Lot Eliminate FN		Recognition Result		
		no	ev	hv
Ground Truth	no	65.46%	13.07%	21.47%
	ev	0.00%	99.10%	0.90%
	hv	7.43%	3.96%	88.61%

図 4.13: False Negative 低減時の判定精度

@Parking Lot 4096 sample / frame		Recognition Result		
		no	ev	hv
Ground Truth	no	96.02%	0.73%	3.25%
	ev	15.57%	83.98%	0.44%
	hv	35.67%	0.38%	63.95%

(1) Frame Size = 4096 Samples (0.085 sec)

@Parking Lot 16384 sample / frame		Recognition Result		
		no	ev	hv
Ground Truth	no	96.82%	0.29%	2.89%
	ev	7.62%	91.93%	0.45%
	hv	24.31%	0.00%	75.69%

(3) Frame Size = 16384 Samples (0.341 sec)

@Parking Lot 8192 sample / frame		Recognition Result		
		no	ev	hv
Ground Truth	no	96.70%	0.54%	2.76%
	ev	10.67%	88.89%	0.44%
	hv	29.02%	0.25%	70.73%

(2) Frame Size = 8192 Samples (0.171 sec)

@Parking Lot 32768 sample / frame		Recognition Result		
		no	ev	hv
Ground Truth	no	95.92%	0.35%	3.73%
	ev	6.31%	92.79%	0.90%
	hv	16.34%	1.49%	82.18%

(4) Frame Size = 32768 Samples (0.682 sec)

図 4.14: フレーム長と判定精度の関係

4.5.2　False Negative の低減

EV や HV の接近検知において，False Positive（誤検知）は，ユーザが後ろを振り返って車両の有無を確認するだけで済むが，False Negative（見落とし）は，交通事故に繋がる危険性があり，相対的にコストが大きい．従って，多少 False Positive を増加させてでも，False Negative を減らすことには価値がある．ここでは，人間の耳でも車両の有無の判別が難しい録音データについて，ラベルを「車両ありの可能性」に変更して学習を行い，「車両なし」と判定にくい決定木を作成した．この決定木を用いた場合には，EV の見落とし確率を 6.3%から 0%に，HV の見落とし確率を 16.3%から 7.4% に低減することができた（図 4.13）．ただし，車両なしの判定精度が 95.9%から 65.5%に落ちたことから，誤警告の回数が増えすぎないよう，前後の判定結果を考慮するロジックを検討する必要がある．

4.5.3　フレーム長と検知精度，消費電力の関係

我々はまだ精密な消費電力の評価を行っていないが，バッテリ駆動時間を伸ばすためには，各処理の間にスリープ時間をなるべく長く取ることが望ましい．我々の検知手法では，前後のフレーム間の相関の情報は使用しないため，各フレームの処理は独立したタイミングで行える．フレームサイズを前述の 0.683 秒（32,768 サンプル）より短くして間欠的に検知処理を行うと，省電力化が期待できる．しかしながら，図 4.14 に示した通り，フレームサイズを短くすると判定精度が低下することが分かっており，判定精度と消費電力はトレードオフの関係にある．

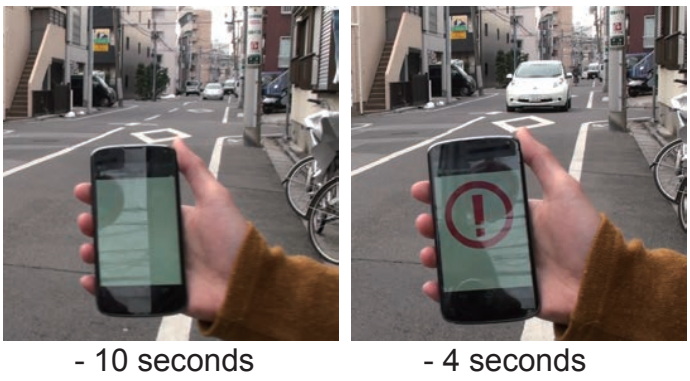

図 4.15: スマートフォンへの実装

4.6　Android 端末と Google Glass への実装

車両接近検知を含むセンシングの分野において，スマートフォンは強力な処理性能を持つデバイスの 1 つである．我々は，上記の車両接近検知アルゴリズムを Android アプリとして実装し，住宅街において Nexus4 で動作検証を行った（図 4.15）．動作検証では，4.2 節で取得した録音データを用いて，事前に分類器を構築し，判定処理のみをスマートフォンで行っている．写真はそれぞれ最接近の 10 秒前と 4 秒前に撮影したものであり，こちらへ向かって来る EV を 4 秒前の時点で検知していることが分かる．

しかしながら，歩行中のスマートフォン利用は，それ自身が交通事故の原因となりうる．つまり，スマートフォンは歩行中のユーザに警告を出すのに適していない．そこで我々は，上記の Android アプリに加えて，Google Glass に警告表示を出すシステムを実装した．

Google Glass 向けのアプリ（Glassware）を開発するには，2 つのアプローチがある．1 つは，Glass Development Kit（GDK）を用いる方法で，音声認識やジェスチャー認識といった低レベルのハードウェア機能を全て利用できる．GDK では，Android SDK とほぼ同等の API セットが利用可能であり，Android アプリ開発のノウハウを活用できる．GDK は我々の車両検知アルゴリズムを Google Glass 上に実装するのに最適な方法であるが，バッテリー寿命や計算リソースに厳しい制約がある．

もう 1 つは，Mirror API を用いる方法で，開発者がクラウド上に実装した Web ベースのサービスに，Google Glass が接続するという形態を取る．簡単のため，今回は Google Glass への通知機能を Mirror API を用いて実装した．Mirror API の短所として，スマートフォンと Google Glass が互いに直接通信できない点が挙げられる．たとえ，スマートフォンと Google Glass が隣り合わせの状態にあっても（さらには，Google Glass がスマートフォンのテザリング機能を通してインターネットへ接続している時でさえ），スマートフォンから Google Glass へのメッセージは一度クラウドサーバへ送信され，携帯電話網または Wi-Fi 経由で Google Glass へと届けられる．

我々はこの「回り道」による遅延時間の評価を行った．Wi-Fi 接続時と LTE 接続時，それぞれ 20 回の平均遅延時間と標準偏差はそれぞれ 3.41 秒（$\sigma = 1.40$ 秒），3.39 秒（$\sigma = 0.71$ 秒）であった．一方，図 4.9 では，大通りの録音データを除くほぼ全てのデータで，最接近の 5 秒前には警告を出せている．従って，Mirror API を用いても最接近までに警告は出せるものの，この遅延時間は GDK を活用するなどして短縮することが望ましい．

4.7　関連研究

本研究は，我々の知る限りにおいて，モータユニットのスイッチング雑音を用いて EV や HV を検知する，初めての研究である．検出ロジックはユーザのスマートフォン上で実行可能であり，インフラ設備によるサポートを必要としない．この節では，EV に限らず，道路上で車両の接近を検知する既存研究を紹介する．これらの研究は互いに相補的な関係にあり，我々の研究成果とも併用可能である．

車両側での歩行者検知は重要な研究領域であり，多数の研究がなされてきたが，歩行者側で接近車両を検知するとなると話は全く異なる [45, 46] ．沖電気工業は，狭域通信を用いて接近車両を検知するスマートフォンを開発した [42]．この手法では，歩行者と車両の双方に狭域通信システムが装備されている必要があり，狭域通信システムが普及しない限り利用できない．また，Car-2-X は，アドホック通信や携帯電話網を用いて歩行者に車両の存在を知らせるシステムであるが，同様の欠点を抱えている [41]．

走行中の車両を検知する研究には，画像処理を用いるアプローチもある．Sivaraman らは，ビデオ映像中の車両を認識し，その移動を追跡する汎用的な学習フレームワークを提案した [47]．The cyber-physical bike も同様のフレームワークであるが，こちらは自転車に特化している [43]．また，Wang らは，スマートフォンのカメラを用いて，歩行者に接近してくる車両を検知する WalkSafe というアプリを提案した [48]．画像処理によるアプローチの長所は，車両を検知するだけでなく，その軌跡から衝突のリスクを評価できる点にある．一方，短所としては，歩行者が意識的にカメラを後方に向けて保持する必要がある．

Tsuzuki らは，LVQ ニューラルネットワークを用いた，携帯電話向けの車両走行音の検知システムを提案した [49]．しかし，残念ながら，彼らは 30km/h 以上で走行するガソリン車で評価を行っており，静音性の高い低速走行中の HV や EV を検知するのには適さない．

第5章

結論

5.1　本論文のまとめ

本研究では，スマートフォンと携帯電話網が世界規模のセンサネットワークとして活用される時代の到来を見据え，(1) 携帯電話網をスマートフォン上のアプリケーションに合わせて最適化し，ネットワーク設備の利用効率の向上を図ること，(2) 従来の参加型センシングにおいてユーザに参加を躊躇わせる原因となっていた，データ送信時のスマートフォンの消費電力および通信量の増加を，ゼロないし無視できる水準に抑えること，(3) スマートフォンのセンサを活用したセンシング技術を確立し，センサデバイスとしての有用性を示すこと，の 3 つの課題に取り組んできた．

課題 (1) については，1,800 ユーザ，のべ 78,200 アプリの通信を分析し，アプリ間で通信頻度に著しい偏りが存在することや，画面点灯状況によってトラヒックの発生確率が大きく変化することなどを明らかにした．また，スマートフォン上で起動中のアプリ一覧と各アプリの通信間隔の分布データから，時々刻々と変化する通信の性質をリアルタイムで予測し，アイドルタイマー長を動的に設定することで，制御信号と消費電力を効果的に削減する手法を提案した．シミュレーション評価では，制御信号を最大 86.1%，通信モジュールの消費電力を最大 37.6%，同時に削減できた．

課題 (2) については，108 ユーザ，81.09GB 分の通信データを解析し，通信量の 1.13%に相当するパディングが存在することを確認した．このパディング領域を活用し，ユーザ端末の消費電力や通信量を増加させることなく，また，携帯電話事業者の保有するネットワーク設備を変更することなく，センサデータを送信する手法を提案した．

課題 (3) については，近年急速に普及し，高い静粛性ゆえに交通事故が多発して社会問題となっている EV と HV について，車両の接近をスマートフォンを用いて検知する技術を開発し，Android 端末と Google Glass 向けのアプリケーションとして実装した．本手法では，モータのスイッチング雑音に着目し，周波数分析と機械学習を用いて接近判定を行う．実際の録音データを用いた評価では，EV を 92.8%，HV を 82.2%の精度で検出できることを確認した．

5.2　本研究の主たる貢献

以下に本研究の主たる貢献をまとめる．

・1,800 ユーザ・78,200 アプリの通信統計の分析

トラヒックの統計データを収集する Android アプリ「通信量お知らせアプリ」を製作し，1,800 ユーザ，のべ 78,200 アプリの通信を分析した．これにより，コネクション数の上位 10 アプリが全通信回数の過半数を占めるなど，トラヒックの性質に対して強い影響力を持つこと，スマートフォンは稼働時間の約 20% しか画面が点灯していないこと，一方で，スマートフォンのトラヒックは約 95% が画面点灯中に集中しており，画面の点灯状況によってトラヒックの発生確率が 80 倍近く変化することを明らかにした．

・制御信号と消費電力の削減を両立する通信制御手法の提案および評価

スマートフォン上で起動中のアプリ一覧と各アプリの通信間隔の分布データから，時々刻々と変化する通信の性質をリアルタイムで予測する手法を提案した．また，制御信号と消費電力のバランスを評価する指標として「アイドルタイマー効率」を定義し，これが最大となるアイドルタイマー

を動的に設定することで，制御信号と消費電力を効果的に削減する手法を提案した．シミュレーション評価では，制御信号を最大 86.1%，通信モジュールの消費電力を最大 37.6% ，同時に削減できることを確認した．

・LTE 網における MAC 層パディング量の推定

Android アプリ「通信量お知らせアプリ with パケットフィルタ」を用いて収集した，108 ユーザ，81.09GB 分の通信データを解析し，通信量の 1.13%に相当するパディングが存在することを確認した．これは，毎月 7GB の通信を行うユーザの場合，毎月 79.1MB に相当する．

・無電力でのセンサデータ送信手法の提案

LTE 網の MAC 層パディングを活用し，ユーザ端末の消費電力や通信量を増加させることなく，センサデータを送信する手法を提案した．また，データを UDP/IP パケットに格納し，ユーザデータと同様のレイヤ 2 処理を施すことで，携帯電話事業者の保有設備に変更を加えることなく，ユーザ端末側の変更だけで実現可能とした．

・機械学習を用いた EV および HV の接近検知手法の提案

本手法では，EV や HV のモータユニットが発するスイッチング雑音に着目し，スマートフォンのマイク入力を周波数分析して機械学習を適用することで，環境雑音や車種の違いにロバストな接近検知手法を実現した．評価実験では，EV を 92.8% ，HV を 82.2%の精度で，最接近の最大 19.8 秒前に検知できることを確認した．

5.3　今後の課題

本研究において，今後解決すべき課題としては以下のものが挙げられる．

・提案手法の実機への実装および評価

本稿で提案した通信制御およびセンサデータ送信の手法は，Android OS や Linux カーネルだけでなく，通信モデムのデバイスドライバも合わせて改変する必要があり，端末メーカの協力なしでは実機への実装が難しい．今後，SONY や京セラなど，有力なスマートフォンメーカと協力関係を築き，実機への実装と評価を行う必要がある．

・超大規模センサネットワークの構築

スマートフォンと携帯電話網を用いて，世界規模のセンサネットワークを構築するにあたっては，送信するデータのフォーマットからサーバの負荷分散方法まで，非常に多くの検討事項が残されている．この構築作業では実現可能性を重視し，他の研究プロジェクトと協力することを視野に入れている．特に，NII や慶應義塾大学，NTT など日欧 13 団体が共同で推進する ClouT プロジェクト [50] は，「センサデータを自由に活用できるプラットフォームの構築」という設計思想が

共通しており，また，ClouT ではセンサデバイスとして専用ハードウェアを想定しているため，スマートフォンをセンサデバイスとして取り込む技術を持ち込むことで，相補的な関係を築くことができると考えている．

謝辞

本研究ならびに修士論文の執筆にあたり，ご多忙にもかかわらず，常に熱心に指導して頂き，多くのアドバイスを下さった浅見徹教授に，心より感謝申し上げます．また，研究テーマの決定から国際学会発表の引率，博士課程への進学相談まで，親身にサポートして下さった川原圭博准教授に，深く感謝いたします．最後に，日々の研究室生活でお世話になりました浅見・川原研究室の皆様にもお礼申し上げます．

参考文献

[1] 東京電力，“スマートメーターについて”． http://www.tepco.co.jp/smartmeter/index-j.html

[2] Ericsson, “Ericsson mobility report”. http://www.ericsson.com/res/docs/2013/emr-august-2013.pdf

[3] Ericsson, “Traffic and market report”. http://www.ericsson.com/res/site_JP/press/2012/doc/Traffic_rep_and_smartphone_issues_120619.pdf

[4] NTT ドコモ，“2012 年 1 月 25 日の foma 音声・パケット通信サービスがご利用しづらい状況について”．http://www.nttdocomo.co.jp/binary/pdf/info/news_release/2012/01/26_00.pdf

[5] A. Carroll and G. Heiser, “An analysis of power consumption in a smartphone,” Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, pp.21–21, USENIXATC’10, USENIX Association, Berkeley, CA, USA, 2010. http://dl.acm.org/citation.cfm?id=1855840.1855861

[6] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, “Diversity in smartphone usage,” Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, pp.179–194, MobiSys ’10, ACM, New York, NY, USA, 2010. http://doi.acm.org/10.1145/1814433.1814453

[7] E. Carolan, S.C. McLoone, and R. Farrell, “Comparing and contrasting smartphone and non-smartphone usage,” Signals and Systems Conference (ISSC 2013), 24th IET Irish, pp.1–6, June 2013.

[8] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, “Periodic transfers in mobile applications: Network-wide origin, impact, and optimization,” Proceedings of the 21st International Conference on World Wide Web, pp.51–60, WWW ’12, ACM, New York, NY, USA, 2012. http://doi.acm.org/10.1145/2187836.2187844

[9] J.-M. Kang, S. seokSeo, and J.-K. Hong, “Usage pattern analysis of smartphones,” Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific, pp.1–8, Sept. 2011.

[10] 高木雅，“通信量お知らせアプリ - google play”．https://play.google.com/store/apps/details?id=akg.traffic.notifier

[11] F. Qian, Z. Wang, A. Gerber, Z.M. Mao, S. Sen, and O. Spatscheck, “Characterizing radio resource allocation for 3g networks,” Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp.137–150, IMC ’10, ACM, New York, NY, USA, 2010. http://doi.acm.org/10.1145/1879141.1879159

[12] GSMA, “Fast dormancy best practices”. http://www.gsma.com/newsroom/ts18-v10-tsg-prd-fast-dormancy-best-practices/

[13] A. Pathak, A. Jindal, Y.C. Hu, and S.P. Midkiff, “What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps,” Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, pp.267–280, MobiSys ’12, ACM, New York, NY, USA, 2012. http://doi.acm.org/10.1145/2307636.2307661

[14] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshp, C. Grunewald, K. Jain, and V.N. Padmanabhan, “Bartendr: A practical approach to energy-aware cellular data scheduling,” In Mobicom, 2010.

[15] 山本享弘，猿渡俊介，森川博之，“Participatory sensing における低消費電力なセンサデータアップロードエンジン,” vol.2010，no.20，情報処理学会研究報告. UBI, [ユビキタスコンピューティングシステム]，jul 2010．http://ci.nii.ac.jp/naid/110007995247/

[16] B. Zhao, Q. Zheng, G. Cao, and S. Addepalli, “Energy-aware web browsing in 3g based smartphones,” Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on, pp.165–175, July 2013.

[17] A. Chakraborty, V. Navda, V.N. Padmanabhan, and R. Ramjee, “Coordinating cellular background transfers using loadsense,” Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, pp.63–74, MobiCom ’13, ACM, New York, NY, USA, 2013. http://doi.acm.org/10.1145/2500423.2500447

[18] 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.,” MOBICOM, eds. by c.B. Akan, E. Ekici, L. Qiu, and A.C. Snoeren, pp.101–112, ACM, 2012. http://dblp.uni-trier.de/db/conf/mobicom/mobicom2012.html#AthivarapuBGNRAPV12

[19] J. Huang, F. Qian, Z.M. Mao, S. Sen, and O. Spatscheck, “Screen-off traffic characterization and optimization in 3g/4g networks,” Proceedings of the 2012 ACM Conference on Internet Measurement Conference, pp.357–364, IMC ’12, ACM, New York, NY, USA, 2012. http://doi.acm.org/10.1145/2398776.2398813

[20] F. Qian, Z. Wang, A. Gerber, Z.M. Mao, S. Sen, and O. Spatscheck, “Top: Tail optimization protocol for cellular radio resource allocation,” Network Protocols (ICNP), 2010 18th IEEE International Conference on, pp.285–294, Oct. 2010.

[21] H. Han, Y. Liu, G. Shen, Y. Zhang, and Q. Li, “Dozyap: Power-efficient wi-fi tethering,” Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, pp.421–434, MobiSys ’12, ACM, New York, NY, USA, 2012. http://doi.acm.org/10.1145/2307636.2307675

[22] F. Xu, Y. Liu, T. Moscibroda, R. Chandra, L. Jin, Y. Zhang, and Q. Li, “Optimizing background email sync on smartphones,” Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services, pp.55–68, MobiSys ’13, ACM, New York, NY, USA, 2013. http://doi.acm.org/10.1145/2462456.2464444

[23] S. Nath, F.X. Lin, L. Ravindranath, and J. Padhye, “Smartads: Bringing contextual ads to mobile apps,” Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services, pp.111–124, MobiSys ’13, ACM, New York, NY, USA, 2013. http://doi.acm.org/10.1145/2462456.2464452

[24] NTT ドコモ，“Technology reports 高速・大容量・低遅延を実現する lte の無線方式概要”. https://www.nttdocomo.co.jp/binary/pdf/corporate/technology/rd/technical_journal/bn/vol19_1/vol19_1_011jp.pdf

[25] NTT ドコモ，“Technology reports lte-advanced － lte のさらなる進化系－ 無線伝送実験”. https://www.nttdocomo.co.jp/binary/pdf/corporate/technology/rd/technical_journal/bn/vol20_2/vol20_2_024jp.pdf

[26] ETSI, “Lte; evolved universal terrestrial radio access (e-utra); packet data convergence protocol(pdcp) specification (3gpp ts 36.323 version 12.1.0 release 12)”. http:// www.etsi.org/deliver/etsi_ts/136300_136399/136323/12.01.00_60/ts_136323v120100p.pdf

[27] ETSI, “Lte; evolved universal terrestrial radio access (e-utra); radio link conrtol (rlc) protocol specification (3gpp ts 36.322 version 12.1.0 release 12)”. http:// www.etsi.org/deliver/etsi_ts/136300_136399/136322/12.01.00_60/ts_136322v120100p.pdf

[28] ETSI, “Lte; evolved universal terrestrial radio access (e-utra); medium access control (mac) protocol specification (3gpp ts 36.321 version 12.3.0 release 12)”. http:// www.etsi.org/deliver/etsi_ts/136300_136399/136321/12.03.00_60/ts_136321v120300p.pdf

[29] 高木雅，“通信量お知らせアプリ with パケットフィルタ - google play”. https://play.google.com/store/apps/details?id=akg.traffic.notifier

[30] ns3, “Lte module - model library”. http:// www.nsnam.org/docs/models/html/lte.html

[31] H. Liu, S. Hu, W. Zheng, Z. Xie, S. Wang, P. Hui, and T. Abdelzaher, “Efficient 3g budget utilization in mobile participatory sensing applications,” INFOCOM, 2013 Proceedings IEEE, pp.1411–1419, April 2013.

[32] M.-R. Ra, B. Liu, T.F. La Porta, and R. Govindan, “Medusa: A programming framework for crowd-sensing applications,” Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, pp.337–350, MobiSys ’12, ACM, New York, NY, USA, 2012. http://doi.acm.org/10.1145/2307636.2307668

[33] T. Das, P. Mohan, V.N. Padmanabhan, R. Ramjee, and A. Sharma, “Prism: Platform for remote sensing using smartphones,” Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, pp.63–76, MobiSys ’10, ACM, New York, NY, USA, 2010. http://doi.acm.org/10.1145/1814433.1814442

[34] N.D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha, “Piggyback crowdsensing (pcs): Energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities,” Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, pp.7:1–7:14, SenSys ’13, ACM, New York, NY, USA, 2013. http://doi.acm.org/10.1145/2517351.2517372

[35] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, “Profiling resource usage for mobile applications: A cross-layer approach,” Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, pp.321–334, MobiSys ’11, ACM, New York, NY, USA, 2011. http://doi.acm.org/10.1145/1999995.2000026

[36] “New car sales ranking”. `http://www.jada.or.jp/contents/data/ranking.html`, (in Japanese).

[37] Commision on measures for quietness of hybrid vehicles, “Report on measures for quietness of hybrid vehicles,” Goverment report of Ministry of Land, Infrastructure, Transport and Tourism, Jan. 2010.

[38] “Hybrid cars are harder to hear,” 2008. University of California, Riverside, Newsroom, `http://newsroom.ucr.edu/news_item.html?action=page&id=1803`.

[39] “Silent prius was exploited for robbery. the victim “didn’t hear the car approaching”,” Asahi Shimbun, April 2010.

[40] “Incidence of pedestrian and bicyclist crashes by hybrid electric passenger vehicles,” 2009. NHTSA Tech. Rep., DOT HS 811 204.

[41] K. David and A. Flach, “Car-2-x and pedestrian safety,” vol.5, no.1, IEEE Vehicular Technology Magazine, March 2010.

[42] L. Oki Electric Industry Co., “Oki succeeds in trial production of world’s first “safety mobile phone” to improve pedestrian safety,” Press release, May 2007.

[43] S. Smaldone, C. Tonde, V.K. Ananthanarayanan, A. Elgammal, and L. Iftode, “The cyber-physical bike: A step towards safer green transportation,” Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, pp.56–61, HotMobile ’11, ACM, New York, NY, USA, 2011. http://doi.acm.org/10.1145/2184489.2184502

[44] A. Cassat, C. Espanet, R. Coleman, E. Leleu, L. Burdet, D. Torregrossa, J. M’Boua, and A. Miraoui, “Forces and vibrations analysis in industrial pm motors having concentric windings,” IEEE Energy Conversion Congress and Exposition (ECCE), pp.2755–2762, Sept. 2010.

[45] F. Bu and C.Y. Chan, “Pedestrian detection in transit bus application: sensing technologies and safety solutions,” Proceedings of IEEE Intelligent Vehicles Symposium, pp.100–105, June 2005.

[46] T. Gandhi and M.M. Trivedi, “Pedestrian protection systems: Issues, survey, and challenges,” vol.8, no.3, IEEE Transactions on Intelligent Transportation Systems, Sept. 2007.

[47] S. Sivaraman and M.M. Trivedi, “A general active-learning framework for on-road vehicle recognition and tracking,” vol.11, no.2, IEEE Transactions on Intelligent Transportation Systems, June 2010.

[48] T. Wang, G. Cardone, A. Corradi, L. Torresani, and A.T. Campbell, “Walksafe: A pedestrian safety app for mobile phone users who walk and talk while crossing roads,” Proceedings of the Twelfth Workshop on Mobile Computing Systems and Applications, pp.5:1–5:6, HotMobile ’12, ACM, New York, NY, USA, 2012. http://doi.acm.org/10.1145/2162081.2162089

[49] H. Tsuzuki, M. Kugler, S. Kuroyanagi, and A. Iwata, “A novel approach for sound approaching detection,” Neural Information Processing. Models and Applications, eds. by K. Wong, B.SumuduU. Mendis, and A. Bouzerdoum, vol.6444, pp.407–414, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010. http://dx.doi.org/10.1007/978-3-642-17534-3_50

[50] ClouT, “Cloud of things for empowering the citizen clout in smart cities”. http://clout-project.eu/ja/

[51] G. Solutions, “G-nettrack phone measurement capabilities survey results”. http://www.gyokovsolutions.com/survey/surveyresults.php

[52] T. software, “tpacketcapture - android ソフトウェア”. http://www.taosoftware.co.jp/android/packetcapture/

発表文献

国内ジャーナル

[1] 高木雅，川原圭博，浅見徹，“アプリ毎のトラヒ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手法，” 情処学論，vol.56, no.3 (March, 2015).

国際会議

[2] M. Takagi, K. Fujimoto, Y. Kawahara and T. Asami, “Detecting hybrid and electric vehicles using a smartphone,” Ubicomp 2014, pp.267-275, Sept. 2014.

研究会

[3] 高木雅，川原圭博，浅見徹，“3G トラヒック削減とプッシュ通知を共存させるアプリ単位の通信制御手法,” DICOMO2013 論文集, 7F-2, vol.2013, pp.1727-1734, Jul. 2013.

[4] 高木雅，藤本浩介，川原圭博，浅見徹，“機械学習によるスマートフォンを用いた電気自動車の接近検知手法,” 信学技報, vol.114, no.65, ASN2014-20, pp.67-68, May 2014.

[5] 高木雅，藤本浩介，川原圭博，浅見徹，“スマートフォンを用いた電気自動車およびハイブリッド車の接近検知手法,” 情処研究報告, vol.2014-UBI-43(12), pp.1-8, Jul. 2014.

国内全国大会

[6] 高木雅，川原圭博，浅見徹，“ユーザ操作に基づく動的な Idle Timer による 3G 制御信号の削減手法，” 信学ソ大，B-15-2, Sept. 2013.

[7] 藤本浩介，高木雅，川原圭博，浅見徹，“スマートフォンのマイクロフォンを用いた電気自動車接近時の高周波音の計測，” 信学ソ大，B-15-9, Sept. 2013.

[8] 高木雅，川原圭博，浅見徹，“スマートフォン・トラヒックのアプリ単位での分析，” 信学総大，B-15-7, March 2014.

[9] 藤本浩介，高木雅，川原圭博，浅見徹，“電気自動車の特徴音を用いたスマートフォンによる車両接近検知手法，” 信学総大，B-15-6, March 2014.

[10] 高木雅，川原圭博，浅見徹，“LTE 網の MAC 層パディングを活用した無電力でのセンサデータ送信，” 信学総大，March 2015 (To appear).

付録

Android 端末における機種別の LTE 通信パラメータ取得可否

Android SDK には，LTE ネットワークの通信パラメータを取得するための `PhoneStateListener API` が用意されているが，API が正しい値を返すかどうかは OS バージョンやメーカの実装状況に依存する [51]．表 5.1 は，機種別の通信パラメータ取得可否であり，表中の数値は API が正しく実装されておらず，常に一定値を返すことを示す（“MAX” は Integer 型の最大値）．なお，Samsung や LG の端末には，いわゆる「ServiceMode」アプリが内蔵されており，より詳細な通信パラメータを閲覧可能であるが，自前のアプリからパラメータを取得する方法は発見されていない．

表 5.1: 機種別の LTE 通信パラメータ取得可否

Maker	Model	OS	SS	RSRP	RSRQ	CQI	RSSNR
LG	Neuxs5 EM01L (LG-D821)	5.0.1	○	○	○	MAX	300
Fujitsu	ARROWS X LTE F-05D	4.0.3	×	×	×	×	×
Fujitsu	ARROWS X F-10D	4.2.2	×	○	○	MAX	○
Fujitsu	ARROWS X F-02E	4.1.2	×	×	×	×	×
Fujitsu	ARROWS Kiss F-03E	4.1.2	×	×	×	×	×
Fujitsu	ARROWS ef FJL21	4.1.2	×	×	×	×	×
Fujitsu	ARROWS Z FJL22	4.2.2	×	MAX	MAX	MAX	MAX
HTC	HTC J Butterfly HTL21	4.1.1	○	○	○	MAX	○
HTC	HTC J One HTL22	4.4.2	○	○	○	MAX	○
HTC	HTC J One HTL22	4.2.2	○	○	○	MAX	○
Huawei	Ascend HW-01E	4.0.4	×	×	×	×	×
Kyocera	DIGNO S KYL21	4.0.4	×	×	×	×	×
Kyocera	URBANO L01	4.2.2	×	○	○	MAX	×
LG	Optimus it L-05D	4.1.2	×	○	○	MAX	×
LG	Optimus Vu L-06D	4.0.4	×	○	○	MAX	×
LG	Optimus it L-05E	4.2.2	×	○	○	MAX	○
LG	G2 L-01F	4.2.2	×	○	○	MAX	○
LG	Optimus G LGL21	4.0.4	×	○	○	MAX	×
LG	isai LGL22	4.4.2	○	○	○	MAX	○
LG	G Flex LGL23	4.2.2	×	MAX	MAX	MAX	×
LG	isai FL LGL24	4.4.2	○	○	○	MAX	○
NEC	MEDIAS TAB N-06D	4.0.4	○	×	×	×	×
NEC	MEDIAS X N-07D	4.1.2	○	×	×	×	×
NEC	MEDIAS TAB UL N-08D	4.0.4	○	×	×	×	×
NEC	MEDIAS U N-02E	4.1.2	○	×	×	×	×
NEC	Disney Mobile on docomo N-03E	4.1.2	○	×	×	×	×
NEC	MEDIAS W N-05E	4.1.2	○	×	×	×	×
Pantech	VEGA PTL21	4.1.2	×	×	×	×	×
Samsung	GALAXY Tab 10.1 LTE SC-01D	4.0.4	×	○	○	MAX	×
Samsung	GALAXY Note SC-05D	4.1.2	×	○	○	MAX	×
Samsung	GALAXY S3 SC-06D	4.1.2	×	○	○	MAX	×
Samsung	GALAXY Tab 7.7 Plus SC-01E	4.0.4	×	○	○	MAX	×
Samsung	GALAXY Note2 SC-02E	4.3	○	○	○	-1	×
Samsung	GALAXY S4 SC-04E	4.4.2	○	○	○	-1	○
Samsung	GALAXY Note3 SC-01F	4.4.2	○	○	○	-1	○
Samsung	Galaxy J SC-02F	4.4.2	○	○	○	-1	○
Samsung	GALAXY S3 Progre SCL21	4.1.2	×	○	○	MAX	×
SHARP	スマートフォン for ジュニア SH-05E	4.0.4	×	×	×	×	×
SHARP	AQUOS PHONE SERIE SHL21	4.1.2	×	×	×	×	×
SHARP	AQUOS SERIE SHL25	4.4.2	MAX	MAX	MAX	MAX	99
SONY	XPERIA GX SO-04D	4.1.2	×	×	×	×	×
SONY	XPERIA SX SO-05D	4.1.2	×	×	×	×	×
SONY	XPERIA A SO-04E	4.2.2	×	○	○	MAX	○
SONY	XPERIA Z1f SO-02F	4.4.2	○	○	○	MAX	○
SONY	XPERIA A2 SO-04F	4.4.2	○	○	○	MAX	○
SONY	XPERIA VL SOL21	4.1.2	×	×	×	×	×
SONY	XPERIA UL SOL22	4.2.2	×	○	○	MAX	○
SONY	XPERIA Z Ultra SOL24	4.4.2	○	○	○	MAX	○

VpnService API を用いたパケットキャプチャ機能の実装方法

本研究では，一般のスマートフォンユーザに対して Google PLAY を通して調査用の Android アプリを配信し，トラヒックの統計的な分析を行ってきた．ここでは，root 権限を必要とせず，Android SDK が提供する API のみを使用して，パケットキャプチャ機能およびパケットフィルタリング機能を実装する方法を紹介する．具体的には，`VpnService API` を利用し，端末上で送受されるあらゆる IP パケットを傍受する．実装にあたり，Tao software 社の Web サイト，および，同社のパケットキャプチャアプリ「tPacketCapture」の挙動が非常に参考になったことを記しておく [52]．

VpnService API の標準的な利用法

Android 4.0 から追加された `VpnService API` は，アプリ開発者が独自の L2TP（Layer 2 Transparent Protocol）の VPN 接続を構成し，機能を拡張するための API である．従来から，Android OS には VPN 接続機能が搭載されているが，`VpnService API` を利用することで，特定の宛先へのパケットを破棄するなど，より高度な制御が可能となる．

`VpnService API` は，仮想ネットワークインタフェースの作成と，IP アドレスおよびルーティング情報の設定を行い，送信 IP パケット読み出し用と受信 IP パケット書き込み用のファイルディスクリプタをアプリ開発者に提供する．アプリ開発者は，VPN サーバへのトンネル接続用のソケットを確立して API に登録（protect）し，ファイルディスクリプタとトンネルソケットの間を取り持つ（単にバイト配列をコピーするだけの）コードを記述すれば，VPN の機能を実現できる．ここまでの内容に対応するサンプルコードが，Android SDK に付属している（ToyVpn）．

ファイルディスクリプタとソケットの間を流れるのはバイナリデータであるが，これを解析することで IP パケットを抽出できる．時刻とパケットの内容を記録すればパケットキャプチャ機能を，特定の条件を満たすパケットを破棄すればパケットフィルタリング機能を実現できる（図 5.1）．

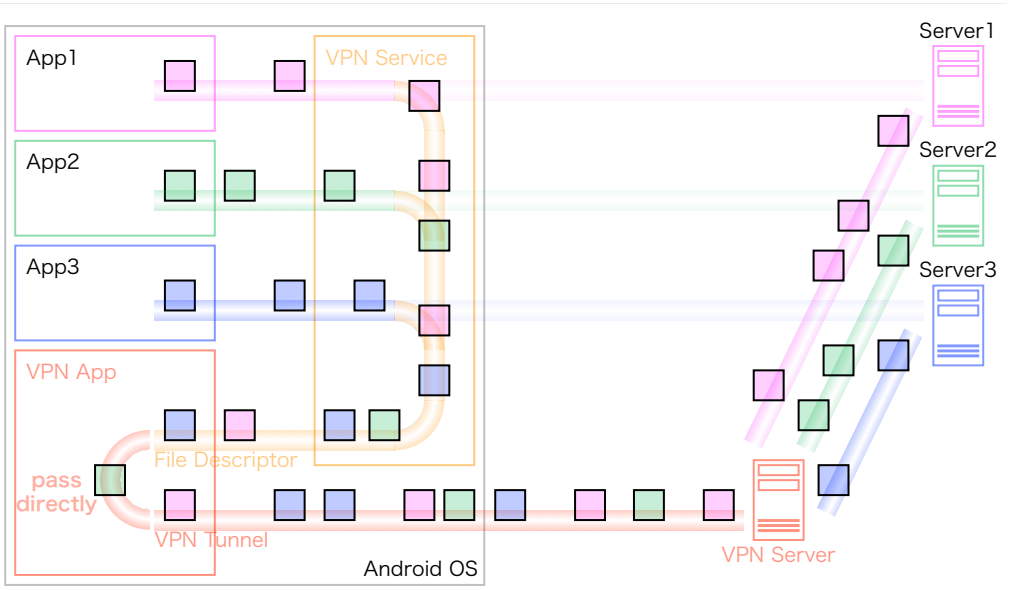

図 5.1: VpnService API の標準的な利用法の概要図

VPN サーバを使用せず Android アプリのみで完結させる方法

前節の方法では，全てのトラヒックが VPN サーバを経由するため，(1) 通信に遅延が生じる，(2) 端末からローカルネットワークにアクセスできなくなる，(3) 接続する端末数に比例して VPN サーバへの負荷が増大する，といった問題が発生する（これらの問題が無視できるのであれば，VPN サーバ上で Wireshark 等のパケットキャプチャソフトを実行するだけで十分である）．

そこで本研究では，`VpnService API` から受信した IP パケットを解析し，ソケット通信で宛先サーバと直接通信を行うことで，VPN サーバを使用せず Android アプリのみで通信処理を完結させる実装を行った（図 5.2）．TCP 通信については，新たな宛先へのパケットを検知する度にソケットを新たに作成し，定期的に全ソケットの受信バッファを確認する．UDP 通信については，単一のソケットで複数の宛先サーバと通信できるため，動的にソケットを生成する必要はないが，送信元の IP アドレスやポート番号を管理するための NAT テーブルを自前で実装する必要がある．なお，他のトランスポート層（レイヤ 4）プロトコルについては，実装を省略した．以下に擬似コードを掲載する．

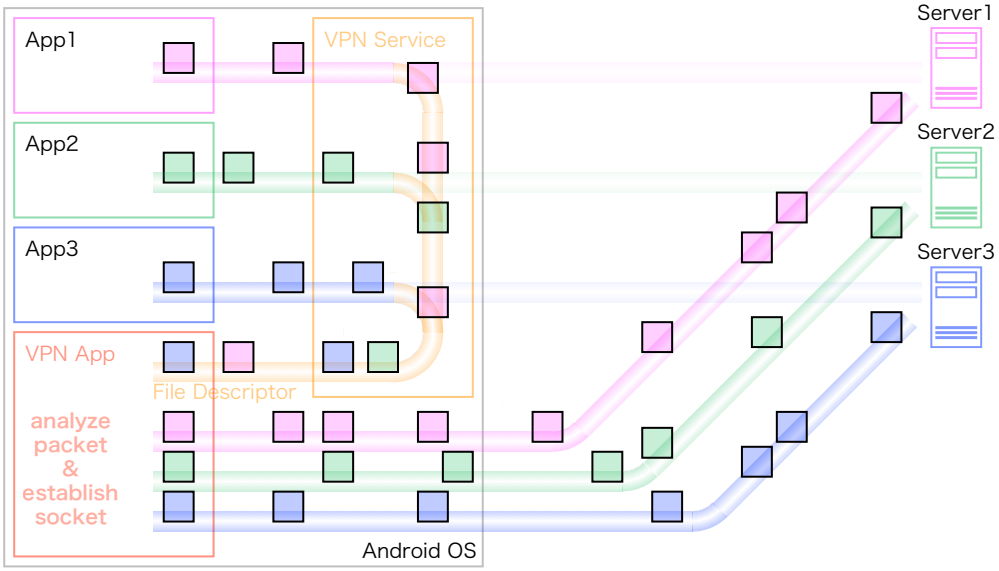

図 5.2: VpnService API を活用したパケットキャプチャの概要図

ソースコード 5.1: VpnSerice API を活用したパケットキャプチャアプリの擬似コード

```
1  // VPN Interface のセットアップ（設定値はダミーで良い）
2  PendingIntent mConfigureIntent;
3  Builder builder = new Builder();
4  builder.setMtu(1400);
5  builder.addAddress("10.0.0.2", 32);
6  builder.addRoute("0.0.0.0", 0);
7  builder.addDnsServer("8.8.8.8");
8  builder.setSession("133.11.236.1");
9  builder.setConfigureIntent(mConfigureIntent);
10 ParcelFileDescriptor mInterface = builder.establish();
11
12 // VPN を迂回する UDP ソケットを作成
13 DatagramSocket datagramSocket = new DatagramSocket();
14 if (!protect(datagramSocket)) {
15     throw new IllegalStateException("Cannot␣protect␣the␣tunnel");
```

```
16 }
17
18 // 送信パケットをここから読み出す
19 FileInputStream in = new FileInputStream(mInterface.getFileDescriptor());
20 // 受信パケットをここに書き込む
21 FileOutputStream out = new FileOutputStream(mInterface.getFileDescriptor());
22
23 // TCP ソケットのリスト
24 List<SocketItem> socketList = new ArrayList<SocketItem>();
25 // UDP 用の NAT テーブル
26 List<NATItem> natList = new ArrayList<NATItem>();
27
28 // パケット処理用のバッファを用意
29 ByteBuffer byteBuffer = ByteBuffer.allocate(32767);
30 // 送信パケットのループまたぎ用のバッファ
31 byte[] txCarryover = new byte[0];
32 // UDP パケット受信用のバッファ
33 byte[] socketRxBuffer = new byte[32768];
34
35 while(true){
36     /* - - - - - 送信パケット - - - - - */
37     // 前回のループからの繰り越し分
38     byteBuffer.position(0);
39     byteBuffer.put(txCarryover);
40     int length = txCarryover.length;
41
42     // 送信バッファの読み込み
43     length += in.read(byteBuffer.array(), txCarryover.length, byteBuffer.capacity()
           - txCarryover.length);
44     if (length > 0) {
45         while (true) {
46             // バイト列を解析して IP パケットを抽出
47             IPPacket ipPacket = new IPPacket(byteBuffer);
48             if (ipPacket.isValid) {
49
50                 // IP パケットを記録する
51                 recordIPPacket(ipPacket);
52
53                 if (ipPacket.protocol == IPPacket.TCP) {
54                     // IP パケットの Payload から TCP パケットを抽出
55                     TCPPacket tcpPacket = new TCPPacket(ipPacket.payload);
56
57                     /* - - - TCP 制御パケット - - - */
58                     if (tcpPacket.syn) {
59                         // TCP ソケットのリストを検索
60                         SocketItem socketItem = null;
61                         for (SocketItem item : socketList) {
62                             if (ipPacket.srcAddr.equals(item.localIP) && tcpPacket.
                                   srcPort == item.localPort && ipPacket.dstAddr.equals(
                                   item.remoteIP) && tcpPacket.dstPort == item.
                                   remotePort) {
63                                 socketItem = item;
64                                 break;
65                             }
66                         }
67                         // 無ければ作成
68                         if (socketItem == null) {
69                             // VPN を迂回する TCP ソケットを作成
70                             socketItem = new SocketItem(localIP, localPort, remoteIP,
                                    remotePort);
71                             if (!protect(socketItem.channel.socket())) {
72                                 throw new IllegalStateException("Cannot␣protect␣the␣
                                       tunnel");
73                             }
74                             socketList.add(socketItem);
75                         }
76
77                         // ACK 番号を加算
78                         socketItem.ackNumber = tcpPacket.sequenceNumber + tcpPacket.
```

```
                        payload.length+1;
79
80                  // SYN+ACK パケットを送信
81                  IPPacket synAckPacket = IPPacket.makeSynAck(socketItem);
82                  out.write(synAckPacket.toBinary(), 0, synAckPacket.length());
83
84                  // シーケンス番号を加算
85                  socketItem.sequenceNumber += 1;
86              }
87              else if(tcpPacket.fin && tcpPacket.ack) {
88                  // TCP ソケットのリストを検索
89                  SocketItem socketItem = null;
90                  for (SocketItem item : socketList) {
91                      if (ipPacket.srcAddr.equals(item.localIP) && tcpPacket.
                            srcPort == item.localPort && ipPacket.dstAddr.equals(
                            item.remoteIP) && tcpPacket.dstPort == item.
                            remotePort) {
92                          socketItem = item;
93                          break;
94                      }
95                  }
96                  // TCP ソケットがあれば切断
97                  if (socketItem != null) {
98                      // ACK パケットを送信
99                      IPPacket ackPacket = IPPacket.makeAck(socketItem);
100                     out.write(ackPacket.toBinary(), 0, ackPacket.length());
101
102                     // TCP ソケットを削除
103                     socketList.remove(socketItem);
104                 }
105             }
106             else if(tcpPacket.fin) {
107                 // TCP ソケットのリストを検索
108                 SocketItem socketItem = null;
109                 for (SocketItem item : socketList) {
110                     if (ipPacket.srcAddr.equals(item.localIP) && tcpPacket.
                            srcPort == item.localPort && ipPacket.dstAddr.equals(
                            item.remoteIP) && tcpPacket.dstPort == item.
                            remotePort) {
111                         socketItem = item;
112                         break;
113                     }
114                 }
115                 // TCP ソケットがあれば切断
116                 if (socketItem != null) {
117                     // ACK 番号を加算
118                     socketItem.ackNumber = tcpPacket.sequenceNumber +
                            tcpPacket.payload.length+1;
119
120                     // FIN+ACK パケットを送信
121                     IPPacket finAckPacket = IPPacket.makeFinAck(socketItem);
122                     out.write(finAckPacket.toBinary(), 0, finAckPacket.length
                            ());
123
124                     // TCP ソケットを削除
125                     socketList.remove(socketItem);
126                 }
127             }
128
129             /* - - - TCP データパケット - - - */
130             if (tcpPacket.payload.length > 0) {
131                 // TCP ソケットのリストを検索
132                 SocketItem socketItem = null;
133                 for (SocketItem item : socketList) {
134                     if (ipPacket.srcAddr.equals(item.localIP) && tcpPacket.
                            srcPort == item.localPort && ipPacket.dstAddr.equals(
                            item.remoteIP) && tcpPacket.dstPort == item.
                            remotePort) {
135                         socketItem = item;
136                         break;
```

```
137                         }
138                     }
139                     // 無ければ作成
140                     if (socketItem == null) {
141                         // VPN を迂回する TCP ソケットを作成
142                         socketItem = new SocketItem(localIP, localPort, remoteIP,
                                remotePort);
143                         if (!protect(socketItem.channel.socket())) {
144                             throw new IllegalStateException("Cannot␣protect␣the␣
                                    tunnel");
145                         }
146                         socketList.add(socketItem);
147                     }
148 
149                     // データ送信
150                     socketItem.channel.write(ByteBuffer.wrap(tcpPacket.payload));
151 
152                     // ACK 番号を加算
153                     socketItem.ackNumber = tcpPacket.sequenceNumber+tcpPacket.
                            payload.length;
154 
155                     // SYN+ACK パケットを送信
156                     IPPacket synAckPacket = IPPacket.makeSynAck(socketItem);
157                     out.write(synAckPacket.toBinary(), 0, synAckPacket.length());
158                 }
159             }
160             else if (ipPacket.protocol == IPPacket.UDP) {
161                 // IP パケットの Payload から UDP パケットを抽出
162                 UDPPacket udpPacket = new UDPPacket(ipPacket.payload);
163 
164                 // NAT テーブルを検索（エントリーが無ければ作成）
165                 NATItem natItem = null;
166                 for (NATItem item : natList) {
167                     if (ipPacket.dstAddr.equals(item.remoteIP) && udpPacket.
                            dstPort == item.remotePort) {
168                         // NAT テーブルを更新
169                         item.localIP = localIP;
170                         item.localPort = localPort;
171                         natItem = item;
172                         break;
173                     }
174                 }
175                 if (natItem == null) {
176                     natItem = new NATItem(ipPacket.srcAddr, udpPacket.srcPort,
                            ipPacket.dstAddr, udpPacket.dstPort);
177                     natList.add(natItem);
178                 }
179 
180                 // ソケットで送信
181                 DatagramPacket dPacket = new DatagramPacket(udpPacket.payload,
                        udpPacket.payload.length, natItem.remoteIP, natItem.
                        remotePort);
182                 datagramSocket.send(dPacket);
183             }
184             else{
185                 // TCP でも UDP でもないパケットは無視する（将来的には実装すべき）
186             }
187         }else{
188             // buffer にコピーして次のループへ
189             txCarryover = new byte[length - byteBuffer.position()];
190             System.arraycopy(byteBuffer.array(), byteBuffer.position(),
                    txCarryover, 0, copyLength);
191             break;
192         }
193     }
194 }
195 
196 /* - - - - - 受信パケット (TCP) - - - - - */
197 // 全ての TCP ソケットを確認する
198 for (SocketItem socketItem : socketList) {
```

```
199         // TCP ソケットから読み込み
200         length = socketItem.channel.read(byteBuffer);
201         if (length > 0) {
202             for (int i = 0; i <= (length-1)/1460; i++) {
203                 // IP パケットサイズの上限（1460byte）に収まるよう小分けにする
204                 int packetSize = (i == (length-1)/1460 ? length%1460 : 1460);
205                 byte[] buffer = new byte[packetSize];
206                 System.arraycopy(byteBuffer.array(), i*1460, buffer, 0, packetSize);
207 
208                 // TCP/IP パケットの生成
209                 TCPPacket tcpPacket = new TCPPacket(socketItem, buffer);
210                 IPPacket ipPacket = new IPPacket(socketItem, tcpPacket.toBinary(),
                        IPPacket.TCP);
211 
212                 // 受信ストリームに書き込み
213                 out.write(ipPacket.toBinary(), 0, ipPacket.length());
214 
215                 // シーケンス番号を加算
216                 socketItem.sequenceNumber += packetSize;
217             }
218         }
219     }
220 
221     /* - - - - - 受信パケット (UDP) - - - - - */
222     // 単一の UDP ソケットのみ確認すれば良い
223     DatagramPacket dPacket = new DatagramPacket(socketRxBuffer, 0, socketRxBuffer.
            length);
224     datagramSocket.setSoTimeout(10); // ブロックしてしまうので 10
            ms で受信タイムアウトさせる
225     try {
226         // UDP ソケットから読み込み
227         datagramSocket.receive(dPacket);
228         length = dPacket.getLength();
229 
230         // NAT テーブルを検索して、宛先 IP アドレス・ポート番号を割り出す
231         NATItem natItem = null;
232         for (NATItem item : natList) {
233             if (dPacket.getAddress().equals(item.remoteIP) && dPacket.getPort() ==
                    item.remotePort) {
234                 natItem = item;
235                 break;
236             }
237         }
238 
239         for (int i = 0; i <= (length-1)/1460; i++) {
240             // IP パケットサイズの上限（1460byte）に収まるよう小分けにする
241             int packetSize = (i == (length-1)/1460 ? length%1460 : 1460);
242             byte[] buffer = new byte[packetSize];
243             System.arraycopy(socketRxBuffer, i*1460, buffer, 0, packetSize);
244 
245             // UDP/IP パケットの生成
246             UDPPacket udpPacket = new UDPPacket(natItem, buffer);
247             IPPacket ipPacket = new IPPacket(natItem, udpPacket.toBinary(), IPPacket
                    .UDP);
248 
249             // 受信ストリームに書き込み
250             out.write(ipPacket.toBinary(), 0, ipPacket.length());
251         }
252     }catch(InterruptedIOException e){
253         // 受信パケットは無かった
254     }
255 }
```