IoTアプリ構築支援のためのSINETStream Android用プラグインの開発

孫 静涛[1,a)] 竹房 あつ子[1,2,b)] 藤原 一毅[1,c)] 吉田 浩[1,d)] 合田 憲人[1,2,e)]

概要：第5世代移動通信システム「5G」の実用化とIoT(Internet of Things)デバイスの小型化・安価化を背景に，モバイルネットワークを介してIoTデバイスから収集された大量な情報をクラウドに蓄積し，機械学習を用いた高度なデータ解析への期待が高まっている．国立情報学研究所はIoTアプリ開発を支援するために，広域データ収集・解析プログラム開発支援ソフトウェアパッケージSINETStreamの開発を進めている．SINETStreamは，Raspberry PiのようなLinuxベースのセンサ端末を前提として開発が進められているが，多様なセンサやバッテリが装備されているスマートフォンには対応していない．本研究では，スマートフォンのような多様なセンサデータの収集・解析を可能にするため，SINETStream Androidプラグインを新たに開発し，Android端末を用いた広域データ収集を可能にする．

キーワード：広域データ，IoT，Android，ストリーム処理，モバイルアプリケーション

An Implemention of SINETStream Android Plugin to Support IoT Application Construction

Jingtao Sun[1,a)] Atsuko Takefusa[1,2,b)] Ikki Fujiwara[1,c)] Hiroshi Yoshida[1,d)] Kento Aida[1,2,e)]

1. はじめに

第5世代移動通信システム「5G」の実用化とIoT(Internet of Things)デバイスの小型化・安価化を背景に，モバイルネットワークを介してIoTデバイスから収集された大量な情報をクラウドに蓄積し，機械学習を用いた高度なデータ解析への期待が高まっている．国立情報学研究所（NII）では，学術情報ネットワークSINETの足回りとしてモバイル網を活用した「SINET広域データ収集基盤」を構築し，センサ端末をSINETのVPNに直接接続可能にする機能を提供している．しかしながら，各応用分野の研究者がデータの収集や解析を行うためのIoTアプリケーションを開発するには，ネットワークに関する高度な知識やプログラミングスキルが必要とされ，容易ではない．

我々はIoTアプリ開発を支援するために，広域データ収集・解析プログラム開発支援ソフトウェアパッケージSINETStream[8]の開発を進めている．既存メッセージング基盤ソフトウェアとして，Apache Kafka[1]やMQTTベースのブローカ，例えば，HiveMQ[2]やMosquitto[3]，VerneMQ[4]等，多数開発されている．また，商用サービスでもAmazon[5]やGoogle[6]，Azure[7]等でIoTプラットフォームが提供されている．しかしながら，各ソフトウェアやプラットフォームには性能面、プロトコルオーバヘッド，コスト面など特徴が異なり，IoTアプリケーションの要求に応じて使い分ける必要がある．そこで，SINETStreamではメッセージング基盤を抽象化したAPI（Application Programming Interface）を提供し，IoTアプリケーションの要求に応じて適切なブローカソフトウェアまたはクラウドPub-Subサービスを利用可能にする．また，IoTアプリケーションに必要とされる認証・認可やデータの暗号化等のセキュリティ機能や，任意のブローカに対応するためのSPI（Service Provider Interface）を提供している[13]．しかし，現状ではSINETStreamはRaspberry PiのようなLinuxベースのセンサ端末を前提として開発が進められて

[1] 国立情報学研究所, 101-8430, 東京都千代田区一ツ橋２－１－２
[2] 総合研究大学院大学, 240-0193, 神奈川県三浦郡葉山町湘南国際村
[a)] sun@nii.ac.jp
[b)] takefusa@nii.ac.jp
[c)] ikki@nii.ac.jp
[d)] h-yoshida@nii.ac.jp
[e)] aida@nii.ac.jp

おり，多様なセンサやバッテリが装備されているスマートフォンには対応していない．

本研究では，スマートフォンのような多様なセンサデータの収集・解析を可能にするため，SINETStream Android プラグインを新たに開発し，Android 端末を用いた広域データ収集を容易にする．SINETStream Android プラグインでは，既存の SINETStream ライブラリで提供するメッセージの送受信のための API に加え，多様なセンサの情報を容易に収集するためのライブラリの提供も行う．

本稿では，まず Android での IoT アプリケーション構築事例を説明したのち，SINETStream Android プラグインの設計と実装の詳細について述べる．以降は，以下の章から構成される．第 2 章は，Android での IoT アプリケーション構築事例を説明する．また，第 3 章では，SINETStream の概要を紹介した上で，第 4 章で SINETStream Android プラグインの API 設計と使用例を紹介する．最後に，関連研究との比較を述べてからまとめと今後の予定を述べる．

2. Android での IoT アプリケーション構築事例

本稿では，「SINET 広域データ収集基盤」上で Android 端末を用いた IoT アプリケーション構築事例 [14] を紹介する．

2.1 IoT アプリケーションのシステム構成

「SINET 広域データ収集基盤」（SINET モバイル）では，SINET の L2 VPN で閉域網を構築し，安全かつ高性能なネットワーク環境下で IoT アプリケーションを構築することができる．センサデータを効率よく収集するため，一般に Apache Kafka ようなメッセージングシステムを利用する．しかし，我々の知る限り Android 端末向けの Kafka ライブラリは用意されておらず，Android 端末から Kafka を利用するにはアプリケーション開発者の負担が大きい．そこで，軽量な MQTT Mosquitto ブローカ [3], [12] を Kafka のフロントエンドに配備して Android 端末からのセンサデータを収集する IoT アプリケーションプロトタイプシステムを構築した．分散メッセージング基盤として，MQTT ブローカのみを利用する方法も考えられるが，性能面や他のストリーム処理基盤やデータベースとの連携を想定し，Kafka をバックエンドで利用することとした．

図 1 にプロトタイプシステムの構成を示す．プロトタイプシステムでは，図の左下に示された Android 端末から SINET モバイル経由でセンサデータを MQTT Mosquitto ブローカに送信する．Android 端末で収集されたセンサデータは，そのままの数値データで送るのではなく，各種センサから収集された数値データをトピックごとに分け，デバイス情報と位置情報とデータの収集時間等を追記した JSON 形式に変換してから Mosquitto ブローカに送信される．Mosquitto ブローカに到着したセンサデータは，Kafka Connect を用いて Kafka クラスタに送信される．本プロトタイプシステムの Kafka クラスタ構成は 3 台のブローカーサーバと 1 台の ZooKeeper サーバからなる．また，Kafka コネクタとして MQTT コネクタ [10] と Elasticsearch[11] コネクタを用いた．前者は，Mosquitto ブローカに到着したセンサデータを Kafka システムに転送する．後者は，Kafka クラスタ経由で到着したセンサデータを Elasticsearch にデータを格納する．蓄積されたデータは，Kibana で可視化した．

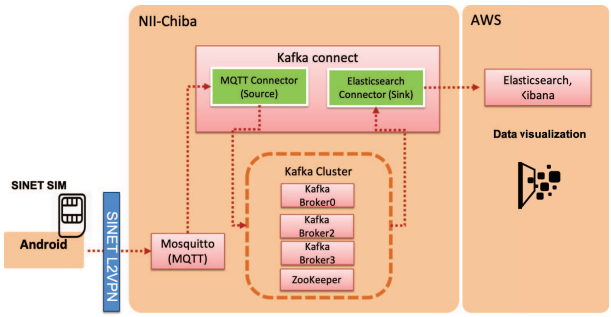

図 1: Android 端末を用いた IoT プロトタイプシステムの構成図

2.2 開発事例

Android は様々なセンサタイプをサポートしているが，本事例では，Android スマホ Pixel3 が提供している照度センサ，気圧センサ，歩数カウンタを用いて送受信を行った．図 2 に開発したアプリのスナップショットを示す．プロトタイプシステムでは，Android 端末から収集されたデータを NII 千葉にある Kafka クラスタ経由で AWS 上に構築した Elasticsearch v.7.0 にデータを格納する．また，Elasticsearch に蓄積された Android 端末のセンサデータを Kibana v7.0 で可視化した結果を図 3 に示す．利用者がトピック名及び時間を変えることで異なるセンサの可視化結果を分析できるようにした．ただし，本システムの開発では SINETStream は用いず実装しており他のメッセージングシステムへの対応にはコードの書き換えが必要である，センサーデータ取得のための実装は自前で行う必要があり容易ではないなど，課題があった．

3. SINETStream

環境測定や生体観測，IoT など広域に分散したデータを活用する研究では，広域ネットワークを介してセンサー等から取得されるデータを欠損なく確実に収集し，解析に用いることが求められてきた．しかし，データの収集や解析を行いたい研究者にとって，広域ネットワークを介してデータを収集・解析するプログラムを作成することは，ネットワークに関する高度な知識やプログラミングスキルが必要とされ，容易に開発できない．そこで，国立情報学研究所では，高度な IoT アプリケーションの開発を容易に

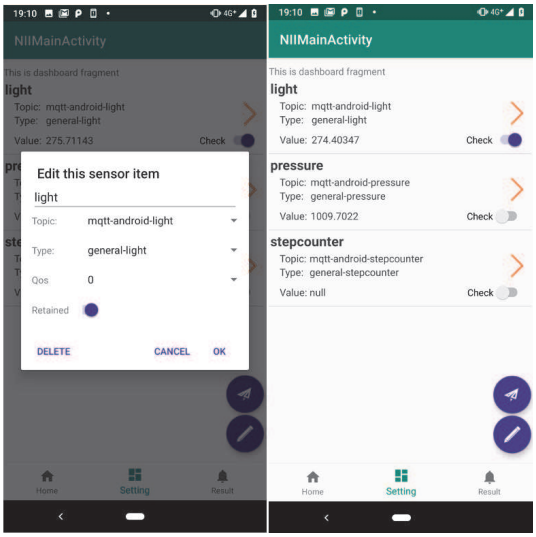

図 2: Android アプリ開発事例

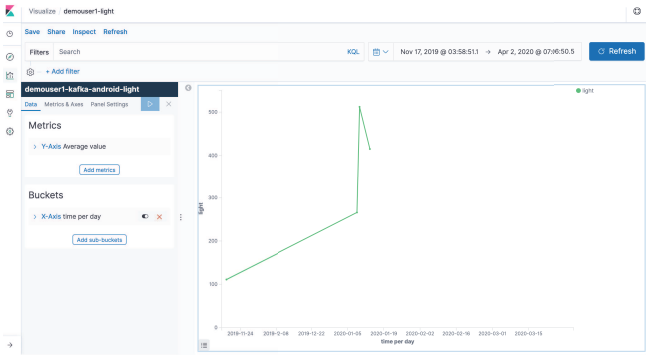

図 3: 照度センサの実行結果

するために，プログラム開発支援ソフトウェアパッケージ SINETStream を開発し，2019 年 12 月 24 日にリリースした [8]．

SINETStream では，トピックベースの Pub-Sub 型非同期メッセージングモデルを前提としている．トピックは，メッセージを送受信するときの論理的なチャネルを表している．図 4 に SINETStream の概念図を示す．SINETStream では，センサデータを送信する IoT デバイス側の Publisher プログラムを Writer，収集したデータを活用するサーバ計算機等で利用する Subscriber のプログラムを Reader と呼ぶ．Writer と Reader では，SINETStream が提供する API を実装することで図 4 の中央にあるメッセージブローカ（Kafka，MQTT ブローカ）を介してセンサデータの送受信が可能となる．また，ブローカは SINETStream に対応したものであれば任意のブローカソフトウェアまたはクラウド Pub-Sub サービスを利用することができるという特徴がある [13]．

また，SINETStream では，IoT ストリームデータ処理のためのメッセージング API（Application Programming Interface），セキュリティ機能，および SPI（Service Provider Interface）の提供を行う．ストリーム処理のための基盤ソフトウェアやクラウドサービスを抽象化するメッセージング API を提供することで，アプリケーション研究者がメッセージング基盤システムを意識することなくセンサデータの収集・解析のためのデータの書き込み・読み出しを容易に行えるようにする．また，セキュリティ機能として，ユーザおよびホストの認証・認可，データ暗号化の機能を提供するだけではなく，SPI により多様なミドルウェアやクラウドサービスを利用できるようにし，アプリケーションと計算基盤の専門家が協同して IoT アプリケーションを構築可能にする枠組みも提供している．

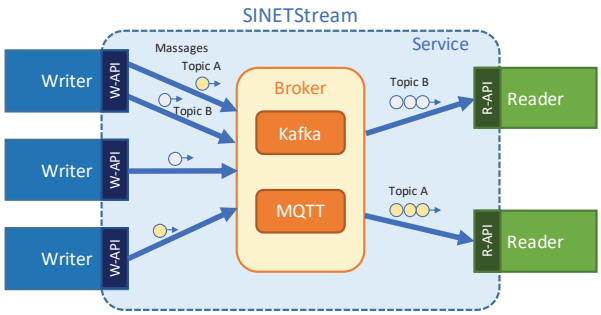

図 4: 広域データ収集基盤

SINETStream を用いた IoT アプリケーション構築事例として，図 5 のと温度・湿度モニタリングライブデモについて紹介する．本ライブデモは，SINETStream の GitHub Pages サイトで公開されている [9]．本実験では，温度・湿度センサ付き Raspberry Pi 端末から SINETStream の Writer API を介して測定した温度・湿度データを Kafka ブローカに常時書き込む．Kafka ブローカおよび Reader 用 Python プログラムはそれぞれクラウドの VM 上で実行しており，Reader プログラムは SINETStream API を介して Kafka ブローカから温度・湿度センサの実測値を取得する．Reader プログラムでは，読み込んだ実測値の 10 分ごとの平均値を計算し，実測値と平均値をそれぞれ AWS のオブジェクトストレージ S3 に書き込む．GitHub Pages では，Javascript を用いて定期的に 5 分間，及び 24 時間の温度・湿度の実測値および平均値をそれぞれグラフ化して表示している．5 右下にある 2 つのグラフのうち，上が 5 分間，下が 24 時間の測定結果を表しており，24 時間のグラフでは温度・湿度変化が明確に現れている．

4. SINETStream Android プラグイン

SINETStream は，Raspberry Pi のような Linux ベースのセンサ端末を前提として開発が進められているが，多様なセンサやバッテリが装備されているスマートフォンには対応していない．本稿では，手軽に利用できる Android 端末をモバイル IoT デバイスとして活用して多様なセンサデータ収集・解析の利便性を向上させるために，新たに SINETStream Android プラグインを開発し，Android

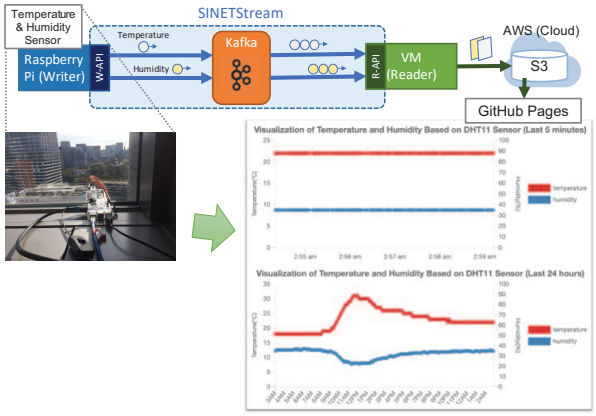

図 5: 広域データ収集基盤

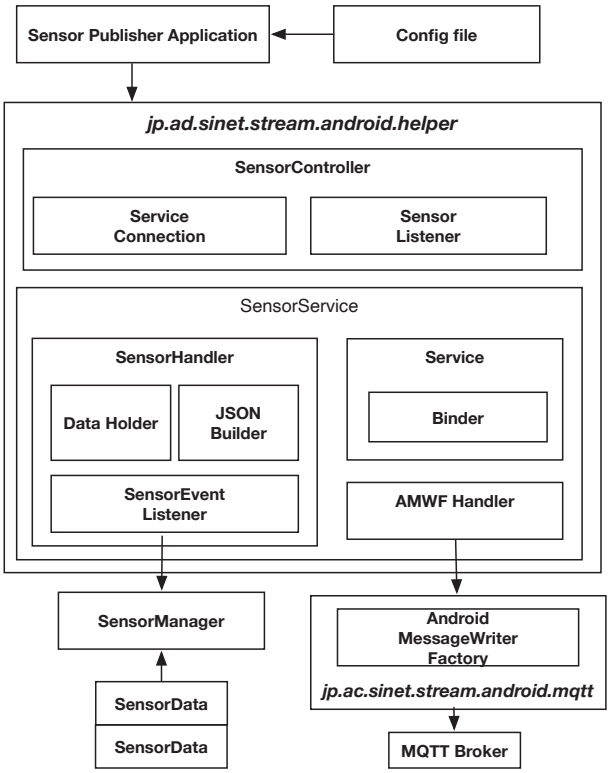

図 6: SINETStream Android プラグインの設計

端末を用いた安全な広域データ収集を支援する．

図 6 に SINETStream Android プラグインの構成を示す．SINETStrean Android プラグインは，Android 用 SINETStream コアパッケージ `jp.ac.sinet.stream.android.mqtt` と，SINETStream Android Helper パッケージ `jp.ac.sinet.stream.android.helper` で構成される．前者は，既存の Linux 向け SINETStream ライブラリを Android 向けに開発したものであり，後者は Android 端末における多様なセンサの処理を容易にするために開発した Android プラグイン固有のライブラリとなっている．

4.1 Android 用 SINETStream コアパッケージ

Android 用 SINETStream コアパッケージは，Linux 版

ソースコード 1: Android Writer API の利用例

```
1  AsyncMessageWriter<String> writer =
2      new AndroidMessageWriterFactory.Builder<
           String>()
3          .context(activity)
4          .service("service-1")
5          .build()
6          .getAsyncWriter();
7
8  writer.write("message")
9      .addCallback(
10         (msg) -> Log.d("SINETStream", "PUBLISH
               SUCCESS"),
11         (msg, ex) -> Log.d("SINETStream", "
               PUBLISH FAILURE", ex));
```

ソースコード 2: Android Reader API の利用例

```
1  AsyncMessageReader<String> reader =
2      new AndroidMessageReaderFactory.Builder<
           String>()
3          .context(activity)
4          .service("service-1")
5          .build()
6          .getAsyncReader();
7
8  reader.addCallback(
9      (msg) -> System.out.println(msg.getValue())
10 );
```

SINETStream の Java API に準拠して開発した．SINETStream では，メッセージングシステムへの書き込み時に利用する Writer API と，メッセージングシステムからの読み出し時に利用する Reader API が提供されている．ただし，Android の制約上同期 API が利用できない，Kafka の Android 向けライブラリは提供されていないため，非同期呼び出しで MQTT ブローカに対する読み書き処理を行うコアパッケージを開発した．本パッケージの実装には Eclipse Paho を用いた．Paho は，オープンソースの MQTT ライブラリであり，多くの言語向けにクライアントライブラリを提供している．MQTT プロトコルの 3.1, 3.1.1 及び 5.0 のバージョンもサポートしており，比較的簡単に Android ベースでの MQTT パブリッシャ／サブスクライバ機能を実装することができる．

ソースコード 1, 2 に非同期の Writer API および Reader API の利用例を示す．`AndroidMessageReaderFactory` は，Linux 版の `MessageReaderFactory` と同じインタフェースを提供する．callback メソッドを用いて非同期でメッセージの書き込み，読み出しを行っている．

4.2 SINETStream Android Helper パッケージ

SINETStream Android Helper パッケージは，下記の二

つのモジュールから構成されている．

- SensorController は，Android 機材のセンサ制御用の内部モジュール「センササービス」のフロントエンドとして機能する．利用者は本クラスを介してセンササービスと結合し，センサ操作に必要なメソッドを呼び出す．
- SensorService は，主に Android 端末から収集したセンサデータを加工し，加工したデータのパブリッシャーとしてバックグラウンド実行する機能モジュールである．

SensorController モジュールは SensorService の制御モジュールであり，Service Connection 機能と Sensor Listener インタフェースから構成されている．前者は SensorService との接続（Binder）の状態監視を行うものであり，後者は SINETStream Android Helper パッケージ利用者へのコールバックインタフェースを提供するものである．

SensorController では，下記の六つのメソッドを提供している．

bindSensorService: アクティビティにバインドされる SensorService との結合要求を発行する．正常終了の場合は「SensorListener の onSensorEngaged」で非同期通知される．

unbindSensorService: アクティビティにバインドされた SensorService との切断要求を発行する．正常終了の場合は「SensorListener の onSensorDisngaged」で非同期通知される．

getAvailableSensorTypes: 当該 Android 端末で利用可能なセンサ種別一覧要求を発行する．正常終了の場合は「SensorListener の onSensorTypesReceived」で非同期通知される．

enableSensors: 指定したセンサ種別群に対して利用開始要求を発行する．

disableSensors: 指定したセンサ種別群に対して利用解除要求を発行する．

setIntervalTimer: ネットワーク送出間隔を設定する．センサによっては通知される測定値は高頻度なものとなるため，これらを即時にネットワークに流すと負荷が高くなってしまう．負荷抑制のためのレート制御として送信間隔を秒単位で指定できる．

ここで，アクティビティ（Activity）は Android アプリケーションのコンポーネントであり，利用者に提供する操作画面を表すクラスを表す．

Sensor Listener コールバックインタフェースは下記の五つ API から構成されている．Android アプリケーション開発者は，本インタフェースに関する個別処理を実装する必要がある．

onSensorTypeReceived: 当該 Android 端末で利用可能なセンサ種別一覧を返す．利用者はこの中から必要なセンサを選択し，利用登録する．

onSensorEngaged: アクティビティがセンササービスと結合されたことを通知する．

onSensorDisengaged: アクティビティがセンササービスと正常に切断されたことを通知する．

onSensorDataReceived: センサ情報の送信タイミングになった時にネットワークに送出されるデータのペイロード部分と同内容のものがアクティビティ側に通知される．

onError エラー発生時にその内容が通知される．

次に，SensorService モジュールは SensorHandler，Service，AMWF Handler の三つの機能から構成されている．SensorHandler は Android のセンサ情報の取得制御モジュールで，Service は Android が提供する「サービス」機構である．また，AMWF Handler は既存の Android 用 SINETStream ライブラリから操作部分である．

さらに，今回設計した SensorHandler は下記の機能を拡張実装した．

DataHolder: ネットワーク送出レート制御のため，通知されたセンサ値を一時的に蓄積する．

JSON Builder: DataHolder に蓄積したセンサ値を集約して JSON 形式に変換する．次の節で詳しく説明する．

SensorEventListener: Android が提供している SensorManager や Sensor クラスからのセンサ値を受け取る．

4.2.1 送信するセンサデータ

SINETStream Android Helper パッケージでは，Android が提供している SensorManager から取得したセンサデータをそのまま送信するのではなく，IoT アプリケーションで必要とされる情報も付与して JSON 形式で送信する．ソースコード 3 に送信するセンサ情報の例を示す．各センサで取得したセンサデータに加え，Android 端末の情報と位置情報等を付与する．本パッケージを利用することで，Android 端末から複数のセンサ値をまとめて 1 つのメッセージとして送信することができるようになる．また，メッセージ送信時のタイムスタンプが SINETStream のコアライブラリの機能により付与される．

4.3 SINETStream Android Helper の使用例

本節では，SinetStream Android プラグインの使用方法を説明する．まず，ソースコード 4 で示すように SinetStream Android Helper の SensorListener インタフェースを実装する SampleActivity クラスを宣言する．

次に，ソースコード 5 で示すように Activity の開始時と終了時の処理を実装する．onStart および onStop メソッド内で，SensorController を介してセンササービスとの結合／解除処理を実装する．

ソースコード 3: 取得したセンサデータを加工した JSON フォーマット

```
"device":{
        "publisher":" publisher id",
        "type": "type value",
        "os version": "osversion",
        "sinetstream version": "
            sinetstreamversion",
        "location":{
               "longitude": "value longitude",
               "latitude" : "value latitude"
        }
},
"sensors":[
{
        "type" : "type valueX",
        "subtype" : "subtype valueX",
        "timestamp" : "time valueX",
        "value" : "sensor valueX"
},
        ...
{
        "type" : "type valueY",
        "subtype" : "subtype valueY",
        "timestamp" : "time valueY",
        "value" : "sensor valueY"
}
        ...
]
```

ソースコード 4: SensorListener インタフェースの実装

```
// Activity の宣言
public class SampleActivity extends
     ActivityCompat
               implements SensorListener {
    ...
    // クライアントID 及び SesnroController の宣言
    private final int mClientId = 1;
    private SesnroController mSensorController
         = null;
    ...
}
```

ソースコード 6 では，センササービスの結合／切断時の実装例を示す．利用者のアプリケーションで利用する Android 端末のセンサ一覧を取得する．そして，ソースコード 7 では取得したセンサ一覧から個別のセンサ情報を返す．また，ソースコード 8 では複数のセンサの登録／削除を行う処理の実装例を示す．取得したセンサ一覧から利用者のアプリケーションが利用するセンサを登録する．利用者は単一のセンサだけではなく，複数のセンサをまとめて利用することができる．

SINETStream Android Helper を利用する場合は，センサデータは AMWF (Android Message Writer Factory) Handler モジュールを介して自動的に送信されるため，利用者は 4.1 節で示した Writer API を実装する必要はない．

ソースコード 5: Activity の初期化と終了化

```
// Activity の初期化
@Override
protected void onStart() {
    super.onStart();
    ...
    SensorController オブジェクトの初期化
    mSensorController = new SensorController(
         this, mClientId);
    mSensorController.bindSensorService();
    ...
}
// Activity の終了化
@Override
protected void onStop() {
    ...
    // 自分自身を停止する前に
         SensorService のバインドを解除
    if (mSensorController != null) {
        mSensorController.unbindSensorService();
    }
    ...
    super.onStop();
}
```

ソースコード 6: センササービスの結合と切断

```
// センササービスと結合時
// SensorListener onSensorEngaged の実装
@Override
public void onSensorEngaged(String info) {
    ...
    // デバイスで利用可能なセンサタイプの取得
    mSensorController.getAvailableSensorTypes
         ();
}
// センササービスと切断時
// SensorListener onSensorDisengaged の実装
@Override
public void onSensorDisengaged(String info) {
    ...
}
```

5. 関連研究

Apache Kafka[1] や HiveMQ[2]，Mosquitto[3]，VerneMQ[4] 等の MQTT ブローカなど，メッセージング基盤ソフトウェアは複数開発されている．その中で，Kafka はメッセージングスループットが比較的高いものの，プロトコルオーバヘッドの点では MQTT の方が優位であり，IoT アプリケーションの要求に応じて使い分ける必要がある．そこで，SINETStream ではそのようなメッセージング基盤ソフトウェアの変更を容易に行えるようにしている．

その他に，Amazon[5] や Google[6] や Azure[7] 等の商用

ソースコード 7: センサ種別一覧

```
// センサ種別一覧取得時
// SensorListener onSensorTypeReceived の実装
@Override
public void onSensorTypesReceived(ArrayList<
     Integer> sensorTypes) {

    for (int i=0, n=sensorTypes.size(); i < n; i
         ++) {
        int sensorType = sensorTypes.get(i);
        Log.(TAG, "SENSOR[" + (i+1) + "/" + n +
             "]: " + "type(" + sensorType + ")");
    }
}
```

ソースコード 8: 複数のセンサ登録／削除処理の例

```
// センサ種別登録または削除時
public void MultipleSensorRegistration(boolean
      isEnable) {
    ArrayList<Integer> sensorTypes = new
        ArrayList<>();
    // ターゲットセンサタイプのセットアップ
    sensorTypes.add(Sensor.TYPE_ACCELEROMETER);
    sensorTypes.add(Sensor.TYPE_STEP_COUNTER);
    ...
    if (isEnable) {
        mSensorController.enableSensors(
            sensorTypes);
    } else {
        mSensorController.disableSensors(
            sensorTypes);
    }
}
```

クラウドサービスでは，IoT デバイス用の SDK の提供や IoT データ処理のためのプラットフォームを提供している．クラウドサービスのロックインの恐れがある，MQTT プロトコルのみを対象としている点で SINETStream と異なる．

IoT アプリケーションの基盤研究では，分散プラットフォームの開発も盛んに行われている．Jose Luis 氏らは軽量かつ安全なデータ送受信を実現するために，組み込み型 MQTT ブローカ Pulga を開発している [15]．SINETStream 同様 Paho を利用した実装となっているが，センサデバイス側のサポートは十分行われていない．SINETStream Android プラグインでは，複数のセンサデータの送信を支援するライブラリを提供している．

6. おわりに

本研究では，スマートフォンのような多様なセンサを IoT デバイスとして活用するアプリケーションの開発支援のために，SINETStream Android プラグインを新たに開発した．本パッケージの API を利用することで，本来の Android センサマネージャーから各種のセンサの扱いをより簡単にするとともに，単一のセンサから複数のセンサをまとめて MQTT ブローカー経由で受送信ができるようになった．また，収集したセンサデータを分析や解析を便利するために，センサデータを共通化した JSON 形式も定義した．

今後は，開発した SINETStream Android プラグインの性能評価をするとともに安全にデータ収集できるように SSL/TLS 対応も取り込む予定である．また，第五世代移動通信システム（5G）の実用化により，これまで収集が難しかった大容量のストリームデータを扱うアプリケーションの構築も期待されるため，5G への対応と SINETStream Android プラグインの有効性に関する評価も行う予定である．

謝辞　本研究を進めるにあたり，実験環境構築および評価実験の実施にご協力いただいた数理技研の遠藤雅彦様，小泉敦延様，鯉江英隆様に深く感謝いたします．

参考文献

[1] Kafka，”https://kafka.apache.org/”.
[2] HiveMQ，”https://www.hivemq.com/”.
[3] Eclipse Mosquitto，”https://mosquitto.org/”.
[4] VerneMQ，”https://vernemq.com/”.
[5] Amazon IoT，”https://aws.amazon.com/iot/”.
[6] Google IoT，”https://cloud.google.com/solutions/iot”.
[7] Azure IoT，”https://azure.microsoft.com/en-us/overview/iot/”.
[8] SINETStream，”https://nii-gakunin-cloud.github.io/sinetstream/”.
[9] SINETStream Live Monitoring Demo，”https://nii-gakunin-cloud.github.io/sinetstream/docs/livedemo/livedemo.html”.
[10] MQTT コネクタ，”https://docs.lenses.io/connectors/source/mqtt.html”.
[11] Elasticsearch コネクタ，”https://docs.confluent.io/current/connect/kafka-connect-elasticsearch/index.html”.
[12] Light, Roger. ”Mosquitto: server and client implementation of the MQTT protocol.” Journal of Open Source Software 2.13 (2017): 265.
[13] 竹房 あつ子，孫 静涛，藤原 一毅，吉田 浩，合田憲人:IoT ストリームデータ処理のためのソフトウェアライブラリ SINETStream の開発, 情報処理学会研究報告 2020-IOT-48(19), pp. 1 - 8, 2020 年 3 月.
[14] 孫 静涛，藤原 一毅，竹房 あつ子，長久 勝，吉田 浩，合田憲人:SINET 広域データ収集基盤のための基盤ソフトウェアの検討, 情報処理学会研究報告 2019-OS-147(11), pp. 1 - 9, 2019 年 7 月.
[15] Jose Luis Espinosa-Aranda, et al. ”Pulga, a tiny open-source MQTT broker for flexible and secure IoT deployments.” in 2015 IEEE Conference on Communications and Network Security (CNS) workshop, pp. 690-694. IEEE, 2015.