Crowdsourcing with Android

3SN Parcours SEmbIIoT

Katia Jaffrès-Runser, Quentin Bailleul

2021-2022

Brève description L'objectif principal de ce module est de vous donner les outils pour construire une application Android capable de mesurer à intervalles réguliers l'activité de leur utilisateur ou de son environnement, via les différents capteurs et sondes offertes part Android. Ces bases vous permettront de développer différentes applications très utiles actuellement dans différents contextes :

- *Device monitoring* : mesure de l'utilisation des resources du smartphone pour en connaitre les performances au cours du temps (monitoring réseau, monitoring système, ..)
- *Quantified self* : mesure de l'activité de l'utilisateur pour qu'il puisse mesurer son comportement
- *Crowdsourcing* : mesure de l'activité ou de données qui sont poussées vers un serveur de façon à améliorer la compréhension d'un phénomène, contribuer à une tâche.

Déroulement Ce module se compose de 5 séances décomposées en 4 séances de travail et une séance de recette. Les séances sont assez espacées et le support très guidé pour vous permettre d'avancer entre les séances de travail. La suite de ce document vous fournira des exercices permettant de :

- Créer l'architecture de votre application,
- Enregistrer les données mesurées (en XML ou dans une base MySQL),
- Visualiser ces données sur le smartphone.

Quand votre apprentissage sera terminé, nous vous demanderons de réaliser une application originale de monitoring et de nous la présenter lors de la dernière séance du module. Ce développement se fait en monôme.

Ressources Des information pratiques relatives au module (consignes supplémentaires, cours Android 2TR, etc.) sont postées sur la page Moodle.
Le site principal pour les développeurs d'applications Android :
`http://developer.android.com`.

Tests Attention, pour tester votre application, il faudra la déployer sur le smartphone fournis. Pour que vous puissiez le faire, il faut que le système Android du smartphone soit en mode *debug*. Pour cela, aller dans les paramètres, chercher le lien vers les paramètres du système (en bas) et taper 7 fois sur la description du 'numéro de build' (en bas). Il s'affichera un message pour vous dire que vous êtes maintenant un développeur Android ;-) Ensuite, revenir en arrière, et vous aurez un nouveau lien options développeurs. Activer le mode Déboggage USB.

Evaluation La dernière séance sera dédiée à l'évaluation de votre application de monitoring. La note se décompose comme suit :

[7pts] Une recette qui aura lieue lors de la dernière séance de TP. On demandera une explication du travail réalisé qui pourra être supporté par une courte présentation (2-3 slides max, pas obligatoire). A la suite de cette explication, il faudra réaliser une démo du travail effectué. Il est demandé de présenter une version du projet *qui marche*. Le passage se fera par ordre alphabétique (nom, prénom).

[5pts] Le code commenté, rendu le jour même de l'évaluation. A la fin de votre évaluation, vous déposerez une archive avec votre projet AndroidStudio en utilisant la plateforme de dépôt de fichier suivante : `https://filesender.renater.fr/` Cette plateforme vous fournit une URL à indiquer sur Moodle.

[4pts] Un rapport d'au maximum 3 pages qui décrit le travail réalisé, l'architecture de l'application (les activités et classes implantées) et le fonctionnement de l'application (copie d'écran des mesures réalisées). La date de remise du rapport vous sera communiqué par l'intervenant.

[4pts] Assiduité / implication.

Votre application pourra mesurer un donnée de votre choix, mais devra comporter les éléments suivants :

— Le monitoring régulier avec la classe `JobService` et le `ForegroundService` comme présenté à la Figure 2 (cf. partie 1, p. 6),

— L'enregistrement des données qui sera réalisé soit via un fichier XML, soit via une base de données (cf. partie 2, p. 7),

— Un affichage des données mesurées soit avec un `ListView` + `ArrayAdapter` (cf. section 2.1, p. 8), soit sous la forme d'un graphique (cf. section 3, p. 11).

Architecture de l'application

Les applications de monitoring mesurent, régulièrement, des données. Par exemple, un moniteur de réseau sans-fil peut enregistrer, à intervalle de temps régulier, la présence de points d'accès WiFi, ou encore la presence de réseaux Bluetooth. Cette mesure est alors enregistrée dans la mémoire du téléphone.

Les données enregistrées sont traitées à la demande de l'utilisateur de l'application quand celle-ci est activée. On pourra par exemple afficher le lieu où les différents points d'accès WiFi on été vus sur une carte. Ou encore afficher le temps passé par un utilisateur à manipuler son smartphone dans une journée.

Dans les premiers sujets de TP, nous vous proposons de créer une première application qui va mesurer et enregistrer le SSID(ou l'adresse MAC) des réseaux WiFi environnants à intervalle régulier. Nous vous proposons de suivre l'architecture présentée à la figure 1.

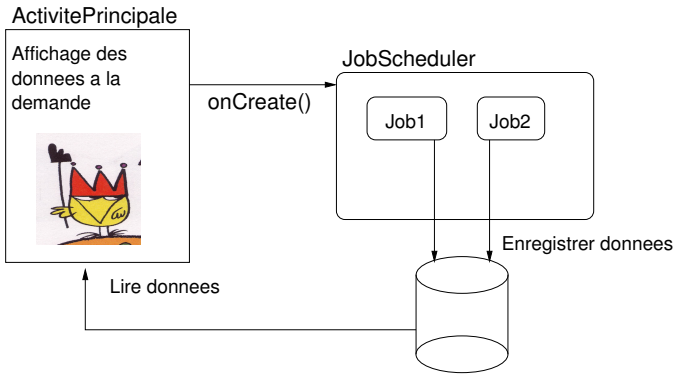

FIGURE 1 – Architecture de l'application de monitoring

Lancement des tâches de mesure L'activité principale de votre application Android va, lors de sa création, lancer différentes tâches de mesure que l'on appellera `Job`. Chaque tâche de mesure est définie dans sa propre classe et regroupe les mesures qui sont faites en même temps ou obtenues par un même service de l'OS Android. Ces tâches, si elles sont lancées périodiquement, sont activées même si l'activité principale n'est pas visible. On dit qu'elles sont en tâche de fond.

Les jobs sont lancés par un objet de classe `JobScheduler`. Cet objet, en interaction avec l'OS Android, décide de la date de lancement des jobs qu'il gère. Il est disponible depuis l'API 21 (i.e. Android 5.0 Lollipop) et a pour avantage de prendre en compte la consommation énergétique du système dans le choix des dates d'appels (les applications de monitoring ont le désavantage de consommer beaucoup d'énergie en général). Les jobs peuvent être appelés périodiquement par le `JobScheduler` ou juste une fois. La partie 1 a pour objectif de mettre en place ce mécanisme d'appel périodique des `Job`.

Il est conseillé de créer une application qui utilise une API 21 ou 22 pour réaliser aisément les tâches de ce projet. Une API plus récente va vous rajouter des contraintes liées à la demande de permissions (notamment pour l'accès au scan des points d'accès WiFi).

Enregistrement des données Une fois la mesure collectée, elle est enregistrée. Ces données peuvent être soit enregistrées dans des fichiers soit dans une base de données. Dans la partie 2, nous verrons en premier comment écrire et lire des données dans un fichier au format XML. Ensuite, nous verrons comment les enregistrer dans une base de données MySql proposée par Android en natif.

Traitement et affichage des données L'activité principale de votre application ira lire les données enregistrées pour les traiter quand on en aura besoin. On peut avoir besoin de le faire quand l'utilisateur veut les

consulter, ou quand certaines données sont arrivées (génération d'une notification), etc. La partie 3 présente quelques outils d'affichage des résultats (si vous trouvez mieux, je ne vous en voudrais pas !).

1 Partie 1 : Mise en place du monitoring régulier.

Cet exercice a pour objectif de générer une première application qui se compose d'une activité principale (`MainActivity.java`) et d'une classe par Job à lancer qui hérite de `JobService`. Dans notre example, on définit une classe `WiFiJobService` qui va mesurer la liste des points d'accès disponibles (SSID ou adresse MAC).

La classe `WiFiJobService` : La classe `WiFiJobService` doit surcharger les méthodes `onStartJob` et `onStopJob`. C'est dans la première que l'on définit the traitement à effectuer. Attention, le Job est lancé dans le même processus que l'application. Pour éviter le blocage de l'activité principale, il faut lancer le traitement dans un processus léger (`Thread`, `Runnable` ou `AsyncTask`). Nous vous conseillons d'utiliser `AsyncTask`.

```
        // Méthode appelée quand la tâche est lancée
        @Override
        public boolean onStartJob(JobParameters params) {
            Log.d(TAG, "onStartJob id=" + params.getJobId());
           // ***** Lancer ici la mesure dans un thread à part *****
            return true;
        }

       // Méthode appelée quand la tâche est arrêtée par le scheduler
       // Retourne vrai si le scheduler doit relancer la tâche
        @Override
        public boolean onStopJob(JobParameters params) {
            Log.d(TAG, "onStopJob id=" + params.getJobId());
            // ***** Arrêter le thread du job ici  ******
            return shouldReschedule;
        }
```

Pour notifier au scheduler que le Job s'est terminé, il faut utiliser la méthode `void jobFinished (JobParameters params, boolean needsReschedule));` à la fin du traitement. Si le second paramètre vaut `true`, le scheduler pourra relancer le job s'il n'a pu se terminer lors de l'appel courant (méthode onStopJob() a été appelée).

Ne pas oublier de déclarer `WiFIJobService` dans le manifest avec la permission `android.permission.BIND_JOB_SERVICE` :

```
    <service
          android:name=".WiFiJobService"
          android:exported="true"
          android:permission="android.permission.BIND_JOB_SERVICE" />
```

Nous vous invitons à consulter la page suivante `https://developer.android.com/reference/android/app/job/JobService.html` pour de plus amples informations sur les JobService.

Lancement de `WiFiJobService` **par** `MainActivity` **:** Dans l'activité principale, il faudra lancer le `WiFiJobService` à l'aide du `JobScheduler`. Pour cela, il faut utiliser la méthode `Builder` de la classe `JobInfo`. L'object `JobInfo` contient tous les paramètres de la tâche que doit prendre le scheduler en compte. Par exemple, `setOverrideDeadline(100)` permet de donner une deadline de 100ms au scheduler pour lancer le Job. Ou encore, `setPeriodic(1000)` permet de lancer une tâche de façon périodique toutes les secondes.

```
    private void scheduleJobWiFi() {
        ComponentName serviceName = new
                ComponentName(this, WiFiJobService.class);
        JobInfo jobInfo = new JobInfo.Builder(JOB_ID, serviceName)
                .setPeriodic(1000)
                .build();

        JobScheduler scheduler = (JobScheduler)
                 getSystemService(Context.JOB_SCHEDULER_SERVICE);
        int result = scheduler.schedule(jobInfo);
        if (result == JobScheduler.RESULT_SUCCESS) {
                Toast.makeText(this, "schedule␣job␣avec␣success",
                    Toast.LENGTH_LONG).show();
        }
    }
```

A partir de ces informations, réaliser une application qui mesure et affiche dans les logs le nom des points d'accès à intervalle régulier. Pour obtenir cette information, il faut faire appel à la méthode `getScanResults()` du `WiFiManager` comme présenté ici :

```
WifiManager wifiMan = (WifiManager) getApplicationContext(
                .getSystemService(Context.WIFI_SERVICE);
List<ScanResult> scanResults = wifiMan.getScanResults();
```

Attention, pour les versions de SDK > 23, il faut demander à l'utilisateur l'autorisation d'utiliser la position ou les données WiFi si on veut les utiliser dans une tâche de fond. Je vous recommande d'utiliser la version 22 du SDK.

`https://developer.android.com/training/location/background`

Rendre la mesure persistante : Avec l'application actuelle, les jobs s'arrêtent si `MainActivity` est stoppée sur le téléphone. Nous voulons faire en sorte que la mesure soit perpétuelle et qu'Android n'arrête pas l'application si les ressources en mémoire ou énergie deviennent trop faibles.

Pour se faire, il faut que les jobs soient lancés par un `service` de premier plan (foreground service). Pour cela, on fait croire à Android que le service a la même priorité qu'une application que consulte un utilisateur (l'application au premier plan).

On va donc introduire dans notre architecture un service intermédiaire entre `MainActivity` et le scheduler. Les Jobs ne seront plus lancés par `MainActivity`, mais dans la méthode `onCreate()` du service `ForegroundService`. Ce changement est illustré à la Figure 2

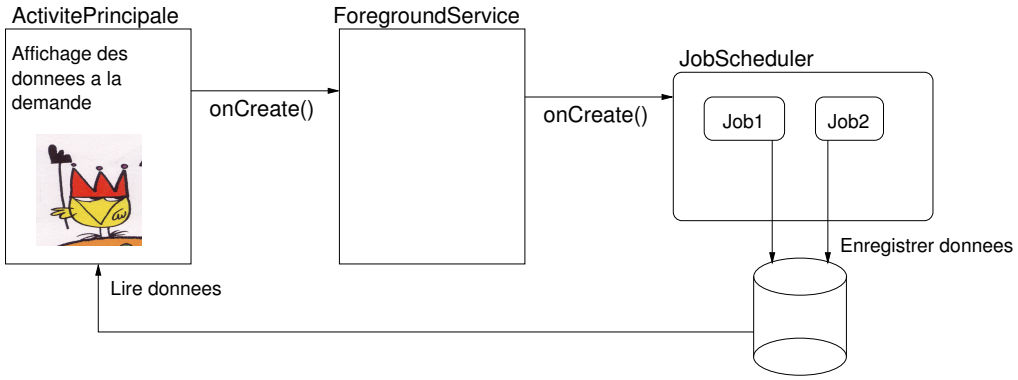

FIGURE 2 – Architecture de l'application de monitoring avec un service premier-plan

Ajouter le service à votre projet en l'appelant `ForegroundService`. Pour que ce service soit lancé au premier plan, il faut lui ajouter une notification (qui apparaitra dans la bare de notifications) à sa création dans `onCreate()` avec le code suivant :

```
@Override
public void onCreate() {
     super.onCreate();

     //Put this Service in foreground
     // -> will not be killed by the system if low on memory
    Intent notificationIntent = new
        Intent(MainActivity.context, MainActivity.class);
    PendingIntent contentIntent =
        PendingIntent.getActivity(this, 0, notificationIntent, 0);

    Notification notif = new Notification.Builder(this)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("NetMonitor is running")
        .setContentText("Click to open app")
        .setUsesChronometer(true)
        .setContentIntent(contentIntent)
        .setOngoing(true)
        .build();

    //On place le service en premier plan
    startForeground(1337, notif);

    // Lancer ici le job de monitoring
```

Dans la même méthode `onCreate` de `MainActivity`, on lance le service avec :

```
//Start foreground service that will schedule the various Jobs.
startService(new Intent(context, ForegroundService.class));
```

Vous avez maintenant une application qui insère un notification permanente dans la barre supérieure du téléphone. Cette notification reste même si l'application est tuée.

2 Partie 2 : Enregistrer les données.

Il est possible d'enregistrer les données lues par les jobs dans un fichier ou dans une base de données MySql offerte nativement par Android. Nous allons présenter les deux. A vous de choisir celle qui vous convient le mieux !

2.1 Fichier XML

Enregistrer dans un fichier XML Cette partie créé un fichier au format XML et utilise un parser XML pour décoder les données. On souhaite générer un fichier XML avec les données WiFi lues dans le job. Ces données sont alors enregistrées dans un fichier interne visible uniquement par notre application. Voici un format XML possible qui enregistre le résultat de deux scans WiFi effectués aux dates 1481231262 et 1481235892 :

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<root>
<measurement>
  <timestamp>1481231262</timestamp>
  <BSSID>
    <record>00:24:d4:69:22:30</record>
    <record>ec:b1:d7:e0:7d:04</record>
  </BSSID>
</measurement>
<measurement>
  <timestamp>1481235892</timestamp>
  <BSSID>
    <record>ec:b1:d7:e0:7d:04</record>
  </BSSID>
</measurement>
</root>
```

Les dates de mesure sont obtenues avec le code :

```
Long tsLong = System.currentTimeMillis()/1000;
String timestamp = tsLong.toString();
```

Elle représente le nombre de secondes qui se sont écoulées depuis le 1er janvier 1970 (epoch Linux). Ce fichier enregistre chaque mesure entre les balises `<measurement>`, et y ajoute la date de mesure et la liste des adresses MAC des APs WiFi observés.

Attention ! Toutes les balises doivent être entourées par une balise unique (`<root>` et `</root>` dans l'exemple ci-dessus).

Pour générer le fichier XML, on utilise la bibliothèque `android.util.Xml` (`https://developer.android.com/reference/org/xmlpull/v1/XmlSerializer.html`). Le code permettant la création de l'en-tête du fichier est le suivant (il utilise la classe `XmlSerializer`) :

```
FileOutputStream fos = openFileOutput(filename, Context.MODE_APPEND);
XmlSerializer serializer = Xml.newSerializer();
serializer.setOutput(fos, "UTF-8"); //definit le fichier de sortie
serializer.startDocument(null, Boolean.valueOf(true));
serializer.setFeature(
    "http://xmlpull.org/v1/doc/features.html#indent-output", true);
serializer.flush();
fos.close();
```

Pour pouvoir générer un balise ouvrante (resp. fermante), on peut utiliser la méthode `startTag (String namespace, String name)` (resp. `endTag(String namespace, String name)` de l'objet `serializer`. On utilisera un namespace `null`. Pour écrire la donnée entre deux tags, on utilise `text(String text)`. Finalement, pour écrire l'objet sérialisé qui contient toutes les balises dans le fichier ouvert, on utilise `serializer.flush()`.

Lire un fichier XML On souhaite lire et afficher le contenu du fichier XML dans `MainActivity` à chaque fois que l'on clique sur le bouton. Pour cela, rajouter un `ListView` dans `activity_main.xml`. Pour pouvoir facilement ajouter les données lues dans cet objet, on va lui associer un `ArrayAdapter` et un `ArrayList` dans `MainActivity` :

```
ArrayList wordList = new ArrayList<String>();
ArrayAdapter adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1,
                wordList);
ListView lv = (ListView) findViewById(R.id.listview1);
lv.setAdapter(adapter);
```

A présent, en ajoutant des éléments à l'objet `adapter`, Android mettra tout de suite à jour le ListView correspondant. En d'autres termes, l'appel à `adapter.add("texte")` ajoutera le texte dans l'objet `wordList` et l'affichera dans l'objet graphique `lv` de classe `ListView`.

Reste à lire le fichier, parser le XML et l'afficher dans l'adapter pour visualiser les données dans l'activité principale.

Voici un exemple de code permettant de lire les données brutes d'un fichier :

```
private String readRawData(String filename, String data) {
    FileInputStream fis = null;
    InputStreamReader isr = null;
    try {
        fis = openFileInput(filename);
        isr = new InputStreamReader(fis);
        char[] inputBuffer = new char[fis.available()];
        isr.read(inputBuffer);
        data = new String(inputBuffer);
        //Log.i(TAG, "Read data from file " + filename);
        isr.close();
        fis.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return data;
}
```

Voici un exemple de code capable de parser les données lues et de les afficher

```
//Read data string from file
data = readRawData(filename, data);

InputStream is = null;
try {
    is = new ByteArrayInputStream(data.getBytes("UTF-8"));
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}

DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();;
DocumentBuilder db;
NodeList items = null;
Document dom;

try {
    db = dbf.newDocumentBuilder();
    dom = db.parse(is);
    dom.getDocumentElement().normalize();

    //get all measurement tags
    items = dom.getElementsByTagName("measurement");
    Log.d(TAG, "nb␣of␣xml␣measurements␣=␣"+items.getLength());
    for (int i=0;i<items.getLength();i++){
        Element measure = (Element)items.item(i);
        //get timestamp
        String timestamp = measure.getElementsByTagName("timestamp").
            item(0).getTextContent();
        //for all elements in the document
        Log.d(TAG,"Measurement␣"+i+ "␣with␣timestamp␣"+timestamp);
        //get all APs
        NodeList aps= measure.getElementsByTagName("record");
        for (int j=0; j<aps.getLength();j++){
            String ap = aps.item(j).getTextContent();
            Log.d(TAG, "␣␣"+ap);
        }
    }
} catch (ParserConfigurationException e) {
    e.printStackTrace();
} catch (SAXException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

2.2 Base de données SqlLite

Il existe un tutorial pour apprendre à manipuler une base de données sous Android :

`https://developer.android.com/training/data-storage/sqlite`

Il est très bien fait et vous donnera toutes les informations pour enregistrer les données dans une base MySQL. Il faut pour cela définir les tables et leurs champs dans une classe dédiée (classe DBContract). Puis, définir les requêtes qui permettent de créer les tables ou les supprimer.

Pour pouvoir faire une requête SQL simple sur la base de donnée, on peut aussi utiliser la méthode :

```
Cursor c = db.rawQuery("SELECT * FROM WIFI",null);
Log.d(TAG, "there are "+c.getCount()+" in result query ");
```

Le curseur `c` comporte les `c.getCount()` résultats liés à la requête. Pour obtenir le premier résultat, on utilise `c.moveToFirst()` et pour passer au résultat suivant `c.moveToNext()`. On peut accéder, pour chaque résultat, à la colonne de son choix avec :

```
String APs = c.getString(
            c.getColumnIndexOrThrow(DBContract.Wifi.MAC_ADDR)
);
```

Dans cet exemple, on accède à la colonne `MAC_ADDR` de la table `Wifi`.

Note : Il existe une couche d'abstraction à la bibliothèque SqlLite qui se nomme Room que vous pouvez tester (je ne l'ai pas fait) :

`https://developer.android.com/training/data-storage/room`

3 Partie 3 : Afficher les données

Pour les tests de la partie précédente, nous avons affiché les données brutes dans un `ArrayList`, au moyen d'un `ArrayAdapter` (cf. 8) Cette affichage permet de présenter les données brutes enregistrées.

Dans cette partie, nous vous présentons une bibliothèque qui permet d'afficher les données collectées sous la forme d'un graphique. En guise d'exemple, nous allons afficher, pour chaque date de mesure, le nombre de points d'accès mesurés. La bibliothèque que nous vous proposons d'utiliser est `androidplot`. Un tutoriel existe ici : `https://github.com/halfhp/androidplot/blob/master/docs/quickstart.md`

3.1 Installation de la bibliothèque

Pour pouvoir utiliser cette librairie, il faut ajouter la dépendance suivante dans le fichier `build.gradle (Module: app)` qui se trouve dans le répertoire `Gradle scripts` :

```
 dependencies {
    compile "com.androidplot:androidplot-core:1.4.0"
}
```

Gradle installera automatiquement la librairie. Il faut aussi rajouter la ligne suivante dans le fichier `proguard-rules.pro` :

```
-keep class com.androidplot.** { *; }
```

3.2 Création de l'activité qui affiche un graphique XY

Ajouter une nouvelle activité vide dénommée `PlotActivity`. Ajouter un bouton sur MainActivity qui permet de lancer `PlotActivity`. Dans le layout XML qui correspond à votre nouvelle activité, insérer un objet graphique XYPlot :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:ap="http://schemas.android.com/apk/res-auto"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <com.androidplot.xy.XYPlot
        style="@style/APDefacto.Dark"
        android:id="@+id/plot"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        ap:title="Points␣d'accès␣WiFi␣observés"
        ap:rangeTitle="Nombre␣d'APs"
        ap:backgroundColor="@color/ap_gray"
        ap:domainTitle="Dates"
        ap:lineLabels="left|bottom"
        ap:lineLabelRotationBottom="-45"/>
</LinearLayout>
```

Pour lire et afficher les données, on peut ajouter dans la méthode `onCreate()` le code suivant (attention, ici il existe deux tables dans la base de données, `WIFI` et `Measurements`, qui possèdent les colonnes `MAC_ADDR` et `Measurement_ID` pour la première, et `Timestamp` pour la seconde).

```
// initialiser la référence sur  XYPlot :
XYPlot plot = (XYPlot) findViewById(R.id.plot);

// lire les données dans la base
SQLiteDatabase db = mDbHelper.getReadableDatabase();
Cursor c = db.rawQuery("SELECT␣Timestamp,␣COUNT(MAC_ADDR)"
   +"FROM␣WIFI,␣Measurements"
   + "WHERE␣Measurements._ID=WIFI.Measurement_ID␣"
   + "GROUP␣BY␣Measurement_ID",null);
Log.d(TAG, "Il␣y␣a␣"+c.getCount()+"␣lignes␣dans␣le␣résultat.");

if (c.getCount()>0) {
    Integer[] ids = new Integer[c.getCount()];
    Integer[] times = new Integer[c.getCount()];
    Integer[] nbAps = new Integer[c.getCount()];
    c.moveToFirst();
    for (int i = 0; i < c.getCount(); i++) {
        //Lire les colonnes de chaque ligne
        ids[i]=i; //measurement ids
        times[i]= c.getInt(0);     //colonne 0 : Timestamp
        nbAps[i] = c.getInt(1);  //colonne 1 : COUNT(MAC_ADDR)
        Log.d(TAG, "Measurement␣id␣"+i+"-␣timestamp␣"+times[i]
            +"␣-␣nb␣APS␣"+nbAps[i]);
        c.moveToNext();
    }

    // transformer les données en series XY
    XYSeries apsXY = new SimpleXYSeries(Arrays.asList(nbAps),
      SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, "Nb␣Aps");

    // définir le format de la courbe (ligne rouge, marqueurs bleus)
    LineAndPointFormatter series1Format = new
      LineAndPointFormatter(Color.RED, Color.BLUE, null, null);

    // ajouter la serie XY au plot:
    plot.addSeries(apsXY, series1Format);


} else {
    Log.d(TAG, "No␣item␣in␣result␣query␣");
}
```

Voici une capture d'écran des données affichées par l'activité `PlotActivity`.

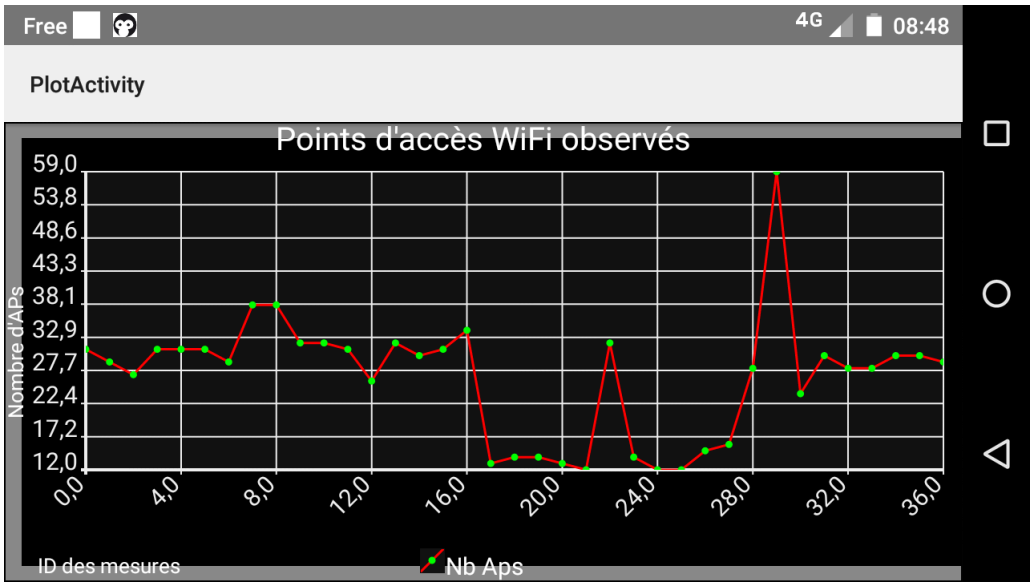

FIGURE 3 – `PlotActivity` : évolution du nombre d'APs mesurés pour chaque mesure effectuée.

et maintenant, y'a plus qu'à ...