I-Advanced Internet of Things Networking Lab

6000FTIIOT

Report: RF Survey Toolkit

Author:
Mats De Meyer

Student Number:
20129544

January 31, 2018

Contents

List of Figures

1 Introduction

The goal of this project is to design a BLE enabled IoT RF Survey Toolkit. This allows areas to be tested with certain LPWAN technologies, with feedback on a smartphone. This project will combine knowledge gained from

2 Analysis of the problem

A major concern in wireless networks is the coverage and link quality between devices. While LPWAN technologies aim to improve this, they can still struggle with large distances and busy metropolitan areas. With RF signal propagation being such a complicated issue, it is nearly impossible to determine if a certain technology will properly work when trying to calculate or simulate this without on site testing.

3 Design choices & specifications

The goal of this project is to make a device which is able to test the wireless coverage and link quality of several wireless networks. This will be done by making an RF survey toolkit. This toolkit consists of two major components, namely an LPWAN enabled embedded part and a BLE connected smartphone.

The toolkit has LPWAN connectivity, and will periodically send messages to query available gateways. The user has to move this device in the environment that is to be tested. After this testing, this data can be saved to a .json file on the smartphone. The initial sketch of this project is shown in Figure 1.

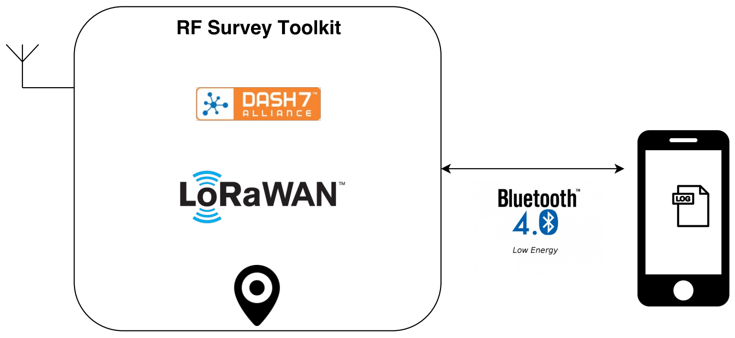


Figure 1: Initial architecture sketch.

3.1 Hardware & Architecture

The hardware used for this project is an NRF52 development kit combined with a B-L072Z-LRWAN1 discovery board used for wireless communication. An android application will be written in Android Studio, which will parse and store the data of the gateways. The main code will run on the android application, with the NRF board transparently forwarding UART and BLE data and the LRWAN board sending the LPWAN gateway queries. The choice of running the core logic as well as generating the ALP queries and parsing the responses on the android application is made to improve flexibility and the ease of programming in java instead of embedded c. Furthermore, new additions or changes can be more easily made and deployed on a smartphone instead of the embedded boards. It also allows other hardware to be used with the same android application instead of having to port the embedded code from board to board.

The final architecture can be seen in Figure 2

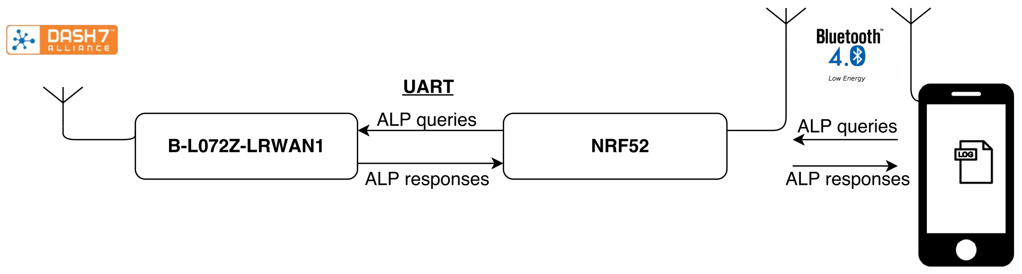


Figure 2: RF Survey Toolkit Architecture.

3.2 Embedded Project

The embedded project behaves as a link between the BLE connectivity of the smartphone and the LPWAN connectivity of the LRWAN discovery board. The LRWAN disco board is flashed as a modem, and will accept ALP commands as an input over UART. Currently, only dash7 is used but LoRaWAN support will be available in the future. The responses to these ALP commands are transmitted back over this UART link. The BLE discovery board will forward the gateway query ALP commands received over bluetooth to the UART input of said LRWAN discovery board. The corresponding responses are forwarded as raw bytes over the BLE connection to the smartphone application, where they are parsed and processed

3.3 Android Application

The android application is where the core of the toolkit is running. The survey is controlled via the user, and the commands are generated and parsed in the application. After the survey, the user is given the choice of discarding the results or storing these to a JSON file.

4 Planning

The timing schedule is slightly altered from the project proposal, due to the first talk about the approach of the project. Initially, the main code would be situated at the embedded microcontroller. However, shifting this code to the android application and only using the embedded hardware to forward ALP responses over BLE shifts a large part of the workload to more high level java programming instead of embedded c code.

Week	**Task**
Week 7	Write project proposal
Week 8	Get familiar with hardware & nrf projects, prepare presentation
Week 9	Start testing wireless technologies
Week 10	Create an android application, capable generating ALP commands
Week 11	Add ALP parsing and periodically sending to application
Week 12	Optimize the android application
Christmas Break	Write the final report

Table 1: Proposed timing schedule.

5 Obtained results

As mentioned before, the end result consists of two main code projects. One project is the embedded BLE to UART and vice versa project on the NRF52 development board, and the other is the android application containing the main survey logic.

5.1 Embedded code

This project started from example code, provided by Kwinten Schram, which forwards received data from the BLE UART service over the UART of the NRF52 board.

5.1.1 BLE UART service

Bluetooth Low Energy transfers data using Services and Characteristics. It makes use of the generic data protocol called the Attribute Protocol (ATT). These services and characteristics are stored in a simple lookup table using 16-bit IDs for each entry in the table. This implementation uses a UART service, over which data is transfered to and from the android application.

5.1.2 Forwarding BLE data to UART

When data is sent from the android application over said UART service, an interrupt is called. In its callback, `nus_data_handler2()`, the data is sent byte per byte over the UART connection of the NRF52 development board. This data is received by the LRWAN modem, which will interpret this data and parse the ALP commands. Furthermore, a LED indicator is turned on when the survey is running.

5.1.3 Sending UART data over BLE & fifo buffer

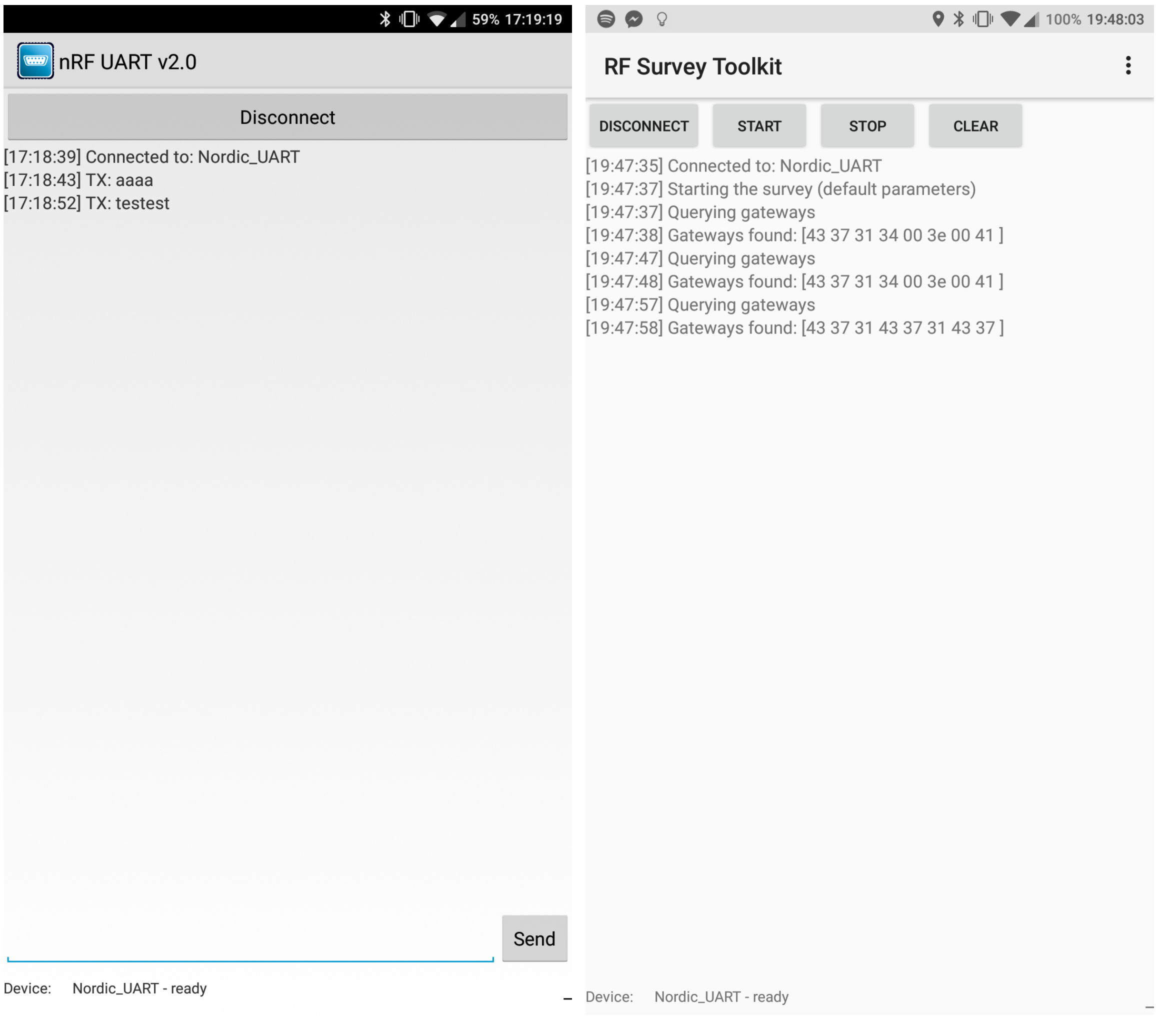
Previous forwarding only works in one way, a method of sending received UART data over BLE to the android application has to be made. When UART data is received, an interrupt is triggered once again. The callback (`uart_event_handle()`) is found in the `nrf52SDKWrap.c` file. Here the received byte will be simply stored in a fifo buffer. This minimizes the time spent in the interrupt callback, and limits the possibility of data being lost or interrupts being missed.

```c
/**@brief  Function for handling app_uart events.
 *
 */
/**@snippet [Handling the data received over UART] */
void uart_event_handle(app_uart_evt_t * p_event)
{
        switch (p_event->evt_type)
        {
                case APP_UART_DATA_READY:
                        scanf("%c",&uartByte);
                        fifo_put_byte(&uartfifo, uartByte);

                        break;

                case APP_UART_COMMUNICATION_ERROR:
                        APP_ERROR_HANDLER(p_event->data.error_communication);
                        break;

                case APP_UART_FIFO_ERROR:
                        APP_ERROR_HANDLER(p_event->data.error_code);
                        break;

                default:
                        break;
        }

}
/**@snippet [Handling the data received over UART] */
```

In the main loop, the size of the fifo buffer is checked. If it contains data, this data is sent over BLE. There is a small delay after sending the BLE data, otherwise the NRF code would crash.

5.2 Android Application

Nordic Semiconductors, the creators of the NRF platform, provide a lot of documentation and open source projects. These projects include an android application, capable of sending and receiving ASCII and UTF-8 text strings. This is done via keyboard input from the user, with feedback on a terminal. This provides a great starting point for the application required for this project. Development is done in Android Studio, debugging on a OnePlus X running an android 8.0 ROM.



```
[17:18:39] Connected to: Nordic_UART
[17:18:43] TX: aaaa
[17:18:52] TX: testest
```

```
[19:47:35] Connected to: Nordic_UART
[19:47:37] Starting the survey (default parameters)
[19:47:37] Querying gateways
[19:47:38] Gateways found: [43 37 31 34 00 3e 00 41 ]
[19:47:47] Querying gateways
[19:47:48] Gateways found: [43 37 31 34 00 3e 00 41 ]
[19:47:57] Querying gateways
[19:47:58] Gateways found: [43 37 31 43 37 31 43 37 ]
```

(a) Original NRF-UART application. (b) Updated RF Survey Toolkit application.

Figure 3: Comparison of original and new application.

5.2.1 BLE UART service

The same UART service as discussed in 5.1.1 is used in the android application. ALP commands will be sent over this service, and their replies will be received by the application to be parsed.

5.2.2 Upgrade to newer Android SDK

While the example application worked fine, it looked and felt outdated. The target android version is 4.3. In the application for the survey, the target android version is updated to android 8.0 with support of devices running 6.0 or higher. This has an impact on two major elements namely the design and permissions. The design is upgraded to the more recent material standards after fiddling with some themes. Since android SDK 23, the application doesn't get any permissions unless specifically given access. This means permissions such as bluetooth, location and storage access have to be prompted to the user and accepted for the application to work, as seen in Figure 4.

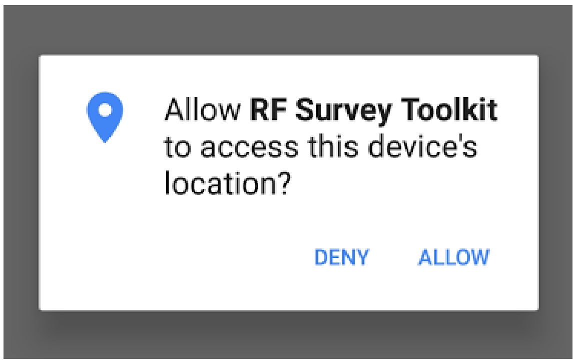

Allow **RF Survey Toolkit** to access this device's location?

Figure 4: Location permission prompt.

5.2.3 Location

After the location permission is granted by the user, the location is requested using Google's Fused Location Provider API. This API fuses data from multiple sensors such as GPS and Wi-Fi to get the most accurate location reading of the smartphone. Low power consumption location can also be requested, but is not really a priority in this implementation. When a new location is found, this is stored in a currentLocation variable. This variable will be used when parsing and storing the responses of the gateways.

```
//create location request with high accuracy as a priority
protected void createLocationRequest() {
       mLocationRequest = new LocationRequest();
       mLocationRequest.setInterval(10000);
       mLocationRequest.setFastestInterval(5000);
       mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
   }

//start location updates, called on onResume()
private void startLocationUpdates() {
       mFusedLocationClient.requestLocationUpdates(mLocationRequest,
              mLocationCallback,
              null /* Looper */);
   }

//stop location update, called on onStop()
private void stopLocationUpdates() {
       mFusedLocationClient.removeLocationUpdates(mLocationCallback);
}

//callback when new location is found, variable currentLocation is updated
mLocationCallback = new LocationCallback() {
          @Override
          public void onLocationResult(LocationResult locationResult) {
              for (Location location : locationResult.getLocations()) {
                  currentLocation = location;
              }
          };
       };
```

5.2.4 ALP Generator

To query the nearby gateways, the modem has to send a message to which every gateway will respond. This is done in the `GenerateALP()` function, which will generate the ALP command with a random tag each time preceded by a serial wrapper. This random tag is used to sort the responses with their corresponding request.

- **Bytes 0-3:** AT Command shell handler (AT$0xd7)
- **Bytes 4-6:** Serial wrapper
- **Byte 7:** Start of ALP, Tag response action (0xb4)
- **Byte 8:** Tag, random each time
- **Byte 9-13:** Forward command over interface
- **Byte 14:** Access class (default: 0x01)
- **Byte 15:** Action, readFileData (0x01)
- **Byte 16:** File ID (default: 0x00 = firmware file)
- **Byte 17:** File offset (default: 0x00)
- **Byte 18:** File length (default: 0x08)

5.2.5 Timer handler

To periodically send these commands, a runnable is created. This runnable will run every `ALPDelay` ms, which is user configurable. A new ALP command is created using the function described in 5.2.4. This byte array is sent over the BLE UART service, and some info is printed in the terminal.

This runnable is called in a handler object, named `PeriodicALP`. To start the periodicSurvey, this runnable is posted to the handler. To stop it, the runnable is removed as a callback.

```
// Create the Handler object, runnable will be called on this handler
Handler PeriodicALP = new Handler();

// Periodically send ALP commands
final Runnable PeriodicSurvey = new Runnable() {
     @Override
     public void run() {
         byte[] ALP = ALPHandler.GenerateALP(accessProfile, fileID,
            fileOffset, fileLength);
          //send data to service
          // Do something here on the main thread
          mService.writeRXCharacteristic(ALP);
          //Update the log with time stamp
          String currentDateTimeString =
             DateFormat.getTimeInstance().format(new Date());
          listAdapter.add("[" + currentDateTimeString + "] Querying
             gateways");
          messageListView.smoothScrollToPosition(listAdapter.getCount() -
             1);
          Log.d("Handlers", "Called on main thread");
          PeriodicALP.postDelayed(this, ALPDelay);
      }
};

//Start the runnable (called when the start button is clicked)
//Start the ALP runnable task by posting through the handler
PeriodicALP.post(PeriodicSurvey);
% * <matsdemeyer@gmail.com> 2018-01-30T16:00:24.168Z:
%
% ^.
//Stop the runnable (called when the stop button is clicked)
//stop the ALP commands
```

```
PeriodicALP.removeCallbacks(PeriodicSurvey);
```

5.2.6 ALP Parser

The LRWAN modem always replies to the ALP request. These responses are structured as follows:

- **No Gateway responses:** (0xc0, 0x00, 0x02, 0xe3, TAG) The 0xe3 byte means there are no gateway responses, the first three bytes are the serial wrapper (length = 0x02)
- **Responses:** (gateway response 1, gateway response 2, ..., gateway response N, 0xc0, 0x00, 0x02, 0xa3, TAG) This final part is the same response from the modem as before, this time with byte 0xa3 meaning there are preceding responses. To parse this, every gateway response has to be parsed an added to a result object.

Parsing is done using the `ParseALP(byte[] response)` function. This function will be called everytime a new byte array arrives over BLE, these byte arrays are concated until a valid return is given by the `ParseALP()` function.

The first check is done to check if the ”No Gateway Responses” array is found, which will add the current location along with the tag to the `NokResults` arraylist of results. This tag is also moved from the list `unhandledTags`, a list in which every tag is stored until it is handled, to the list `handledTags` containing handled tags.

The second check is to check if the (0xc0, 0x00, 0x02, 0xa3, TAG) sequence is found, which indicates the presence of gateway responses preceding this sequence. This last sequence is cut off of the array, leaving only (gateway response 1, gateway response 2, ..., gateway response N). Because the firmware file is queried, these responses will have an identical length. First a result object is created with the current location and tag of the response. This tag is, once again, moved from `unhandledTags` list to the `handledTags` list. Secondly, the sequence is cut into chunks of 39 bytes, one for every gateway response. Out of these responses, the UID part is stored in the result object. Finally, this object is added to the `OkResults` list. A small check is done right before parsing the chunks, to check if the default parameters are used in the settings. If this is the case, parsing is done as described. If not, this means the response will have a different structure and will require additional parsing. This more generic parsing is not yet implemented in this project.

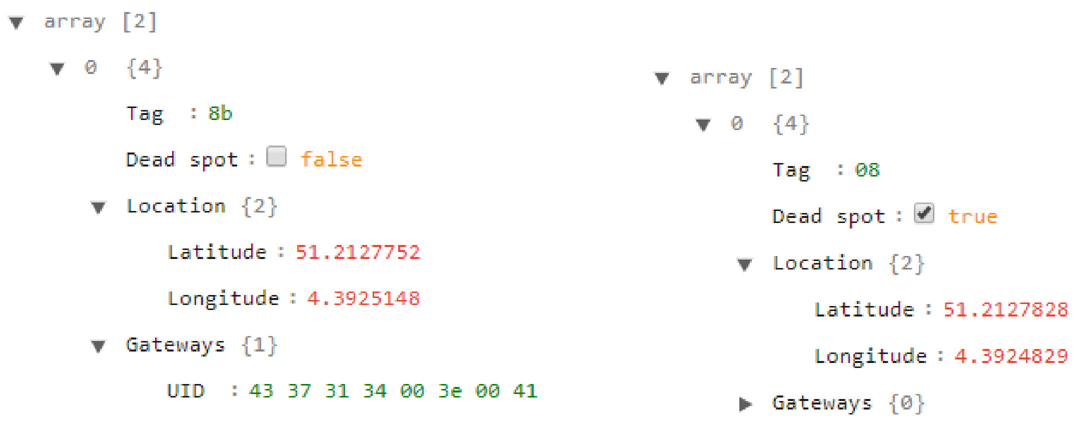
If none of the short final sequence is found, a ”NOK” is returned. This means the response is not yet complete or has an invalid structure.

5.2.7 Exporting results

The parsed results are stored in two arraylists of Result objects (OkResults and NokResults). These objects have a method to return a JSON object as follows: Tag, Deadspot (true or false), Location (lat and long), list of UIDs. When the user stops the survey, a prompt appears to discard or save the results, shown in Figure 6.

These JSON objects are combined into one JSON object and written to a file with name e.g. `Survey Result Dec 21,2017-19:17:39.json`. This file is stored on the phone itself, but can easily be sent to a remote device.

```java
public class Result {
    Location location;
    byte tag;
    List<String> UIDs = new ArrayList<String>();
    Boolean deadspot = true;
```



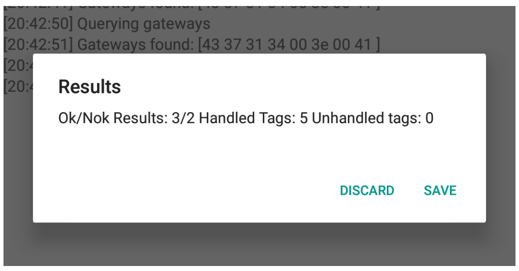
```
▼ array [2]
  ▼ 0 {4}
      Tag  : 8b
      Dead spot : ☐ false
    ▼ Location {2}
        Latitude : 51.2127752
        Longitude : 4.3925148
    ▼ Gateways {1}
        UID  : 43 37 31 34 00 3e 00 41
```

(a) JSON result with gateway UID.

```
▼ array [2]
  ▼ 0 {4}
      Tag  : 08
      Dead spot : ☑ true
    ▼ Location {2}
        Latitude : 51.2127828
        Longitude : 4.3924829
    ▶ Gateways {0}
```

(b) JSON deadspot result.

Figure 5: JSON results with gateway result and deadspot



```
[20:42:50] Querying gateways
[20:42:51] Gateways found: [43 37 31 34 00 3e 00 41 ]
```

Results

Ok/Nok Results: 3/2 Handled Tags: 5 Unhandled tags: 0

DISCARD SAVE

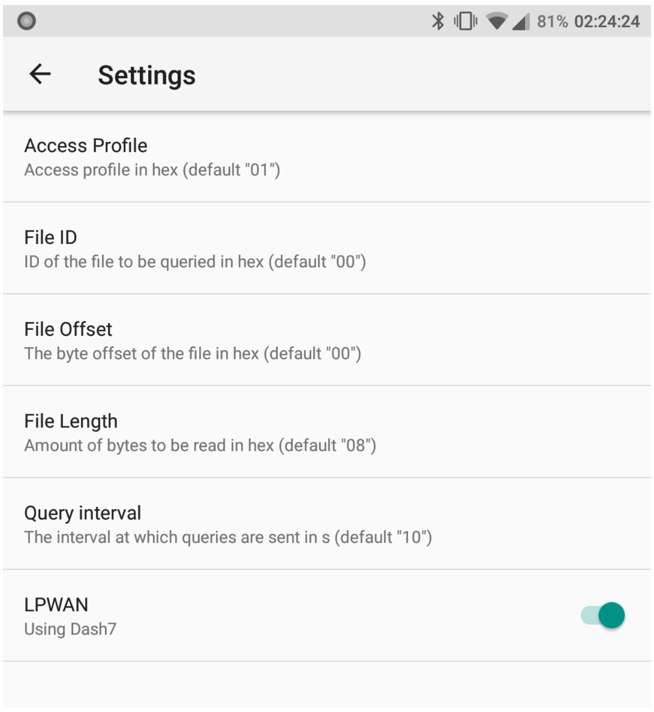
Figure 6: Prompt to save the survey results.

5.2.8 Settings

To make the application more user configurable, some parameters are configurable in the settings screen of the application. To implement settings, a Settings activity is used which allows the user to configure certain preferences. This guide is followed to implement these preferences. The preferences are stored as key value pairs in the SharedPrefences, accessible by other activities in the application. The SettingsFragment will load the preferences from the xml resource, to be displayed in the settings screen. The following parameters are configurable:

- **Access Profile:** Access profile used in the ALP command (default 01)
- **File ID:** ID of the file to be read (default 00)
- **File offset:** Byte offset of the file (default 00)
- **File Length:** Amount of bytes to be read (default 08)
- **Query Interval:** Interval in seconds (default 10)
- **LPWAN:** Dash7 or LoRaWAN (currently not used)

The end result is shown in Figure 7



81% 02:24:24

Settings

Access Profile
Access profile in hex (default "01")

File ID
ID of the file to be queried in hex (default "00")

File Offset
The byte offset of the file in hex (default "00")

File Length
Amount of bytes to be read in hex (default "08")

Query interval
The interval at which queries are sent in s (default "10")

LPWAN
Using Dash7

Figure 7: Settings Screen.

5.2.9 Code Flow

The flow of the code when surveying is shown in Figure 8 and Figure 9

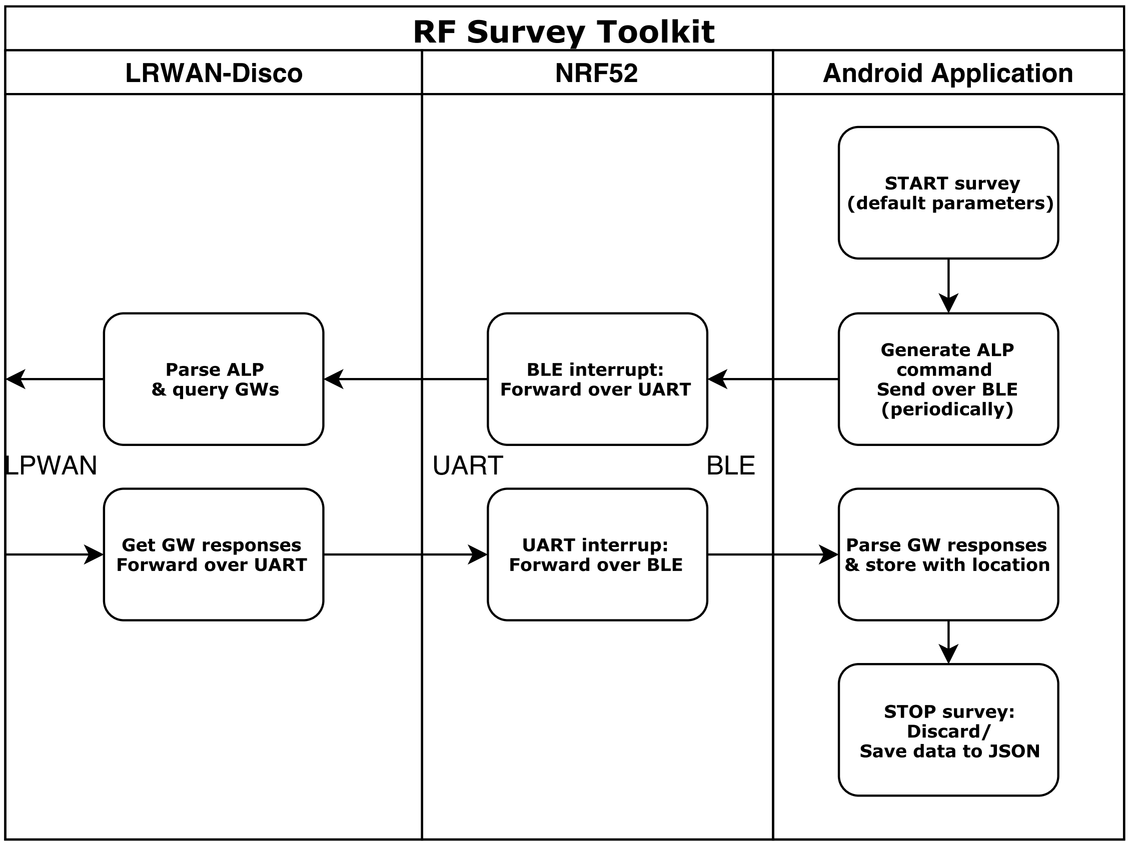

RF Survey Toolkit
LRWAN-Disco
NRF52
Android Application
START survey (default parameters)
Generate ALP command Send over BLE (periodically)
BLE interrupt: Forward over UART
Parse ALP & query GWs
LPWAN
UART
BLE
Get GW responses Forward over UART
UART interrup: Forward over BLE
Parse GW responses & store with location
STOP survey: Discard/ Save data to JSON

Figure 8: Main code flow when gateways are found.

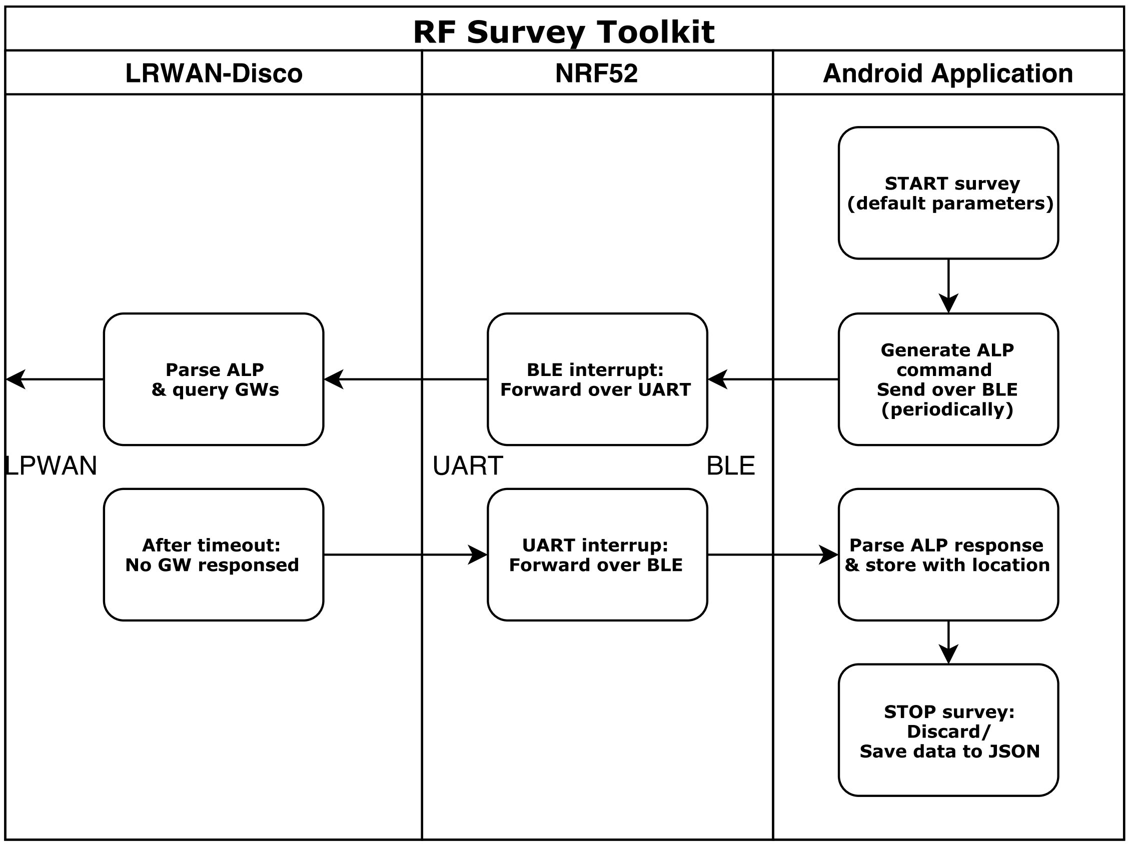

RF Survey Toolkit
LRWAN-Disco
NRF52
Android Application
START survey (default parameters)
Generate ALP command Send over BLE (periodically)
BLE interrupt: Forward over UART
Parse ALP & query GWs
LPWAN
UART
BLE
After timeout: No GW responsed
UART interrup: Forward over BLE
Parse ALP response & store with location
STOP survey: Discard/ Save data to JSON

Figure 9: Main code flow when no gateways are found.

6 Tests

Tests are done using another LRWAN Discovery board, flashed as a gateway. To test multiple gateways, when cutting the response into chunks, the response of the first gateway is copied as a second gateway. This tests adding multiple gateways to the result object, albeit two times the same UID. Tests are also done with the gateway being powered on and off during the survey, to check if it correctly detects and reports blind spots.

7 Reflection

While this survey toolkit works well in my testing, some further and more in depth testing has to be done. In this section, some issues, problems and shortcomings will be discussed

7.1 Issues when making this project

The main issues I had when making this project were the ALP command parser and the UART to BLE forwarding. The ALP command parser was rather complex but after some thorough debugging, it was working. To transparently forward every UART byte over BLE, I initially put this BLE transmit code in the UART interrupt callback, which crashed the program or omitted some bytes. This was due to the delay of the BLE transmits, handled in the interrupt callback. When offloading these bytes into a fifo and sending this fifo when ready over BLE fixed this problem.

7.2 Analysis of problems & shortcomings

The main shortcoming of the toolkit in its current from is the lack of a generic ALP parser. This parser is currently only made for parsing the UIDs out of the firmware file and storing these. The user can choose alternate settings, but the main survey code of storing the UIDs only works when the default parameters are used. A logical next step would be to generalize the parser, or at least add a second, more general parser. Another good addition would be sending the JSON files to a remote location or database to process them.

8 Conclusion

Considering the fact this course started the 7th week of the semester, I believe a lot has been accomplished in a relatively short time. A working prototype for surveying dash7 networks is provided with an android application to control everything. While some issues occurred during the project, everything went relatively smooth overall. This is also thanks to the support of Kwinten Schram, who provided a great starting point for the BLE board and the help of Glenn Ergeerts from some specific embedded and Dash7 related questions. More extensive testing has to be done in the future, along with some improvements such as a more generic ALP command parser.

9 Practical info

The code can be found on github. The `nrf52SDKWrap.c, fifo.c` files from the embedded project can be found in `/Embedded NRF Project/pca10040/s132/armgcc`. The main file can be found in the parent directory. Along with the gradle project of the android application, a debug .apk is provided.