Laboratory 6: RF channel measurement and using NTP servers

Introduction

This laboratory aims to equip you with knowledge on how to:

- Measure how RF propagation influences the communication link.
- Utilise a NTP time server.

Equipment and resources

Hardware:

- ESP8266 Aston IoT hardware platform.
- USB A to micro B cable.
- The shared class computer acting as an access point.

Software & documentation:

- Arduino Integrated Development Environment (IDE).

A reminder on saving and file name convention

You should keep a copy of every task that you complete on your **student network drive (H:), in the folder H:\EE3IOT\labs.** Sketches should be saved in the format of **surnameinitialLTX**, where **L** is the lab number and **X** is the task number.

For example, if Fred Smith has completed task 1 of Lab 1, the sketch should be saved as **smithfL1T1**.

Task 1: Measuring received signal strength

In the lecture, the Free Space Path Loss equation was introduced. This task will help you determine if the equation holds true in a typical indoor environment (hint: it does not!).

1. Create a copy of your sketch from **laboratory 2 task 1** as a starting point for this task. Utilise the file naming convention given above and used throughout the module.
2. In the **loop()** function, add code to print the received signal strength indicator (RSSI) on the top line of the LCD display once a **second**. There is no need to use the cyclic executive for this.
   - The `WiFi.RSSI()` class method (think of this as a function) will return the RSSI level in dBm in the range of 0 to -100.
   - Refer to previous laboratories for how to use the LCD display.
3. Estimate the approximate distance to the WiFi access point (router or phone). Make a note of this value in the box below.
4. Run the code on the board and make a note of the received signal strength in the box below.
5. Convert the received signal power from dBm to Watts and record the value in the box below (or on paper).

Distance to access point:	m
Measured RSSI:	dBm
Received signal power:	Watts

6. Assuming the WiFi router transmits 100mW EIRP and the transmitting and receiving antennas have a gain of 1, work out the theoretical received power in Watts and dBm below (or on paper).

Theoretical received power:	Watts
Theoretical received power:	dBm

7. Calculate the received power as a percentage of the ESP8266's receiver sensitivity (91 dBm) below or on paper.

Margin with respect to ESP8266 sensitivity:	%

8. If you are able to, move the board around the room (note once programmed, all you need is to plug the board in to a USB power source, such as computer or mobile phone charger) and take received signal strength measurements at various distances.

   Plot at least 4 RSSI figures in the excel file included on blackboard. How close do the results match the theoretical trace?

Task 2: Using a Network Time Protocol (NTP) time server

Time can be problematic to keep track of in IOT applications. Some microcontrollers utilise a 32,768kHz external crystal and a 2^{15} (32,768) frequency divider that is internal to the microcontroller (essentially just a binary counter) to generate a 1 second period clock that can be used as a source for a time keeping algorithm implemented by the microcontroller. However, if the microcontroller is reset or the power is cycled, the microcontroller will start measuring time from zero again.

There are also real-time clock (RTC) integrated circuits (chips) which utilise a similar approach to keep track of time. However, these low power chips typically utilise a battery back-up system, which allows for the time keeping functions to carry on regardless of the circuits state. These chips can be easily interfaced to a microcontroller allowing for the user to just set the time and date infrequently.

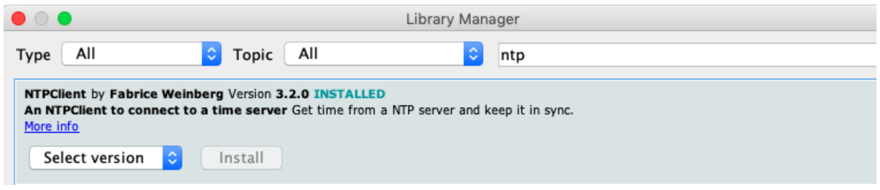
However, both approaches suffer from the problem of the 32,768kHz crystal oscillator's frequency drifting with age and temperature. This typically results in the RTC gaining or losing time as time progresses. If you have had to adjust the time of a digital quartz watch after years of use due to it gaining or losing time, this is the same problem. Such approaches still require the end user to enter the time and date periodically as the starting reference for the time keeping algorithm.

Hence, it would be much easier if there was a way for the current time and date to be communicated to the IoT device via the internet from a highly accurate and stable clock, such as an atomic clock. This is where the Network Time Protocol (NTP) comes in. NTP is a UDP based protocol, which typically uses the request and response model. The client (our Aston IoT trainer board) requests the time and the NTP server responds with a packet containing a time stamp. Note that the server is typically a lower stratum (level of communication) down from the accurate clock source and this in conjunction with network propagation results in a typical accuracy of <=50ms (Note the accuracy is different to precision and is an offset from where time should actually be). However, we can periodically request the time, which removes the problem of drift and having to adjust the current time manually.

This relatively short lab will guide you through using NTP to retrieve the current time.

1. Create a copy of your sketch from **laboratory 2 task 2** as a starting point for this task. Utilise the file naming convention given above and used throughout the module.
2. Add the NTP library (note you will only have to do this once for a particular machine):
   - Sketch | Include Library | Manage libraries
   - Search for NTP in the top right box as shown in Figure 1.
   - Select the latest version and click install.

- When complete, click close to return to the normal Arduino IDE.



NTPClient by **Fabrice Weinberg** Version **3.2.0** **INSTALLED**
An NTPClient to connect to a time server Get time from a NTP server and keep it in sync.
More info

Figure 1: Installing an NTP Library

3. Remove code at the end of your **setup()** function which opens a UDP socket for listening (i.e. UDP.begin()).
4. Include the library for NTP communication by adding `#include <NTPClient.h>` to the top of your sketch.
5. Create a class instance of type **NTPClient** called **ntpTIME** as shown below.

```
NTPClient ntpTIME( UDP, "pool.ntp.org", 0);
```

   - The class constructor takes three arguments. The first one is the class instance from the of type **WiFiUDP** you created in lab 2. This library uses the UDP class instance to actually perform the UDP communications.
   - The second argument is an NTP server which we will be using.
   - 0 is the offset in seconds from UTC/GMT. If you are working outside of the UK, replace 0 with the UTC/GMT offset in seconds.
   - Remember that this class instance should be global. Hence, it should be placed outside of any functions (after your instantiation of the UDP class and preferably towards to the top of the sketch).
6. At the end of your **setup()** function add `ntpTIME.begin();` to initialise the NTP library/stack.
7. In the **loop()** function (which should be empty), call the **update()** method for the class of **ntpTIME** (`ntpTIME.update();`) once a second using a software delay.
8. Use the **getHours()**, **getMinutes()** and **getSeconds()** methods for the ntpTIME class to print the current time to the Serial Monitor.
9. Upload your code to the Aston IoT trainer board and confirm functionality.