ESP8266

Application Note on Geolocating

Version 1.1
Espressif Systems

About This Guide

This application note describes how to geolocate a device which has an integrated ESP8266 module.

Release Notes

Date	Version	Release notes
2017.09	V1.0	First release.
2019.03	V1.1	Updated version.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at https://www.espressif.com/en/subscribe.

Certification

Download certificates for Espressif products from https://www.espressif.com/en/certificates.

Table of Contents

1. Overview

Ability to geolocate a device is indispensable for B2B and end-user applications. Accurate localization can help optimize processes in manufacturing and logistics, and also can improve user experience in many ways either known or yet to be discovered.

The most popular geolocation methods are as follows:

- Using a **GPS module**: A rather conventional and reliable approach, providing the best available accuracy for outdoor applications.
- Positioning based on **cellular networks** (GSM, LTE, etc): A reliable method for both indoor and outdoor use. However, a device needs to support cellular connectivity.
- **Wi-Fi** positioning: An efficient approach for environments where a device is located within the range of numerous Wi-Fi access points (AP).

ESP8266 can be a very good solution for Wi-Fi positioning, the location can further be refined if a host system supports cellular connectivity.

2. System Architecture

This application note assumes that a host system connects to a Wi-Fi network using an ESP8266 module, such as *ESP-WROOM-02* manufactured by Espressif. Another assumption is that this module uses Espressif's *AT firmware*.

An ESP8266 module needs to be connected to a host system via UART. The device acting as a host system is responsible for configuring, managing, and controlling the module.

An ESP8266 scans for nearby Wi-Fi access points and obtains their SSID, RSSI, and MAC address characteristics.

Then the ESP8266 module is connected to Google's Geolocation API to determine a location based on the obtained AP characteristics. However, the accuracy of results may vary depending on how many access points are available and how many of them are connected to the internet.

Availability of cellular network information is not necessary for geolocating with ESP8266. However, if the host system supports cellular connectivity, this information can be used to further improve the localization accuracy.

3. Geolocating with ESP8266

As stated previously, geolocation of a device with an integrated ESP8266 module involves the following actions:

1. A device obtains SSID, RSSI, and MAC address characteristics of the nearby Wi-Fi access points and cellular sub-systems.
2. The obtained information is then consolidated into a data block.
3. The data block is transmitted to Google's geolocation service with the help of Geolocation API.
4. The geolocation service estimates the latitude and longitude in degrees as well as the accuracy of the estimated location in meters.
5. The service returns the JSON-formatted response back to the device.

3.1. Obtaining AP Characteristics

ESP8266 is able to actively scan for nearby Wi-Fi access points operating in the 2.4 GHz band. The scanning takes only a couple of seconds, after which ESP8266 may be put into a power-saving mode, thus prolonging battery life in battery-powered systems.

The scanning time is that short, because the process involves executing a single AT command that lists all APs available in the area and their corresponding SSID, RSSI and MAC addresses.

1. Make sure that ESP8266 is powered up and fully operational by running the command:

   ```
   AT
   ```

   The response must be:

   ```
   OK
   ```

📖 ***Note:***

*All the **commands issued must be terminated with CR-LF** (\r\n combination).*

2. Switch ESP8266 to Station mode by running:

   ```
   AT+CWMODE=1
   ```

3. To obtain the list of all available Wi-Fi access points, run the command:

   ```
   AT+CWLAP
   ```

   The command returns miscellaneous data, including the list of available APs, their SSID and RSSI values, and MAC addresses.

See the example of the command's output below.

```
+CWLAP:(2,"HotelFlower",-76,"c8:3a:35:b3:16:48",11,0,0)
+CWLAP:(3,"IoTBits",-93,"08:bd:43:66:6a:86",6,-27,0)
OK
```

📖 ***Note:***

For more information on the AT commands, please refer to Espressif's AT Instruction Set.

3.2. Obtaining Geolocation

For sending a request to the geolocation service, ESP8266 must be connected to a router with Internet access. To establish a connection, ESP8266 should be switched to Station mode.

3.2.1. Establishing a Secure Connection

1. ESP8266 can be connected to an access point by running:

```
AT+CWJAP="SSID","password"
```

📖 ***Note:***

*Here **SSID** and **password** are placeholders for the actual network characteristics.*

If the connection is successful and ESP8266 receives a local IP address, the command returns:

```
• WIFI CONNECTED
• WIFI GOT IP
```

ESP8266 can now establish a secure connection with the geolocation service using the https protocol.

2. Initialize the SSL buffer by running:

```
AT+CIPSSLSIZE=2048
```

3. Connect securely to Google's geolocation service by running the command:

```
AT+CIPSTART="SSL","www.googleapis.com",443
```

If successful, this command returns:

```
• CONNECT
• OK
```

Otherwise, ESP8266 returns:

```
ERROR
```

📖 Note:

SSL is memory-intensive. Make sure that even in the worst-case scenario the outgoing and incoming transactions do not exceed the size of the SSL buffer.

3.2.2. Using Geolocation API to Obtain a Location

Google's *Geolocation API* requires a number of parameters, which must be sent via a POST request formatted as JSON to the following URL:

`https://www.googleapis.com/geolocation/v1/geolocate?key=YOUR_API_KEY`

📖 Note:

For quota management purposes, the user must get a key from Google and replace `YOUR_API_KEY` *in the URL above with the provided key.*

The steps to obtain a location using Google API are as follows:

1. Send a standard HTTP POST header with your API key.
2. Send POST data with JSON-formatted geolocating parameters.

The host system should initiate a request at this stage. For sending data to the geolocation service, use the `CIPSEND` command, followed by the data length to be sent in bytes (e.g. 336 bytes):

```
AT+CIPSEND=336
```

ESP8266 responds with:

```
>
```

After that, send the actual request as follows:

```
POST /geolocation/v1/geolocate?key=AIzaSyCNStbdlDS_L HTTP/1.1
Host: www.googleapis.com
Content-Type: application/json
Content-Length: 166

{"homeMobileCountryCode": 310, "homeMobileNetworkCode": 410, "radioType": "gsm",
"carrier": "Vodafone", "considerIp": "true", "cellTowers": [], "wifiAccessPoints":[]}
```

📖 Note:

- *The JSON-formatted request format must comply with the structure outlined in the Google geolocation API documentation: https://developers.google.com/maps/documentation/geolocation/intro.*
- *The* `cellTowers` *and* `wifiAccessPoints` *fields contain objects which are omitted in the above example to make the request contents more comprehensible.*

4. Power Consumption

4.1. Testing Environment

For evaluation of the power consumption, this application note uses a standard AT firmware for ESP8266 (based on non-OS SDK V2.0.0) and the following hardware:

- PC as the host system
- ESP-Launcher as the ESP8266 module
- NetGear 2.4 GHz router with Internet access
- 26 MHz crystal oscillator
- 40 MHz SPI flash (QIO mode)

Some tips for reducing power consumption:

- Disable RF calibration on waking up from deep sleep
- Use low power flash running at 26 MHz, if possible
- Use lower transmit power, if possible.

📖 Note:

The current consumption data shown in this chapter also includes the current consumed by the flash chip.

4.2. Brief Note on Power Consumption

The approximate current consumption in some low-power modes is given below.

- Modem-sleep mode: ~15 mA
- Light-sleep mode: ~3 mA
- Deep-sleep mode: ~20 uA (also depends on a flash standby mode)

The figure below shows the power consumption for the test, during which ESP8266 detected more than 10 APs in about 2.2 sec. The ESP8266 module had the following setting:

- Low-power mode set to Modem-sleep (i.e. `AT+SLEEP=2`)
- DTIM interval for connections to an AP set to ~100 ms

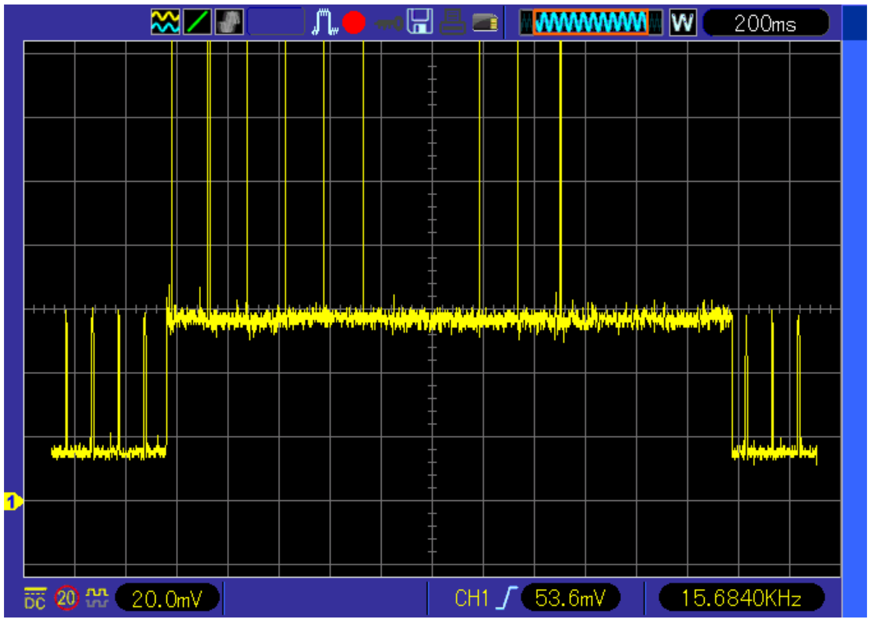

Figure 4-1. Active SSID Scanning with Modem-sleep Mode Enabled (X: 200 ms/div, Y: 20 mA/div)

- For applications that stay connected to APs:

  For ESP8266 maintaining connection with an AP, with the DTIM interval of 100 ms and in Light-sleep mode while inactive, the average current consumption is less than 20 mA.

- For applications that simply scan for APs:

  A typical scan time for active SSID scanning ranges from 2 sec to 2.5 sec. The average current consumption during active scanning is about 75 mA.

📖 Note:

- *Starting from V1.6.0, AT firmware supports passive scanning, which can help further reduce current consumption.*

ESP8266 may be put to deep sleep when scanning is not required, by running:

```
AT+GSLP=<time>
```

The power consumption for ESP8266 in Deep-sleep mode is in the order of micro-amperes, which eliminates the necessity to disconnect its power supply during inactivity.

📖 Note:

The power consumption data for transmitting geolocation parameters over a Wi-Fi network has not been listed in this application note because it heavily depends on the network setup, latency of the network, and the interference from nearby devices. The end users are encouraged to evaluate the power consumption themselves under the conditions in which their product is supposed to be used.

Espressif IoT Team

www.espressif.com