Node System Documentation

T. Buckingham

June 9, 2021

Contents

1 Introduction

As part of the Engineers without Borders challenge we were tasked with coming up with a solution to a problem that the people of Lobitos and Periditas in Peru, face. Among those described within the document we chose to focus on their water supply; which they have problems with water tapping and sanitisation. Our team's application works in conjuncture with a network of nodes that log data about their respective areas, allowing workers to locate the areas with problems more effectively and reducing the man hours needed to find and fix regularly occurring problems

2 Device

The nodes within the network can work on a plethora of IoT devices all with their benefits and draw backs, a few will be mentioned here however, the key requirements will be network capabilities and data storage connectivity such as an SD card. Systems such as these are designed to be as cost effective as possible however, this will largely be beyond the scope of this project as the development and documentation will be on development versions of the ESP32 and ESP8266 which provide a much simpler and easier interface but with elements unnecessary to this project.

2.1 Boards — Found here

The are a wide range of network orientated boards available all with different benefits, features and costs. The ESP series of boards is among the most popular and as the project was developed on two of these boards (ESP32 and ESP2866) this report will also look at others from the same family. Both boards used in this project were already sourced and were not chosen specifically for it, as such the ESP32 was chosen to be the bridge node due to it's dual core processor and the ESP8266 was chosen to be the logging node for its lower cost.

When deciding for the data logging node the main features we are looking for is enough RAM to handle a single request at a time. As an we are going to take the formatting of a response and assume that all measurements are floats or longs (4 bytes on a 32-bit machine), with the formatting $'''' ticket'' :' [ticket - num] : [r_1, r_2, r_3, r_4, r_5, r_6], \ldots''$ — the ticket system is explained later in this document

Starting the request with ''{''ticket'': requires 11 characters.

Ending the request with }'' requires 2 characters.

The unix timestamp ''[timestamp]:'' requires a total of 13 characters.

Each log is comprised of 6 longs/floats with brackets and commas although these will be captured as longs or floats the precision needed can be changed and each sensor reading has max realistic values. The expected readings are described further later in this document but a minimum total of 19 characters are expected to be used.

Readings In total this system requires $45 + 32(n - 1)$ characters, where n is the number of readings.

With the ESP8266 used in this project this would allow for ~1124 logs to be sent at one time, now this of course ignores the board using RAM for other uses and assumes — with a 1 minute logging interval — that the user waits over 18 hours to record and store the devices readings. If the user reguarly polls each node on the network then using this board will not be a problem however, it will produce more network traffic which could slow down the network. Two solutions to this problem are as follows

- Using a board with higher RAM

  - The main issue with this is allowing for larger requests that then have to be dealt with through the bridge node, if this isn't handled properly then the bridge node may crash and reboot — losing all other current requests.

- Sending requests in chunks
  - The bridge node currently store requests in RAM however, the if the number of requests is too much for the boards RAM then they can be stored and read on an SD card instead, this will reduce response time but will help to prevent memory overflows along with proper checks in place. This allows allows for the logging nodes to have smaller RAM sizes and potentially be less costly.

All boards are easily programmable using the Arduino IDE and related libraries available to the user additionally, code can, in theory, be written in such a way that the same sketches can be used on any device mentioned; reducing the amount of time writting and debugging the implementation.

2.1.1 Device specifications comparison

Table 1: Comparison of key features between ESP models

	ESP32-C3	ESP32-S2	ESP32	ESP8285	ESP8266
CPU	Single core, 160MHz	Single core, 240MHz	Dual core, 160 or 240 MHz	Single core, 80MHz (up to 160MHz)	Single core, 80MHz (up to 160MHz)
RAM	400KB	320KB	520KB	160KB	160KB (36KB user)
ESP-MESH	Yes	Yes	Yes	Yes	Yes
WiFi	802.11b/g/n	802.11b/g/n	802.11b/g/n	802.11b/g/n (up to 65Mbps)	802.11b/g/n (up to 65Mbps)
SPI	3	4	4	2	2
Security	secure boot, flash encryption	secure boot, flash encryption	secure boot	x	x

The key differences among the modules is the level of security and performance. It would be suggested that bridge nodes be installed on an ESP32 variant due to the higher level of performance thus increasing the throughput of requests by the client — especially if more than one client is in use.

Security is a concern of network therefore having a device with a secure boot would help prevent from physical access attacks however, depending on the nature of the placement e.g. below or above ground, it may not be necessary. The security of the network itself and the cost of devices needs to be balanced with the features of each board for example, using a module or custom board rather than a development board however, this is beyond the scope of project subsequent documentation.

Purchasing bulk modules would be substantial saving however, to reduce costs it would be suggested that logging nodes be a cheaper model variant to reduce costs and allow more budget for the sensors needed.

2.1.2 Power consumption comparison

Table 2: Comparison power consumption in different modes

	ESP32-C3	ESP32-S2	ESP32	ESP8285	ESP8266
Modem-sleep	15–20mA	12–19mA	27–68mA	15mA	15mA
Light-sleep	130 µA	450 µA	0.8mA	0.9mA	0.9mA
Deep-sleep	5 µA	20 µA–170 µA	10 µA–150 µA	20 µA	20 µA
Active (RF working)	87–335mA	68–310mA	95–240mA	72–197mA	56–170mA

All information was found in their respective datasheets links provided here

All modes have different subsections which is why some have ranges for their given sections.

- Modem-sleep is when the CPU is powered on however, some such as the ESP32-C3 has two operating modes and the ESP32 is a dual module therefore, the power consumption will depend on what state the processor(s) are in.
- Light-sleep in this mode the CPU is typically suspended whilst maintaining WiFi connection without data transmission.
- Deep-sleep mode is when the WiFi is turned off and either only the RTC is on and/or a sensor-monitored pattern is active.
- RF data usage is dependant on the the signal frequency and whether data is being transmitted or recieved.

2.2 Sensors

- Turbidity
- Total Dissolved Solids (TDS)
- Pressure
- Flow rate
- pH
- Temperature

3 Network

The network is built using a mesh network, with a mesh network each node is connected to other nodes that are within range and keep a routing table of the network. This allows the nodes to pass requests and messages to nodes outside the range of the client. The main advantage of this is allowing for nodes to be spread out across the areas needed without requiring multiple access points and clients, keeping the network whole so that any client can access any node within the network.

The inital idea was to use GSM to send SMS messages between the nodes however, this can become impractical, slow and costly when sending large data logs at once. Using a mesh network may be slower in areas however, it would not require an additional cost such as GSM. The main drawback of using a mesh network is each nodes communication distance, depending on how spread out each node is there may be nodes that are cut off from the rest of the network and depending on how short this cut off distance is this may require a lot of extender nodes in the network to improve range.

A potential alternative to this is using the Long Range Radio (LoRa) network, this will be discussed more in depth here however, briefly the LoRa network is a low power, low range communication network aimed towards IoT devices. It can achieve a data spead of 0.3 kbit/s to 27 kbit/s which is not a high speed but for the data transfer of simple strings it is more than sufficient, the key feature of this network however, is the range which is quoted at 2–5kM in urban areas and 15km in suburban areas and with antennas these distances can fairly easily be achieved.

3.1 HTTP vs MQTT

In this section we will discuss some of the key differences between HTTP and MQTT, why HTTP was ultimately chosen for this project and whether we would make the same decision in real world deployment.

- HTTP

Protocol HTTP is a request response protocol that only keeps the connection between client and server whilst the request is being made. This reduces the number of connections between the bridge node(s) at any one time however, it does mean that a connection must be establish each time a request is made thus slowing down the time it takes for a request to complete.

Data transfer Requests and responses are handled using encoded text with the requests using text parameters and reponses being sent in text formats such as plain, HTML, JSON, etc. This is fine when wishing to handle or parse these data types but for raw data such as binary data this is not possible and can result in slower responses times.

Accessing the server A HTTP server is easily accessible and can be done so using just a web browser or a minimal program; most high level languages provide libraries for HTTP clients and servers. This would allow a user to access the node network even if their system or program is non-functional, albeit without the additional features provided by the program.

- MQTT

Protocol As opposed to HTTP, MQTT is a client—broker system which allows for both clients and brokers to send and recieve data. This works by using a subscribe-publish model in which clients subscribe to a variable on the server and when this variable is updated the server sends the new data to the client, conversly clients can publish data directly to the server which can in turn be sent to other clients e.g. a sensor publishing to server with a logging node storing the data. This does require a constant connection between client and server in order to recieve the server's published data.

Data transfer Unlike HTTP the data transfered between client and server does not have to be encoded text and can be raw binary data, this in turn speeds up responses and can requires less parsing on the clients part.

Accessing the sever The server is accessed via a 'broker' which is a little more involved to set up and cannot be done through a simple request through a web page. Finding an already implemented MQTT broker is not difficult however, implementing a bespoke one or integrating it into another program can be more a challenge.

3.2 LoRa

4 References