Advances in Computer, Signals and Systems (2023)
Clausius Scientific Press, Canada

DOI: 10.23977/acss.2023.070807
ISSN 2371-8838 Vol. 7 Num. 8

A Modular Home Automation System Based on IoT Technology

Qizhang Deng

Warsaw University of Technology, Warsaw, 00-665, Poland

Keywords: ESP8266, Arduino, MQTT, home automation

Abstract: With the continuous development of the information society, wireless networks have appeared in daily life on a large scale. The necessities of the modern human lifestyle can be answered and the quality of life advanced through wireless home automation, and it will definitely change people's family lifestyle. The thesis explores the possibility of using cheap IoT (Internet of Things) components of the Arduino class in a wireless environment home automation system: a demonstration system developed in this thesis uses the ESP8266 boards to serve two rooms in a house, with a clear path to extend the system to a useful size. Apart from easy installation and immediate extension possibility, the equal treatment of same-room and other-room operation is the key component of the suggested method. The design uses the MQTT protocol running on ESP8266 boards. Power plugs, lighting systems, temperature and humidity sensors, smoking, gas, and fire detectors, emergency and security systems, as well as the expansion of basic remote control ability, can all be controlled locally or remotely by the system.

1. Introduction

1.1. Background

In order to create a centralized administration and intelligent control of the home, the so-called smart home merges audio and video devices, network communication technology [1], security features, automatic control technology, and wiring technology using the house as a platform.

These are more important than any other technology because they are so close to the users. Mature products use ZigBee, Z-Wave, and WIFI technologies extensively. Modern Home Automation systems are considered to be the prime example of implementing the ideas and technologies of IoT (Internet of Things), these technologies include particular control protocol (such as MQTT), and small, cheap hardware platforms such as microcontrollers of the Arduino family.

1.2. Aim and scope

This thesis demonstrates a low-cost smart home system based on the MQTT protocol running on the ESP8266-Arduino platform. The demonstrator will just control several keys and LED lights. Its main purpose is to verify that the use of MQTT on Arduino can also allow users to experience a more convenient and comfortable lifestyle and a safer environment. Its main feature is that it is meant to

be easy to operate, very friendly to users, and easy to install indoors, without users need to use any additional wiring.

In the demonstrator, key switches are used to simulate light intensity sensors, temperature and humidity sensors, gas and flame sensors, and LED lights are used to simulate actuators such as infrared controllers, motors, and buzzers. The demo in this thesis bases on the minimum usable system, but the consider action of final system is also included. The key feature of the final system is to communicate with the user in a user-friendly manner. Therefore, likely further development of this system will use an LCD data display with push buttons to provide the user with status and feedback parameters on the screen. Users can add a GSM SHIELD, which adds to the system has the ability of remote communication. This guarantees that the system should be easily accessible from anywhere and at any time. When the simplest model is completed, extended devices will be added easily. The required modularity and extensibility is provided “by design”, due to the features of the MQTT protocol.

2. Requirements and ideas

The general requirements, or other guidelines adopted for the home automation system under design, are as follows:

The system should be build using familiar ideas, hardware, software environments and experiences of the IoT system.

Cost factors will have to be considered, for choosing the hardware. Preference should be given to hardware in the same price that has more resources and extensions.

As far as possible, the system should use the pre-existing technology that is likely to be present in the home, where the system will be installed. In particular, the system could use the existing WI-FI access point.

The system (Fig 1) may be severed by a WI-FI router, without using the WAN (external internet) access function. When the Personal Area Network (PAN) is open, all local devices can use it for communication. With the additional device the system can get remote communication function, but this is not absolutely necessary for basic operation. This is in contrast to those house automation system that rely on internet services (recently offer location in the cloud).

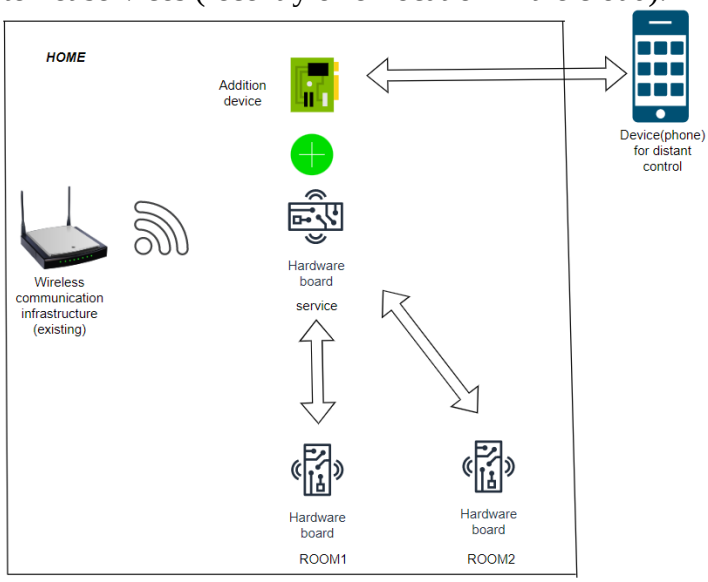

Figure 1: The system of home automation-the idea

3. Underlying technologies and design options

In this chapter, the author will introduce the theoretical background of the IoT, transmission control protocols and hardware we are using in this system.

3.1. IoT

The IoT is made up of millions of connected devices communicating with each other via radio frequency and cellular signals - always on. An addition to the Internet is the Internet of Things. It is to connect devices to the Internet through sensors, chips, and wireless modules. The resulting structure is highly complex. This thesis gives the author the opportunity to redesign home automation, and to show that it can be organized by bypassing the massive and complex device structures and protocols of the traditional Internet.

3.2. Protocol stacks

There are two main protocol stacks to be consider: OSI or TCP/IP.

OSI (Open Systems Interconnection), defines a communication model and a protocol stack that was developed in the context of traditional telecommunications and data transmission. There are seven layers in OSI, and each layer implements its own protocols and functions as well as completing the interface communication with the layer above it [2].

Transmission Control/Internet Protocol, or TCP/IP, is the name given to a protocol stack develop directly with Internet in mind. Quite naturally protocols used in IoT systems will be mapped to the levels of this model, rather than the OSI model. In particular, TCP (Transmission Control Protocol) is the transport-level protocol that will be used to organize transmission in the system developed here. Another crucial protocol of this thesis, MQTT, is the application protocol.

3.3. Network topology and system architecture

In this thesis the reader will find some comments on the MQTT protocol (a structure with some distinguished elements).

3.3.1. Broker

This pattern mainly serves decoupled components in a distributed system. Among them, the broker acts as a server (mediator). Devices connected to it can interact without knowing about each other. A broker is responsible for the communication between each device. The most classic representative of an application protocol for this type of interaction is MQTT. A client can "subscribe" to an event or pattern of events to indicate his interest, and he will then receive notifications of any events created by other clients and "published" that fit his registered interest. Events are propagated asynchronously to all registered clients interested in that given event [3].

3.3.2. MQTT

A simple application-layer protocol for Internet of Things applications is called MQ Telemetry Transport (MQTT). Over a TCP/IP network connection, data is transferred using a publish/subscribe technique. MQTT includes two parts: a broker and clients (Fig.2).

MQTT means a lightweight publish-subscribe system, which is tailored for low power consumption and small data traffic. The client can publish some parameter data under a topic (Topic). When these are published, another client who is interested in the topic can subscribe to it and receive

updates. The subscription is handled by the broker. MQTT provides various levels of quality of service. Using broker Quality of Service (QoS)(Fig. 3), a crucial message may be sent to the target. The delivery is then guaranteed, although the transmission speed will be slower. Less critical data can be sent using a lower QoS, more akin to an UDP-style "fire-and-forget" event[4].

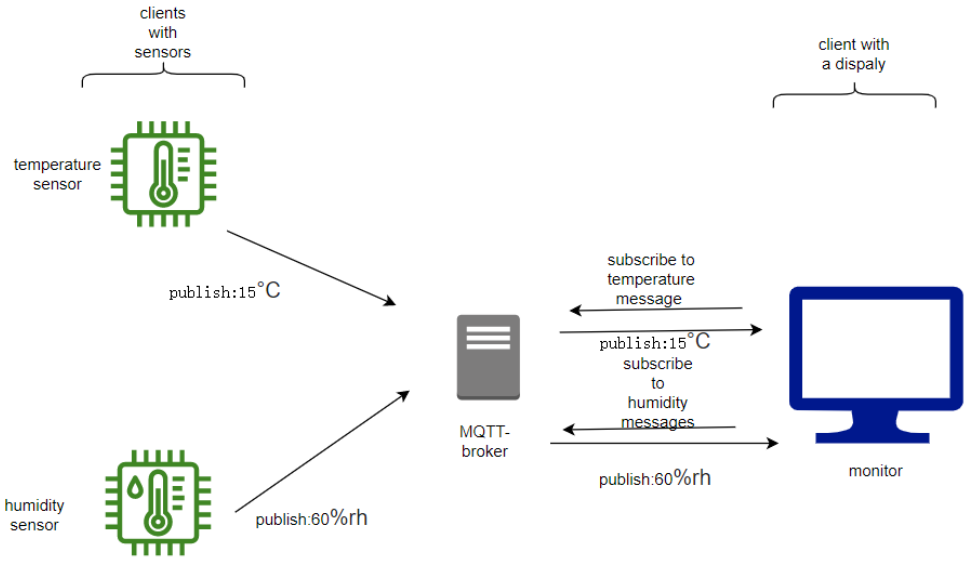

Figure 2: The idea of MQTT

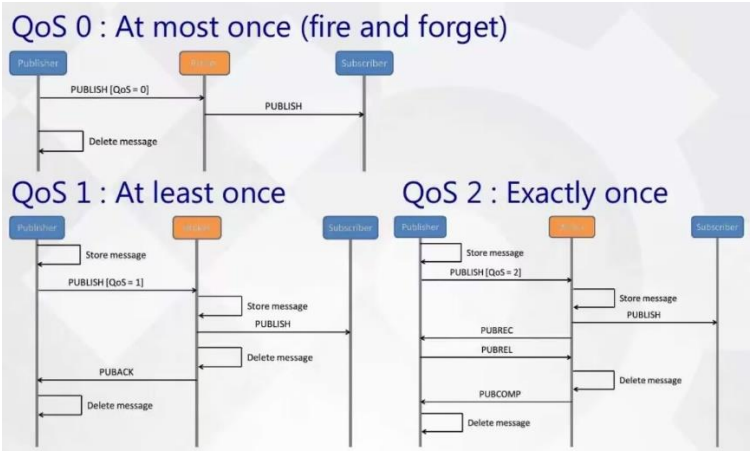

Figure 3: The MQTT quality of service concepts[5]

3.4. Wireless network interface

Examination of possible wireless technologies is explained by the need to avoid laying cables and thus demolishing the building.

3.4.1. Wi-Fi

Wi-Fi is currently the most common way of exchanging data with radio waves. WIFI is a series of wireless network protocols based on the IEEE 802.11 family (Fig.4) of standards [6].

Generation/IEEE Standard	Maximum Linkrate	Adopted	Frequency
Wi-Fi 6E (802.11ax)	600 to 9608 Mbit/s	2019	6 GHz
Wi-Fi 6 (802.11ax)	600 to 9608 Mbit/s	2019	2.4/5 GHz
Wi-Fi 5 (802.11ac)	433 to 6933 Mbit/s	2014	5 GHz
Wi-Fi 4 (802.11n)	72 to 600 Mbit/s	2008	2.4/5 GHz
802.11g	6 to 54 Mbit/s	2003	2.4 GHz
802.11a	6 to 54 Mbit/s	1999	5 GHz
802.11b	1 to 11 Mbit/s	1999	2.4 GHz
802.11	1 to 2 Mbit/s	1997	2.4 GHz

Figure 4: The generations of Wi-Fi [7]

Using packets, WiFi stations communicate with one another. Wi-Fi 4, 5, and 6 have multiple carriers operating at marginally different frequencies inside a channel (OFDM), and these channels can be joined to create larger channels for greater throughput. Wi-Fi is already in the house, so this system can re-use the existing Wi-Fi network,

3.5. Hardware and software platform

For IoT applications, there are a plethora of micro-controllers on the market. Each of them has its advantages, so choosing the right micro-controller for this system must be influenced by considering specific design criteria, including the availability of a mature, free programming language and development environment, as well as active and helpful community of developers. Additionally, the chosen controllers must be able to run the communication software in every layer of a chosen protocol stack: there must be sufficient processing power, and the relevant protocol library must be available.

3.5.1. ESP8266 board

Arduino platform is convenient, cheap, reliable, and readily available. Due to its sizable support community, vast support library, and add-on boards that increase its interface capabilities, Arduino is appealing. It can use official products or third-party circuit boards. After having considered a number of different board types from the Arduino family, the author decided to use the ESP8266 series board produced by Espressif Systems.

4. Design and implementation

This section presents the basic model for the operation of this home automation system. A button and two LED lights are installed on the ESP8266board. These devices will communicate via Wi-Fi, using the communication module built into the ESP8266 board. In this chapter the author will introduce the suggested Home Automation system's fundamental concept: the use of MQTT topics.

4.1. Topics in MQTT

In MQTT, the UTF-8 string that the broker filters for each connected client is referred to as subject(Fig. 5). A topic can be broken down into one or more sections, each of which has a level of subject inside it. There is a / between each topic level. For example: level 1/ level 2/ level 3.

A single-level wildcard can represent (match) any text of one level of a topic. This wildcard is the + sign. For example: “home/room1/+/temperature” will match any topic, with level 1 = “home”, level 2 = “room1” , level 3 being anything, and level 4= “temperature”. In the previous example, the client that subscribed to the “home/room1/+/temperature” topic will receive all messages that were published (sent) with relevant topics, such as:

“home/room1/bed/temperature”
“home/room1/air control/temperature”
“home/room1/cabinet/temperature”

Multi-level wildcards cover multiple topics. A multi-level wildcard “#” must be used as the topic's final character and begin with / in order for the broker to know which subjects to match. After subscribing to “topic home/room1/#”, the client will receive all messages from room1. When designing communication logic, the core idea of the author is to realize both peer-to-peer communication and broadcast basing on particular topic patterns. At the same time, it is important to ensure that the length of the topic will not be too long.

4.2. Topic design

4.2.1. Communication patterns

There are four data communication situations (patterns) have been identified. They had to be mapped to four topic patterns:

Emergency communication
Broad-to-broad (room-to-other room) communication
Remote (external) control
Same-board (same-room) communication

An important principle of Topic design is to filter out unnecessary (irrelevant) topics. For example, a board that has subscribed to topic “home/room2/room1/#” will not receive “home/room1/room2/temperature” or “home/room1/room2/ humidity”. Note that the filtering is not done by the receiver client board. A broker will simply not sent (re-transmit) messages with a topic to a client board which has not subscribed.

In real situation, the message will not just contain a topic alone. It will also carry some load. So in the program the whole message may look like (“home/room1/room2/ humidity/msg”) or (“home/room1/room2/ humidity”, “msg”). The load msg can be char, string or int structure, to carry data. For example, msg=10. In this regard, the program will turn on the light when the first load equals to 10.

4.2.2. Emergency communication

In this case, no matter which sensor senses the occurrence of danger, a board to which this sensor is connected will send an alarm to all devices. For example: If sensor of room1 detects dramatic rising of temperature like a fire, it will send a message with a topic “home/emergency/room1/temperature”. It shows something happening in room1 urgently. It will be sent as a danger alarm message to every client, even to the remote device. Then other boards and a remote device subscribing to topic “home/emergency/#” will get the emergency message. So user can know there is a dramatic raising of temperature happening in any room (in this case-in room1).

4.2.3. Board-to-board communication

Another topic pattern is meaning for board-to-board communication. This situation is also the most commonly used function of this system. In this topic, the second section is source, and the third sector is destination. For example, the topic “home/room1/room2/temperature/on” means user in room1 wants to turn on the air-conditioning in room2. The load “on” (msg) of the message with the topic: “home/room1/room2/temperature” is a report message coming from room1. For actuators, the author puts them in the ending part of the topic. The translation from a name of an actuator to a pin number, to which this actuator is attached, will have to be performed by the program located on the destination

board.

4.2.4. Remote control

The third topic pattern describes remote control. It is assumed that for remote (external) control, old-fashioned GSM phones will be used, basing on SMS services. This requires the author to design a set of message texts that would serve the same purposes as board-to-board communication topics. In these versions of a topic, the addressing pattern “room1/room2” is retained, but “room1” is changed to “phone” representing the phone to send data to the board in room2. For instance, “home/phone/room2/temperature/off” means the topic will ask the broad to turn off the air-conditioning. The topic “home/room2/phone/temperature” with the data of temperature is sent form the room2 to the phone.

4.2.5. Same-board communication

As a special case, a user in room1 may want to control the appliances in the same room1. In that case, topic “home/+/room1/#” is subscribed by the client in room1.That means that messages with the destination room1 are sent to the board in room1.Then the appearance of "home/room1/room1/humidity" allows room1 not only to display humidity data by itself, but also to facilitate other users to observe its value.

	phone	Room1	Room2
subscribe	home/emergency/# home/+/phone/#	home/emergency/# home/+/room1/#	home/emergency/# home/+/room2/#
publish	home/phone/room1/temperature/on home/phone/room1/temperature/off home/phone/room1/humidity/on home/phone/room1/humidity/off home/phone/room2/temperature/on home/phone/room2/temperature/off home/phone/room2/humidity/on home/phone/room2/humidity/off	home/room1/room1/temperature/on home/room1/room1/temperature/off home/room1/room1/humidity/on home/room1/room1/humidity/off home/room1/room2/temperature/on home/room1/room2/temperature/off home/room1/room2/humidity/on home/room1/room2/humidity/off home/room1/room1/temperature home/room1/room1/humidity home/emergency/room1/temperature home/emergency/room1/humidity	home/room2/room1/temperature/on home/room2/room1/temperature/off home/room2/room1/humidity/on home/room2/room1/humidity/off home/room2/room2/temperature/on home/room2/room2/temperature/off home/room2/room2/humidity/on home/room2/room2/humidity/off home/room2/room2/temperature home/room2/room2/humidity home/emergency/room2/temperature home/emergency/room2/humidity

Figure 5: Some topics example of MQTT

4.3. The MQTT broker for the system

4.3.1. uMQTT Broker

uMQTT Broker is a MQTT Broker library for ESP8266 Arduino [8]. MQTT protocol version v3.1 and v3.1.1 are supported. It can support 8 clients at the same time, it has built-in buffer, and can retain messages. QoS level 0 is always supported. It has two modes to choose from. One is to use the ESP8266 board as an access point, which will not communicate with the external network. The second is to connect to WIFI.

The home automation system considered in this thesis is designed to use in indoor conditions with existing Wi-Fi. This indicates the broker will be connected to WIFI (Fig.6).

```
bool WiFiAP = false;//I choose to connect WIFI.
```

Figure 6: Choosing Broker to connect to existing Wi-Fi network

In Fig. 7 to 9 it is shown how to install the Broker software on ESP8266 hardware, and how to initialize it.

```
#include <ESP8266WiFi.h>
#include "uMQTTBroker.h"

/*
 * Your WiFi config here
 */
char ssid[] = "HUPEI";     // your network SSID (name)
char pass[] = "jj123456"; // your network password
bool WiFiAP = false;      // Do yo want the ESP as AP?

/*
 * Custom broker class with overwritten callback functions
 */
class myMQTTBroker: public uMQTTBroker
{
public:
    virtual bool onConnect(IPAddress addr, uint16_t client_count) {
      Serial.println(addr.toString()+" connected");
      return true;
    }

    virtual void onDisconnect(IPAddress addr, String client_id) {
      Serial.println(addr.toString()+" ("+client_id+") disconnected");
    }

    virtual bool onAuth(String username, String password, String client_id) {
      Serial.println("Username/Password/ClientId: "+username+"/"+password+"/"+client_id);
      return true;
    }

    virtual void onData(String topic, const char *data, uint32_t length) { // create a memory to store data
      char data_str[length+1];
      os_memcpy(data_str, data, length);
      data_str[length] = '\0';

      Serial.println("received topic '"+topic+"' with data '"+(String)data_str+"'");
      //printClients();
    }

    // Sample for the usage of the client info methods

    virtual void printClients() { // collect clients' data
      for (int i = 0; i < getClientCount(); i++) {
        IPAddress addr;
        String client_id;
```

Figure 7: Using uMQTT Broker (1)

```
        getClientAddr(i, addr);
        getClientId(i, client_id);
        Serial.println("Client "+client_id+" on addr: "+addr.toString());
      }
    }
};

myMQTTBroker myBroker;

/*
 * WiFi init stuff
 */
void startWiFiClient()
{
  Serial.println("Connecting to "+(String)ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) { // test connect wifi successfully or not
    delay(500);
    Serial.print(".");
  }
  Serial.println("");

  Serial.println("WiFi connected");
  Serial.println("IP address: " + WiFi.localIP().toString());
}

void startWiFiAP()
{
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, pass);
  Serial.println("AP started");
  Serial.println("IP address: " + WiFi.softAPIP().toString());
}

void setup() // set up communication
{
  Serial.begin(115200);
  Serial.println();
  Serial.println();


  if (WiFiAP) // WIFI start working
    startWiFiAP();
```

Figure 8: Using uMQTT Broker (2)

```
  else
    startWiFiClient();


  Serial.println("Starting MQTT broker");//  broker start working
  myBroker.init();

/*
 * Subscribe to anything
 */
  myBroker.subscribe("#");
}

int counter = 0;

void loop()
{
/*
 * Publish the counter value as String
 */
  //myBroker.publish("house/room2/room1/BUTTON1", (String)counter++);

  // wait a second
  delay(1000);
}
```

Figure 9: Using uMQTT Broker (3)

4.3.2. The buffer of the broker

In the code, uMQTT broker has one buffer. Firstly, create a buffer. Then determine the length of the buffer, and place the received data at the initial position of the string. Then output each topic and the data it carries in turn. The mode of buffer is First In First Out (FIFO) (Fig.10), which means the first data sent into buffer it also the first sent out from the buffer.

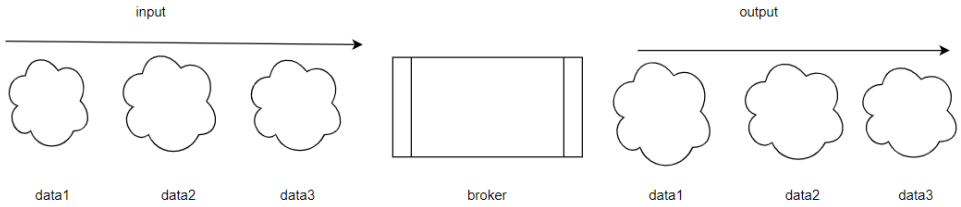

Figure 10: The idea of a buffer in a MQTT broker

4.4. The MQTT client for the system

For the MQTT client, the author chose to use the pubsubclient protocol on the ESP8266 board. Its description and available libraries can be accessed from [9]. This protocol can only subscribe to or broadcast Quality of Services (QoS) 0 or QoS 1 messages. The keepalive interval is usually set at 15 seconds. This QoS fully meets the requirements of the system for message quality. In the next part, the most simplified simulation model based on the pubsubclient protocol will be introduced.

4.5. Hardware considerations

4.5.1. Pins on ESP8266 board

The pins of ESP8266 can be found in Fig 11, GPIO16 (D0) and GPIO5 (D1) interfaces are respectively selected as LED (execution device) (Fig. 12) and button (command device) pins (Fig.13). Pressing the button may simulate the temperature change detected by the sensor, and lighting the LED may simulate the pulling of the electric curtain or infrared remote control command. Demonstration the basic communication patterns (in particular-those that involve MQTT operation) basing on the use of these two pins is sufficient as the proof-of-concept.

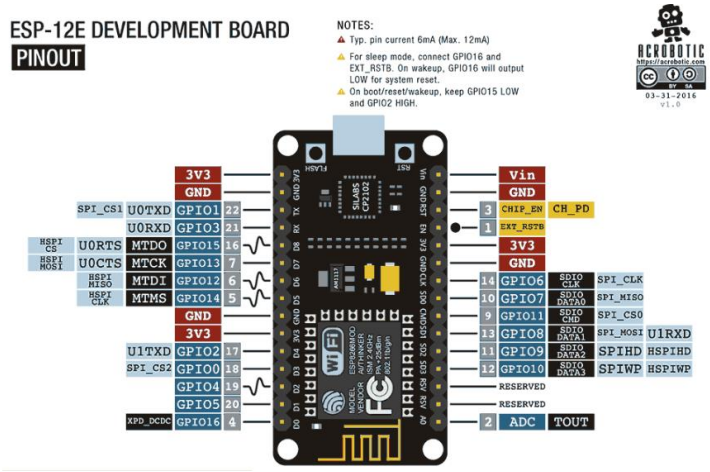

Figure 11: Pins of ESP8266 [10]

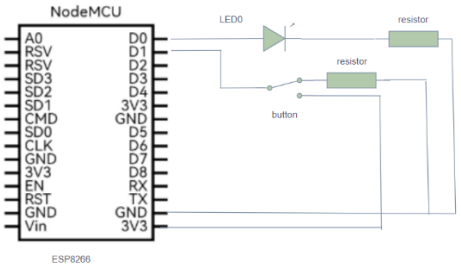

Figure 12: Circuit schematic for room 1's board

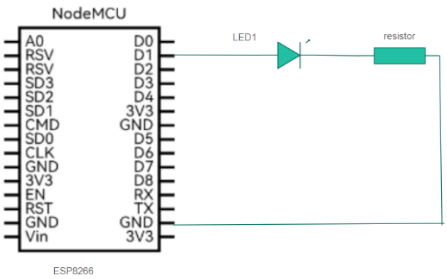

Figure 13: Circuit schematic for room 2's board

4.6. Software considerations

Buttons and LEDs are connected to the client boards. When the button is pressed in room1, the change in potential will be read by the board. Then the message will be sent in the form of (“house/room1/room2/BUTTON1”, msg). If both client room1 and client room2 will subscribe to this topic, then LEDs at both boards (LED0 and LED1, and the LED0 in code is shorten to LED) will be lit at the same time. When the LED (LED1) in client room2 is turned on, ("house/room2/room1/LED1", "on") will be sent to the broker. This whole process may simulate sending a signal in the bedroom to turn on the air conditioner in the bedroom and the light in the study. Of course, this is just example of system that needs to be explicitly programmed into the clients stations.

In all clients, it is first necessary to establish a callback function to create a buffer (Fig.14). Because the old payload buffer will be overwritten while creating the PUBLISH packet, it is vital to republish this payload.After that, it will keep the low potential according to the first value 0 (indicated by 48 in

ASCII code) in the payload. And when 1 is received (represented by 49 in ASCII code), the LED will be turned on to simulate the operation of the device. If the LED is on, room2 will send a message to room1 that the LED is on (Fig.15).

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");//only for debugging
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
   Serial.println();

  // Switch on the LED if an 1 was received as first character
  Serial.println();
  if ( (byte)payload[0] == 49) {
    digitalWrite(5,HIGH);//D1(gpio5),Pin for button
    Serial.println("turning the LED off");
  } else {
     digitalWrite(5, HIGH);
     Serial.println("turning the LED on");
  }

}
```

Figure 14: Code for buffer deciding to turn on or off

```
void loop() {

  if (!JIAOclient.connected()) {
    reconnect();
  }
  JIAOclient.loop();
    value=digitalRead(ledpin);
    delay(1000);
    if (value==1){
    //snprintf (msg, MSG_BUFFER_SIZE, "%ld",value);
    Serial.print("Publish message ");
    Serial.println("LED1 IS ON");
    JIAOclient.publish("house/room2/room1/LED1","on");
    }

}
```

Figure 15: Message “LED1” is on sent to room1

Determine whether to send information (Fig.16) to room2 by reading the potential state of the button (Fig.17). After reading the data, the data will be stored in the pre-set MSG_BUFFER_SIZE (a char to store messages), and the int will be converted to char for output.

```
void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  value=digitalRead(button);
  Serial.print("I have read button ");//only for debugging
    snprintf (msg, MSG_BUFFER_SIZE, "%ld", value);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("house/room1/room2/BUTTON1", msg);
    delay(1000);

}
```

Figure 16: Detecting the button pressed and sending a message

```
 value=digitalRead(button);
  if (value==1) {
  client.publish("house/room1/room2/BUTTON1","on");
}else{
   client.publish("house/room1/room2/BUTTON1","off");}
  }
```

Figure 17: Code to determine the status of the button

5. Experiments

5.1. Hardware experiment 1

Before starting to test the final model the author first tested the following model. The only difference between it and the final model is that there is only one button in board1 (Fig.18), and the structure of final design is exactly the same in board2 (Fig.19).

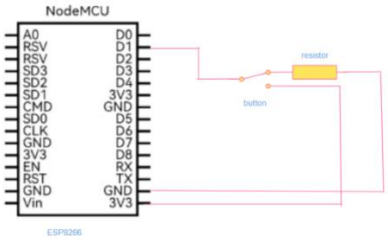

Figure 18: Circuit diagram for room1- initial experiment

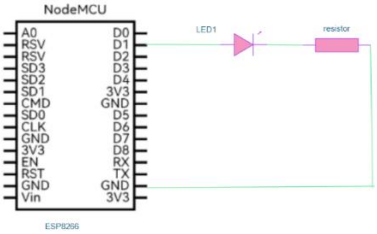

Figure 19: Circuit diagram for room2- initial experiment

Figure 20: Results for turning on the LED(1)

The significance of this model is that the author tried to change the MQTT load through the button first to verify whether the memory of the board is available, whether the MQTT communication model is compatible with the ESP8266, and the sensitivity of the button circuit before the complete model is completed. Figures 20 and 21 show the incorrect operation.

```
14:50:23.330 -> Message received[house/room1/room2/BUTTON1] BUTTON VALUE #1
14:50:23.330 -> Setting PIN off to HIGH
14:50:24.330 -> Publish message: LED VALUE #1
14:50:24.330 -> Message received[house/room1/room2/BUTTON1] BUTTON VALUE #1
14:50:24.330 -> Setting PIN off to HIGH
14:50:25.317 -> Publish message: LED VALUE #1
14:50:25.317 -> Message received[house/room1/room2/BUTTON1] BUTTON VALUE #1
14:50:25.317 -> Setting PIN off to HIGH
14:50:26.308 -> Publish message: LED VALUE #1
14:50:26.308 -> Message received[house/room1/room2/BUTTON1] BUTTON VALUE #1
14:50:26.348 -> Setting PIN off to HIGH
14:50:27.327 -> Publish message: LED VALUE #1
14:50:27.327 -> Message received[house/room1/room2/BUTTON1] BUTTON VALUE #1
```

Figure 21: Results for turning on the LED (2)

The reason for this is payload [0]. When converting int to char, the Button value and %Id are written together. Cause B to be the first payload, so no matter whether the received button data is 0 or 1, the LED will remain lit. After canceling the Button value and changing the confirmation value to ASCII code, the program will run as expected (Fig.22 to 25).

This shows the typical amount of debugging while writing the code for Arduino- type controllers.

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message received[");
  Serial.print(topic);
  Serial.print("] ");
  for (int i=0;i<length;i++) {
     Serial.print((char)payload[i]);
     Serial.println((byte)payload[0]);
  }
  Serial.println();
  if ( (byte)payload[0] == 48) {
    digitalWrite(5, LOW);
    Serial.println("Setting PIN on to LOW");
  } else {
     digitalWrite(5, HIGH);
     Serial.println("Setting PIN off to HIGH");
  }

}
```

Figure 22: Setting the first payload

```
15:46:17.677 -> Message received[house/room1/room2/BUTTON1] 048
15:46:17.677 ->
15:46:17.677 -> Setting PIN on to LOW
15:46:18.693 -> Publish message: 0
```

Figure 23: Receiving 0 turning off LED

```
15:52:00.263 -> Message received[house/room1/room2/BUTTON1] 149
15:52:00.263 ->
15:52:00.263 -> Setting PIN off to HIGH
15:52:01.246 -> Publish message: 1
```

Figure 24: Receiving 1 turning on LED

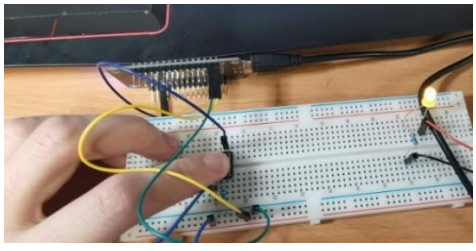

Figure 25: Turning on LED

5.2. Hardware experiment 2

In this model, the author also adds a small light to room1, and achieves simultaneous control by subscribing to the same topic as room2. Room2 will also send a “confirmation” message to the broker

that the LED is turned on. Figure 26 to 29 demonstrate the correct operation.

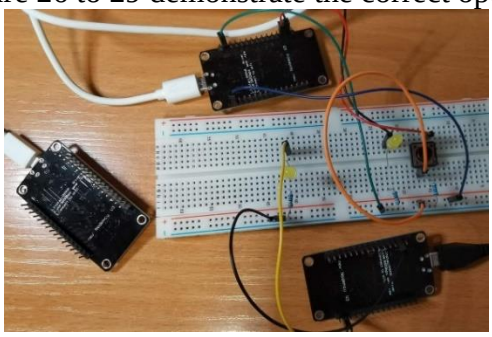

Figure 26: Total system with a Broker and clients

```
12:53:51.711 -> received topic 'house/room2/room1/LED1' with data 'LED VALUE'
12:53:52.267 -> received topic 'house/room1/room2/BUTTON1' with data '0'
```

Figure 27: Broker received message

```
12:52:46.604 -> Message received[house/room1/room2/BUTTON1] 149
12:52:46.644 ->
12:52:46.644 -> Setting PIN off to HIGH
12:52:47.642 -> Publish message LED1 IS ON
12:52:48.629 -> Publish message LED1 IS ON
12:52:48.629 -> Message received[house/room1/room2/BUTTON1] 048
12:52:48.629 ->
12:52:48.629 -> Setting PIN on to LOW
```

Figure 28: Room2 received message

```
12:53:01.169 -> Message received[house/room1/room2/BUTTON1] 149
12:53:01.169 ->
12:53:01.169 -> Setting PIN off to HIGH
```

Figure 29: Room1 received the same message

6. Future improvements and conclusion

Future improvements will add, for example, sensors to gauge the environment's temperature and humidity levels. Keypad and a small local display may be added to each client board, to allow user to operate the system easily and efficiently. Another ESP8266 board with a GSM shield will make a “phone” component, for remote operation via a dumb phone.

6.1. Sensor

The DHT11 (Fig.30) is a straightforward, low-cost digital temperature and humidity sensor. It measures the air around it using a thermistor and a capacitive humidity sensor (Fig.31), and it provides a digital signal on the data pin. In the Arduino library you can download a direct call and it will collect data every two seconds. This will be fairly simple to use on the fly.

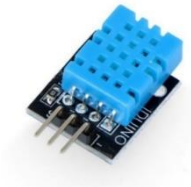

Figure 30: DHT11 a sensor with function collecting temperature and humidity

```
//#define DHTPIN=D4     // what pin we're connected to
const int  DHTPIN = 4;
#define humidity_topic "humidity"
#define temperature_topic "temperature"
```

Figure 31: Pin and topics for sensor

The data measured by the sensor will be sent to the board or remote phone. If the same data is recorded for each measurement, it will lead to a lot of wasted resources and wear down the user's patience. Therefore, the author sets the frequency of data transmission in the case of a sudden change (Fig.32). In the figure, when the two data have an integer gap of 5, the board with sensor will broadcast to every device. And when the second data is consistent with the first or when the difference is not reached, the second data is automatically discarded. When the difference is reached, the second data will automatically replace the first data (Fig. 33).

```
bool checkBound(float newValue, float prevValue, float maxDiff) {
    return newValue < prevValue - maxDiff || newValue > prevValue + maxDiff;
}

long lastMsg = 0;
float temp = 0.0;
float hum = 0.0;
float diff = 5.0;
```

Figure 32: Detecting temperature and humidity changes

```
    long now = millis();
    if (now - lastMsg > 30000) {
        // Wait a few seconds between measurements
        lastMsg = now;

        float newTemp = dht.readTemperature();
        float newHum = dht.readHumidity();
        if (checkBound(newTemp, temp, diff)) {
            temp = newTemp;
            Serial.print("New temperature:");
            Serial.println(String(temp).c_str());
            client.publish("house/room2/room1/temper", String(temp).c_str());
        }

        if (checkBound(newHum, hum, diff)) {
            hum = newHum;
            Serial.print("New humidity:");
            Serial.println(String(hum).c_str());
            client.publish("house/room2/room1/humidity", String(hum).c_str());
        }
    }
}
```

Figure 33: Handling temperature and humidity sensor

6.2. Keypad

One operation button for a system is far from enough. A button can only send the simplest command in the communication between two boards. So the keypad will be replaced with a 4*4 type (Fig.34).

Figure 34: 4*4 Keypad

The combination of sending programs will become quite simple with this improvement. For example, it becomes possible to validate user commands with a password. For this, after pressing five keys, the array char Data [Password_Length] will be compared with the array char room1 [Password_Length] "123A456" already written in the system. If both are equal, board in room1 will be authorized to send topic ("house/room1/room2/BUTTON1", msg) (Fig.35). After each comparison, the system will automatically eliminate the data in the array (Fig.36).

```
void loop() {
  // Get key value if pressed
  char customKey = customKeypad.getKey();

 if (customKey) {
    // Enter keypress into array and increment counter
    Data[data_count] = customKey;
    data_count++;
  }

  // See if we have reached the password length
  if (data_count == Password_Length - 1) {


    if (!strcmp(Data, room1)) {
    client.publish("house/room1/room2/BUTTON1", msg);
    }

    // Clear data
    clearData();
  }
}
```

Figure 35: Comparing button value

```
void clearData() {
  // Go through array and clear data
  while (data_count != 0) {
    Data[data_count--] = 0;
  }
  return;
}
```

Figure 36: Clean data

6.3. Devices improvement and I2C bus communication

In the design with two clients, all available pins of ESP8266 are occupied. This is not reasonable for a system that ends up with 8 clients, and it's not user friendly. The author plans to connect the display and keypad in series using the I2C bus (Fig.37), and improve the keypad.

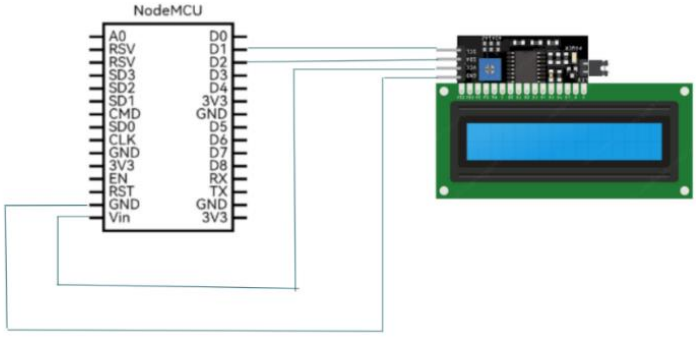

Figure 37: Display

6.3.1. Display

After the display with I2C protocol (Fig.38) [11] is added, the system can display the status of each LED and the connection with the broker. In the demonstration, when the light-on signal of room1 is received, the display connected to it will display: LED R1 ON. When the LED of room2 is lit, LED R2 ON will also be displayed (Fig.39). In the process of client and broker reconnection, BROKER DISCONNECT will be accompanied by every attempt (Fig. 40).

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x3F, 16, 2);
```

Figure 38: Library for I2C display

```
Serial.println();
if ( (byte)payload[0] == 48) {
  digitalWrite(5, LOW);
  Serial.println("Setting PIN on to LOW");
} else {
   digitalWrite(5, HIGH);
   Serial.println("Setting PIN off to HIGH");
   lcd.clear();
   lcd.setCursor (0,1);
   lcd.print("LED R1 ON");
}
if ( (byte)payload[0] == 50) {
  lcd.clear();
   lcd.setCursor (0,0);
   lcd.print("LED R2 ON");
}
}
```

Figure 39: Code for display

```
Serial.print("failed, rc=");
Serial.print(JIAOclient.state());
Serial.println(" try again in 5 seconds");
lcd.clear();
lcd.setCursor (0,2);
lcd.print("BROKER DISCONNECT");
// Wait 5 seconds before retrying
delay(5000);
```

Figure 40: Displaying disconnection

6.4. GSM shield

In principle, in order to control the house from a distance (not from any room station), an Internet connection and a smartphone of a laptop could be used. But, in this demonstrator system, another method is envisaged: using a GSM shield (Fig.41) connected to one distinguished board (named, e.g., “room 0” or “phone”), and this board would serve as a gateway, translating internal MQTT messages to external SMS messages, and in the other direction as well. This would allow simple non-smart GSM phones to be used to remotely control the house, without the need for establishing any external Internet connection. Any device with a SIM card in the 2G network can send and receive it. This design can ensure that both old fashion phone and smart phone can receive and send message meanwhile the structure and logic of topics are basically the same as those for local communication-within the house. For example, using the topic: “home/phone/room2/temperature/on", users can use the keyboard on the mobile phone to send information to the devices in the room.

Figure 41: GSM shied- an example [12]

6.5. Conclusion

The author of this thesis verified that with the help of IoT technologies: the cheap ESP8266 boards and software libraries for Wi-Fi communication and MQTT protocol, the Smart Home model can be efficiently built, without relying on a full-time internet connection. The model can be refined, as suggested here in recommendations for scaling up and optimizing the system. When increasing the number of clients and considering the need to measure more data, the system will connect other devices, such as monitors, keypads, and sensors, in the form of I2C. And the GSM shield will ensure that users can operate the system at a long-distance range through mobile phones or other communication tools.

References

[1] Tiago D. P. Mendes, "Smart Home Communication Technologies and Applications: Wireless Protocol Assessment for Home Area Network Resources". Available: https://www.mdpi.com/1996-1073/8/7/7279

[2] Hungryday,"Advantages and disadvantages of OSI reference model and TCP/IP reference model". Available: https://blog.csdn.net/Hungryday/article/details/38419899

[3] Patrick Th. Eugster, "The Many Faces of Publish/Subscribe". Available: http://systems.cs.columbia.edu/ds2-class/papers/eugster-pubsub.pdf

[4] Kate Brush, "MQTT (MQ Telemetry Transport)". Available: https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport

[5] Satyavrat,"IoT Protocols : An Overview", 5 Feb 2017. Available: https://community.element14.com/ technologies/ internet-ofthings/b/blog/posts/iot-protocols-an-overview

[6] "Understanding the IEEE 802.11 Standard for Wireless Networks". Available: https:// www.juniper.net /documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-80211.html

[7] "List of Wi-Fi Generations". Available: https://pl.wikipedia.org/wiki/Wi-Fi

[8] Jimmo,"uMQTT". Available:https://github.com/micropython/micropython-lib

[9] Knolleary. "pubsubclient protocol". Available: https://github.com/knolleary/pubsubclient

[10] "Pins of ESP8266". Available: https://acrobotic.com/.2016.03.16

[11] Joaopedrosgs, "Arduino-Liquid Crystal-I2C-library". Available: https://github.com/ fdebrabander/Arduino-Liquid Crystal-I2C-library

[12] "Arduino GSM Shield". Available: https://docs.arduino.cc/retired/shields/arduino-gsm-shield.