SAPIENZA
UNIVERSITÀ DI ROMA

Machine to machine technologies in Internet of things

A study of microcontrollers communication security and p2p capabilities

Microcontroller System Design exam - Final project

Luca Vargiu - 1422135

November 5, 2019

Contents

Chapter 1

Introduction

The increasing number of small and cheap devices with high and different sensing capabilities can be now integrated and used in different systems in order to improve their capabilities. These devices can have different implementations in scenarios where they can supply and manage different types of data and information.
The revolution in the infrastructural field of telecommunications that is taking place now through the introduction of the *fifth generation mobile network* (5G) will allow the use of a greater number of simultaneous connections to the same network cell with greater bandwidth[17]. In this way different types of devices like cameras, sensors and embedded systems acquire the ability to better communicate each other and provide information and services that contribute to the large distributed infrastructure we call the *Internet of Things* (IoT). Under these conditions, the use of safe and efficient protocols becomes a central and non-trivial need for the regulation of communication between the various devices.
This report will first focus on the main security aspects of the existing protocols in the IoT field of *Machine to machine* (M2M) technologies. Later we will introduce the exam project, describing its specification, implementations and test results. The project simulates a scenario where microcontrollers provide realtime information about their sensors through the usage of M2M technologies and cloud storage services.

1.1 IoT and M2M

Let's first of all introduce the communication technologies at the base of the project and in particular the definitions of IoT and M2M and what differentiate them.
IoT is defined as an international infrastructure for the society of information which allows dispensing sophisticated services by the physical and visual interconnection between things. This connection is based on the presence of mutual

communication and information technologies[9].
In other words, IoT is about embedded interconnections of various electronic devices, merging wireless connectivity and smart sensors technologies. Sensors and other electronic devices are integrated so that they connect to the internet through wired or wireless networks. It grants the ability to gather, analyze, store and distribute a huge amount of data that can be later transformed into information, intellect and knowledge[16].
The main goal of IoT is to provide a convergence between the smart connectivity of the involved devices and their context in order to obtain a *context-aware computation*.
An entity's context, as described here [2], can be used to improve efficiency, optimize resources and detect anomalies during data gathering and processing. Maintaining a valid information context with a huge amount of data remains still an open challenge since a unique context representation isn't sufficient to cover the knowledge behind all data in a IoT network where often context representations have an high rank of diversity[3].
M2M is a technology which allows both wireless and wired systems to communicate each other with devices of the same type. It is designed for general communications, withouth focusing on any specific wireless or wired networking, information and communications technology[16].

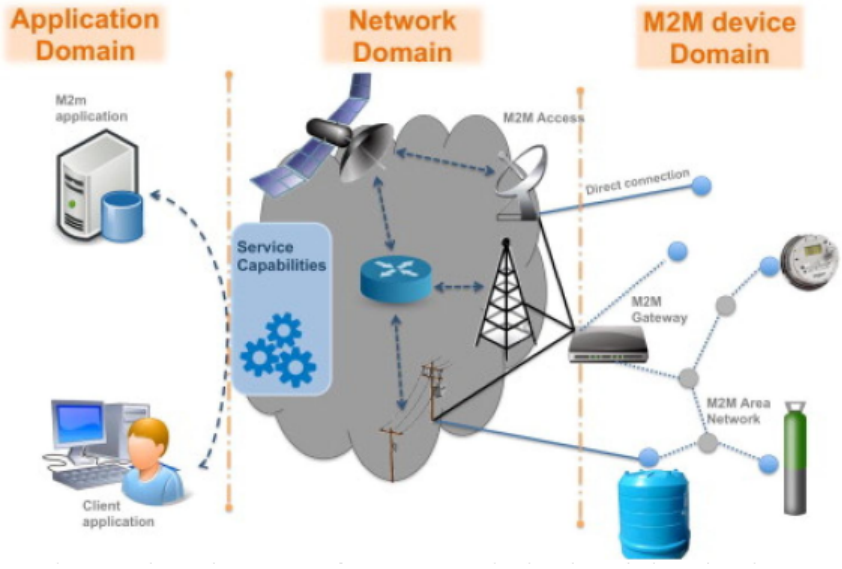


Figure 1.1: M2M concept of communication through three domains. Source: [9].

M2M communication is based on four main phases:

1. Collection of data;

2. Transmission of the data through the communication medium/network;
3. Assessment of the data collected;
4. Response to the machine based on the assessment.

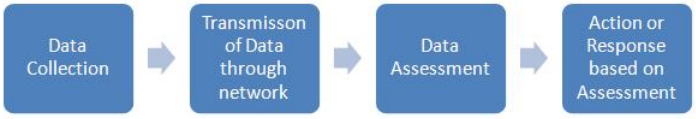


Figure 1.2: Four phases of M2M. Source: [22].

M2M interaction can so be basically summarized as a communication between a machine and a device with a remote computer. The information flows as the following:

1. M2M connects the device to the cloud;
2. it manages the device;
3. it collects machine and/or sensor data.

The main difference between IoT and M2M comes from the way data are transmitted and received[16]. IoT comes as the more general approach of communication system, where each device communicates through IP (IPv4/IPv6) networks. In this way M2M can be considered as a subset of IoT, restricting the devices that can communicate each other to the ones of the same type.
M2M is often related to heavy engineering and to business solutions, while IoT is often used in electronics consumer markets. Usage of both architectures is also supported due to the relationship that exists between both the solutions, allowing the usage of IoT with the entire physical infrastructure closely coupled to the information and communication technologies of M2M.

1.2 M2M protocols

First of all we can categorize the protocols of M2M communication in three major groups, as also suggested in this article[8]:

- *Service-oriented architectures* (SOA):
  consisting in the definition of a service as a discrete unit of functionality that can be accessed remotely, acted upon and updated independently. An example of SOA protocol in M2M is *OPC Unified Architecture* (OPC UA) for checking performance evaluation.
- *Representational State Transfer* (REST):
  is an architecture style which defines constraints to the used components,

connectors and data elements. It allows an easy integration of sensors and actuators using such protocols into the existing Internet infrastructure, that is almost composed by REST services. *Constrained Application Protocol* (CoAP) is an example of this type of protocol.

- *Message oriented protocols*:
  an implementation design that supports the asynchronous data transfer between distributed systems. The protocols that belong to this category often use specialized message transfer agents which can buffer messages on their transmission path. The *Message Queuing Telemetry Transport* (MQTT) is a lightweight representative of this protocol category.

We can now introduce some of the main protocols at the base of M2M communication technology, describing their main features and comparing their performances and security concerns.

1.2.1 MQTT

Message Queue Telemetry Transport (MQTT) is an open source protocol designed to be simple, lightweight, and especially with designed features for embedded devices with limited battery, processor and/or memory resources. The header of the exchanged messages has a fixed length of two bytes, granting a small transport overhead while transmitting messages. This feature makes MQTT an interesting solution for unreliable networks with restricted resources, such as low bandwidth and high-latency[14].

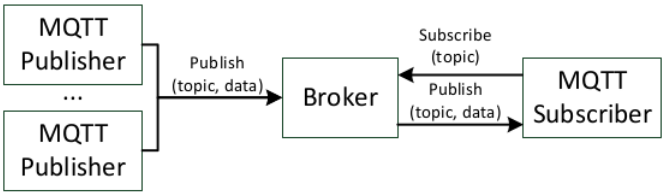


Figure 1.3: MQTT communication system. Source: [8].

The protocol is based on the *publisher/subscriber* model where the broker server acts as an intermediary for messages sent from a publisher client to subscriber clients, providing one-to-many message distribution and decoupling of use case application. Publisher's messages will be delivered via the server to the subscribers of a topic that is an additional information that marks each message and makes possible to classify and filter it by its content.
We can also highlight other types of messages supported by the protocol to guarantee different functions relating to the current network status or connected clients[10]:

- *Keep-Alive* message (PINGREQ, PINGRESP), where the broker can detect client disconnection even when it doesn't send explicit DISCONNECT messages;
- *Retain* message, where a PUBLISH message on a specific topic can be retained on the server allowing a new connected subscriber, on the same topic, to receive it;
- *Last Will* message, allows subscribed clients being informed about an unexpected client disconnection. It is possible to specify it in CONNECT message with topic, QoS, and retain. It keeps all subscriptions retained on the server when a client is disconnected, and allows them to be recovered on client reconnection.

The protocol in its standard version is developed to work on networks that support the TCP/IP protocol to send and routing messages.

MQTT-SN

Another version of the protocol, *MQTT for Sensor Networks* (MQTT-SN), has been introduced to overcome this restriction, supporting also UDP, 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) and customized radio frequency serial protocols under the IEEE standard 802.15.4.
Another main difference that comes from the standard is the simplification simplification in the messages exchanged among broker and clients, using "predefined" topic identifiers and short topic names in addition to small messages format [13]. MQTT-SN isn't fully standardized, having some compatibility issues with the standard MQTT implementation.

1.2.2 CoAP

The *Constrained Application Protocol* (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation with optimization for devices with constrained power and reduced processing capabilities[4].
At its core CoAP it's based over HTTP and the UDP transport protocol. It specifies a minimal subset of REST requests including POST, GET, PUT, and DELETE, supporting resource caching and built-in resource discovery.
CoAP adopts a *request/response* model, where each device acts as "client" or "server" and the resources can be accessed by URIs. Unlike the standard HTTP protocol, CoAP's connection is not established before message exchanging.
The communication happens in an asynchronous way. The supported types of messages are:

- CON (Confirmable),
- NON (Non-Confirmable),

- ACK (Acknowledgment),
- RESET.

In comparison to HTTP, CoAP results more cost-effective since the protocol performs less data exchange between client and server, resulting in lower power consumption when using cheaper equipment in both sides of the connection[18]. This is achieved by using a compact binary header[1] in combination with the UDP based transport that reduces the overhead data and consequently decreases the delay and minimizes battery usage during transmission.
The protocol supports the asynchronous information push that allows smart objects to send resource information only when there is a change. The device can stay in *sleep mode* most of the time, which means a reduction in the power consumption.
As a final remark, the usage of a minimal subset of REST requests lowers the hardware requirements to run the protocol, allowing also cheap devices to interact in a CoAP network[10].

1.2.3 OPC UA

OPC Unified Automation (OPC UA) is a platform-independent industrial middleware technology[8]. It allows interoperability between heterogeneous system components over various types of networks and defines methods for both data modeling and transport[8].
OPC UA is based on the *server/client* communication pattern. For transmission OPC UA currently defines two protocol mappings and two encodings. The data can be encoded generically in UA XML or more efficiently in UA Binary. For transport OPC UA can use common web standards like SOAP and HTTP which allow an easy crossing of firewalls. For resource constrained devices the payload can be directly integrated into TCP by the UA Native protocol. It is also possible to transport UA Binary by SOAP and HTTP.
Nevertheless, even the combination of UA Binary and UA Native causes a significant overhead as also underlined in this study[23].

1.3 Comparative analysis of M2M protocols in literature

In literature it is possible to identify several studies about comparative analysis of M2M protocols. In this paragraph we will analyze some of these papers and report back their results and conclusions.
In Frigieri, Mazzer and Parreira work[10] we can identify a comparison of MQTT, MQTT-SN and CoAP protocols in a constrained network where several sensors provide through an internet gateway a cloud service. The network where

[1] Composed by a 4 byte fixed protocol header, a token field with a maximum of 8 bytes and an option field with variable length.

data are exchanged is designed to be as similar as possible to a real use case, so with mixed unreliable connections that could be wired or wireless between the sensor nodes, which have different constrained resources[2] that may vary on a fixed range.

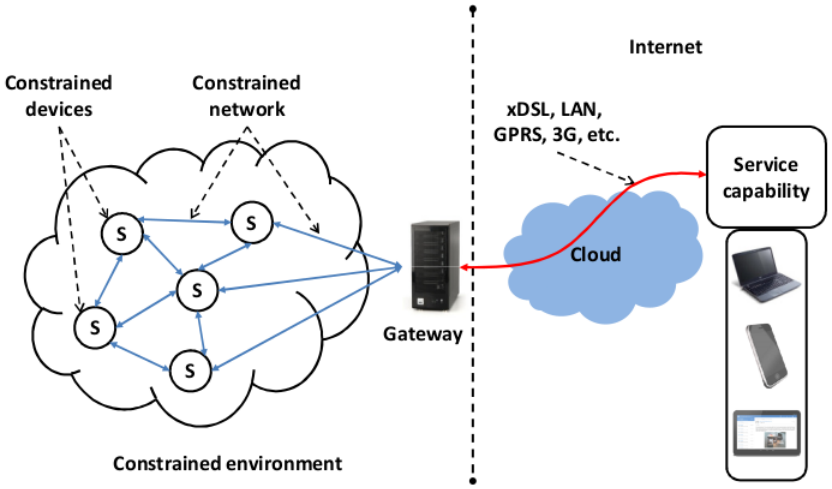


Figure 1.4: Study's test scenario. Source: [10].

We will report briefly below the results following the discussion already taken up in the document emphasizing the following aspects:

- *Implementation* (size cost):
  MQTT results to have a simpler protocol specification in comparison to CoAP, giving the it a small advantage with memory and processing capabilities limitations;

- *Data Transport* (transmission cost):
  MQTT employs connection oriented communication given by TCP, which is more costly than UDP used by CoAP especially when running over constrained networks where reducing the message overhead is central to limit packet fragmentation and also increase their delivery probability.
  On the other hand MQTT, with its small fixed-size header, becomes a suitable solution for networks with low transmission rate, and also a more general solution for the payload in comparison to CoAP because of its support of any type of data instead of only binary payloads;

- *Communication patterns*:
  the discussion focuses on the following patterns:

[2]Using 8-32 bits microcontrollers with 16-190 MHz clock rate, 8K-64M bytes RAM (DRAR, SRAM) memory, and 64K-1M bytes flash memory.

  - *Telemetry* pattern (status changes from device to the cloud):
    not suitable for CoAP because the connection must be started from system (client) to the device (server), which can confront addressing problems like mobile roaming or NAT. The MQTT publish/subscribe model matches with Telemetry pattern facilitating its application.
  - *Inquiries* pattern (requestes from device to the cloud for information):
    CoAP have better performance for this pattern since they are based on request/response model, while MQTT comes with the necessity of defining a response topic for communication since there is not a built in response path support which configures an implementation difficulty.
  - *Commands* pattern (commands from system to devices/sensors for specific activities):
    CoAP has the same problems that we saw under Telemetry pattern, while MQTT needs the definition of a result topic for working as an answer path.
  - *Notifications* pattern (data exchange from system to devices for handling status changes at physical level):
    the CoAP addressing problems persists, while, on the other hand, MQTT publish/subscribe model fits in the notification architecture having problems only if a better flow control is required for big amount of data at high data rates;

- *Reliability and QoS* (data delivery and/or duplication of packets avoidance):
  both MQTT and CoAP have QoS options that can be used depending on the desired data flow control, obviously with different implementations at any QoS level;

- *Scalability*:
  MQTT allows an easy *horizontal* scale because it is based on the publisher/subscribe model. The protocol allows decoupling in time and space since publisher and subscribers don't need to transmit data at the same time and subscribers don't need to know each other. Moreover, events can be produced or consumed in an asynchronous way allowing greater scalability and flexibility.
  In CoAP scalability is also possible but in a different mode since sensors and devices are considered as resources. It is implemented using the *Telemetry* interaction model described before and in particular using caches and intermediaries (proxy) nodes that multiplex the interest of multiple clients (subscribers) in the same event into a single association.
  As both protocols rely on broker to exchange messages, the system infrastructure can be easily scaled up if more bandwidth or processing power is provided;

- *Security*:
  MQTT protocol was not designed with security in mind and, as many other protocols based on TCP, it uses the Security Socket Layer (SSL) or Transport Layer Security (TLS) for security. But the username and password credentials are transmitted without any encryption, given rise to one of the security problems in the protocol[7]. However, since MQTT doesn't focus on a specific payload type, payload can be encrypted by other terms and directly sent, increasing so the overall security.
  CoAP uses the DTLS protocol (Datagram Transport Layer Security), which is based on TLS (Transport Layer Security) over UDP. DTLS maintains security problems as seen in TLS. Problems come on DTLS translation when CoAP mapping is used at a proxy level for providing end-to-end secure connection and on secure multicast communications that are not natively supported.

In the paper's conclusions, MQTT is shown to have a better accordance to the presented communication patterns besides a lightweight and simple implementation. On the other hand, CoAP can be applied considering each component as a resource that can be accessed through an URL, directing the choice of one of the two protocols in accordance of the implementation scenario.

	MQTT	MQTT-SN	CoAP
Network Protocol	TCP/IP	Not specified	UDP
Payload type	Binary	Binary	Binary
Suitable for microcontrollers	Yes	Yes	Yes
Security	SSL/TLS	Not specified	DTLS
Scalability	Simple	Simple	Complex
Network architecture	Broker based (publish/subscribe)	Broker based, client/server, client/client	Client/server (request/response)
Communication pattern	Topic based	Topic based	REST architecture
QoS options	Yes	Yes	Yes

Figure 1.5: Summary table of the results. Source: [10].

Another document[8] provides us a benchmark study conducted on MQTT, CoAP and OPC UA protocols. Tests were done emulating cellular networks GSM (2G), UMTS (3G) and LTE (4G) communications, taking measurements on the transmitted payloads several times.
We can briefly summarize the obtained results in the following lines:

- OPC UA resulted to have the lowest transmission time despite having has the largest overhead of all measured protocols. The result is explained as

a result of a "pipeline" strategy that occurs on sequential message sending. In particular the TCP acknowledgment for the second ReadResponse message is included in the ReadRequest frame of the following data transfer. MQTT and CoAP showed worse performances due to the way the connection is established with more messages needed between the actors. Moreover, in CoAP each packet needs to be also acknowledged whereas in OPC UA no explicitly acknowledgment is necessary.

- There are periodic spikes at short intervals in the transmission time that occurs using the TCP-based protocols OPC UA and MQTT over EDGE and UMTS, while not affecting LTE connections.

- In contrast to CoAP, the transmission times of MQTT and OPC UA over LTE remain largely constant. In LTE data transmitted over the air interface are divided into the so called transport blocks which length is determined by the LTE base station. If the length of one transport block is larger than the length of one IP packet sent by the mobile device, LTE concatenates the IP packets until the transport block size is reached. After the concatenation the data is transmitted over the air interface. This feature leads to the observed constant transmission time that occurs in the TCP-based protocols[3]. On the other hand, CoAP, that bases its functioning over UDP transport protocol, can't concatenate messages in LTE's transport blocks since UDP doesn't support transfer windows like TCP, so each block must be sent separately over the air interface.

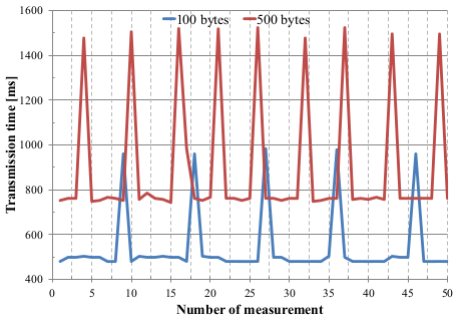


Figure 1.6: OPC UA, transmission time for different payload sizes. Source: [8].

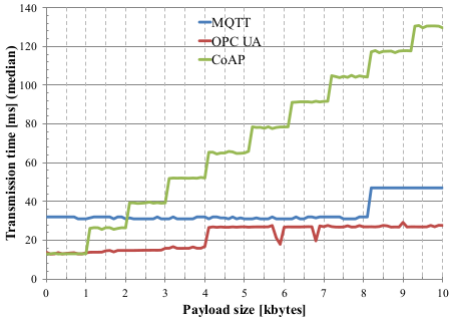


Figure 1.7: LTE comparison results. Source: [8].

The tests showed that OPC UA protocol provides the best performances while transferring data over cellular networks with special remarks on LTE, due to its design addressed towards cyclic data transfer. Another remarkable result comes form CoAP that proved itself to be in its implementation not suitable for the transmission of large payloads over cellular networks.

[3]MQTT and OPC UA.

Chapter 2

Library requirements

Taking in consideration what we found and discussed in the previous chapter about the three main suitable M2M protocols for the proposed project (*MQTT*, *CoAP* and *OPC UA*), we now proceed to provide some information about the availability of libraries for the Arduino platform and about the possibility of peer-to-peer(P2P) communication support.

2.1 MQTT

On the Arduino platform we can identify different libraries and wrappers designed for easily implementing the protocol for the system. In particular *LwMQTT*[12] and *Arduino-MQTT*[11] (that also bundles the first one) are the official open-source libraries for the framework. Other solutions are more hardware specific and provide a direct implementation with IoT services in order to easily manage online the topics generated by the protocol. Examples of these libraries are *Adafruit-MQTT-Library*[1], *Cayenne-MQTT*[5] and *Losant-MQTT-Arduino*[19].

2.1.1 P2P support

As described in the introduction, MQTT bases its communication over a publisher/subscriber model with a broker server, not allowing so a direct P2P communication between nodes by design. This restriction is also underlined in an article about the implementation of an Instant Messaging (IM) application based on the protocol[6]. However, a creation of a P2P channel is still possible by using the protocol with unique topics as a way to instantiate the communication over a different protocol on an higher level.

2.2 CoAP

CoAP protocol is available to Arduino via a simple library[21] that allows the device to configure itself as server or client when necessary.

2.2.1 P2P support

A P2P communication can be setup very easily with this protocol by correctly configuring each node both as a server and client and by giving with the REST responses information about the URIs of the resources that are implied in the communication. An interesting implementation of the protocol in this way can be found in a study[15] in which we can also underline the usage of the RELOAD (*REsource LOcation And Discovery*)[1] protocol for interconnecting nodes from local networks with other main nodes. This implementation in particular overcomes some CoAP limitations regarding web linking problems related to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient.

2.3 OPC UA

We can identify only one official library for Arduino that supports the protocol[20].

2.3.1 P2P support

There hasn't been found any documentation about P2P applications of the protocol. However since client/server communication model is the core of the protocol's transactions, we can suppose that a P2P communication may be setup between nodes in a similar way as we saw with CoAP.

[1] A P2P signaling protocol that is being specified by the Peer-to-Peer Session Initiation Protocol;

Chapter 3

The project application

It has been chosen to propose a CoAP implementation as an example of a M2M P2P protocol architecture. This choice is reinforced by what was reported in the previous chapters:

- MQTT despite being under some technical aspects a more efficient protocol when compared to CoAP requires more tuning when we need direct communication between several microcontrollers by defining unique topics for each required channel. In scalability terms it may end up in several requests to the broker server with *one-to-one publish/subscribe* communications performed in an inefficient way. On the other hand CoAP requires only the implementation of a simple REST request while maintaining callbacks as a server for its provided resources. The response of a "client" request can be simply identified by checking the IP address of the sender (in case of unique resources on the remote server) or by checking the unique 8-bit token of the message that can be provided while forging the header of the initial REST request[1];
- OPC UA was also an interesting implementation choice but unlike CoAP there haven't been found any documentation about P2P examples nor actual hardware compatibility with the WiFi module that has been used in the project.

[1]The protocol requires that the server that receives a CoAP request maintains the same token also in its response as we can read in the protocol specifications: `https://tools.ietf.org/html/rfc7252`

3.1 System Architecture

The hardware structure implemented in the project can be summarized in the following scheme:

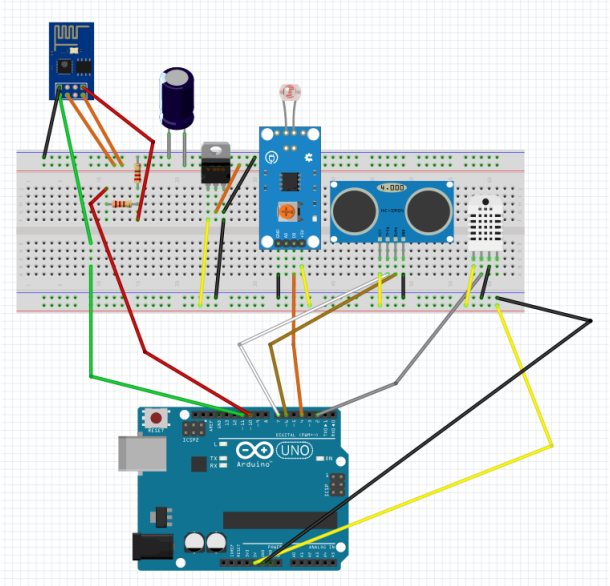


Figure 3.1: Project's circuit connection scheme.

The hardware modules used are the following[2]:

- 1x *Arduino Uno RV3*;
- 1x *ESP8266-01* (ESP-01) WiFi module;
- 1x *DHT22* temperature and humidity sensor;
- 1x Ultrasonic sensor *HC-SR04*;
- 1x Light Level Sensor (*LDR*) - *LM393*;
- 1x *AMS1117* 3,3 voltage regulator.

Each sensor module is connected to the Arduino board with standard links that will be briefly explained in the software section. The ESP-01 connections instead require a more detailed description that will be provided in the next subsection.

3.1.1 About the ESP-01 setup

The ESP-01 WiFi module is a microcontroller with WiFi capabilities. As also defined in its data sheet, the module works at a voltage of 3.3 V on all its pins

[2]The data sheets of the components are provided in a short manner (a maximum of two pages) in the Appendixes chapter of this document with also a link reference of the source.

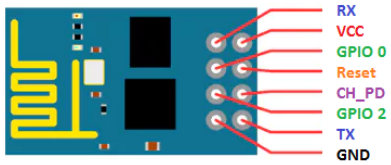


Figure 3.2: ESP-01 pin configuration.

admitting a maximum of 3.6 V to work in a proper way. It has been chosen to supply the energy to the circuit by reducing the voltage through an AMS1117 regulator despite using the 3.3 V pin on the Arduino Uno board in order to avoid possible voltage drops that may arise. There has been taken also introduced a countermeasure for possible voltage peaks on the corresponding breadboard side by introducing a 100 µF capacitator between the resulting voltage and the ground.
Another precaution has been taken for the RX pin of the module that receives data from the Arduino board. Since the Arduino output pin voltage is about 5 V, a voltage divider circuit has been created in order to obtain the required 3.3 V by introducing two resistances of 1000 Ω[3] and 2000 Ω[4].

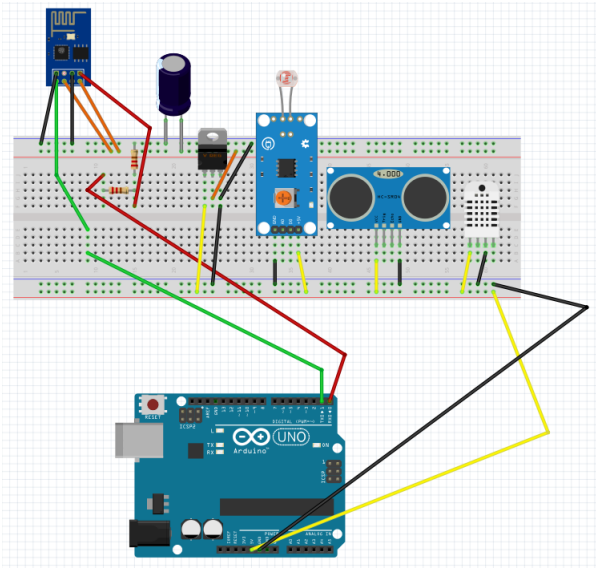


Figure 3.3: ESP-01 upload code circuit scheme.

[3]Between the Arduino's connection line and the ESP-01's RX pin.
[4]Between the Arduino's pin line and the ground.

We can report a final remark about how the board has been programmed since the used circuit is a little bit different. The module can be programmed in different ways by flashing a new firmware on it. In this project the Arduino code provided through the Arduino IDE[5] has been directly injected on the ESP-01 by changing some of the wiring connections with the Arduino One Board[6]:

- the GPIO 0 pin, otherwise not connected, must be connected to the ground in order to inhibit any code routine on the microcontroller;
- ESP-01's RX and TX pins must be connected to Arduino's RX and TX counterparts always regulating the voltage of RX.

3.2 Software organization

Both the Arduino One and the ESP-01 microcontrollers have been programmed in order to act together while implementing the CoAP services and providing sensor data. Arduino IDE and the Arduino C++ code have been used to accomplish this goal.

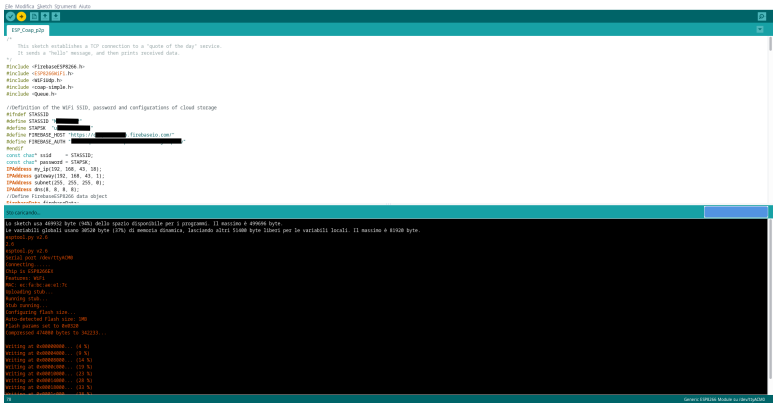


Figure 3.4: Arduino IDE during ESP-01 code upload.

In particular all the logic related to CoAP and Firebase communications resides on the ESP-01 while the sensor readings and their interpretation are implemented on Arduino. Despite having the possibility of coding everything on the WiFi module, this choice of implementation is strengthened by the restrictions of the device regarding the number of digital ports available (only GPIO 0 and GPIO 2) and the memory constraints, taking also in consideration that the project occupies the 94% of the memory available[7]. The modules intercommu-

[5] A brief guide on how to setup the IDE can be found here: `https://www.hackster.io/ROBINTHOMAS/programming-esp8266-esp-01-with-arduino-011389`

[6] During this operation a "barebone" empty sketch was loaded on the Arduino board.

[7] Information given by the IDE when the code was compiled.

nicate each other via the Serial ports[8] of the ESP-01 and the 10th and 11th pin of the Arduino Uno board.
In the following subsections we will introduce and comment the most crucial parts of the code that has been loaded on the modules[9].

3.2.1 Arduino Uno code

As described before the main purpose of the Arduino Uno module in this project is to read requests coming from the WiFi module through its Serial ports and give them the right responses. The communications are regulated by reading and writing on a *SoftwareSerial* object as shown in this code snippet:

```
1  ...
2  SoftwareSerial Esp_serial(pinTXesp,pinRXesp); // TX,RX
3  ...
4  void setup() {
5    ...
6    Esp_serial.begin(115200);
7  }
8
9  void loop() {
10    if(Esp_serial.available()){
11      String s=Esp_serial.readStringUntil('\n');
12      Serial.println(s);
13      if(s.charAt(0)=='0' || s.charAt(0)=='3' ||
            s.charAt(0)=='4'){
14        getDHT22Data(s);
15      } else if(s.charAt(0)=='1'){
16        getlLDR393Data();
17      } else if(s.charAt(0)=='2' || s.charAt(0)=='5'){
18        getHCS04Data(s);
19      }
20    }
21 }
```

As we can see, the correct callback for the sensor is simply identified by checking the input's first character. Any callback, after reading its sensor data, writes back on the "*Esp_serial*" channel the actual response.
Any message exchanged is fully sent with the "*println*" function, so that at the other end it can be easily read with the "*readStringUntil('\n')*" function called on the corresponding Serial reference.
At this point we can underline some implementation aspects related to the specific callbacks. Each callback returns a response to the requesting Serial

[8] TX and RX pins.
[9] The code is available in its full length in the second Appendix of this document and online at: https://github.com/bersa125/Coap-P2P-Implementation.

channel including the sensor data preceded by the char value of the request.

DHT22 sensor handling

```
#include <SimpleDHT.h>
...
//variables DHT-22
float temperature = 0;
float humidity = 0;
...
SimpleDHT22 dht22(pinDHT22);
...
void getDHT22Data(String type){//0 string, 3 temp, 4 hum
  float temperature_l = 0;
  float humidity_l = 0;
  int err = SimpleDHTErrSuccess;
  if ((err = dht22.read2(&temperature_l, &humidity_l,
      NULL)) == SimpleDHTErrSuccess) {
     temperature=temperature_l;
     humidity=humidity_l;
  }
  if(type.charAt(0)=='0'){
    Esp_serial.print("0");Esp_serial.print((float)temperature);
        Esp_serial.print(" *C,
        ");Esp_serial.print((float)humidity);
        Esp_serial.println(" RH%");
  }else if(type.charAt(0)=='3'){
    Esp_serial.print("3");Esp_serial.println((float)temperature);
  }else if(type.charAt(0)=='4'){
    Esp_serial.print("4");Esp_serial.println((float)humidity);
  }
}
...
```

The DHT22 sensor module requires an additional library to read data from the temperature and humidity sensors. There is the possibility that the read operation fails since the module requires some time delay in order to register new values. In order to cope with this situation temperature and humidity values are stored globally so that any call to the function will return the most recent valid value. The sensor function can provide different values based on the input.

HC-S04 sensor handling

```
...
```

```
//variables HC-SR04
long duration;
int distance;
...
void getHCS04Data(String type){
  // Clears the trigPin
  digitalWrite(trigHCPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigHCPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigHCPin, LOW);
  // Reads the echoPin, returns the sound wave travel
     time in microseconds
  duration = pulseIn(echoHCPin, HIGH);
  // Calculating the distance
  distance = duration*0.034/2;
  if(type.charAt(0)=='2'){
    Esp_serial.print("2");Esp_serial.print("Distance:
       ");Esp_serial.print(distance);
       Esp_serial.println(" cm");
  }else if(type.charAt(0)=='5'){
    Esp_serial.print("5");Esp_serial.println(distance);
  }
}

void setup() {
  pinMode(trigHCPin, OUTPUT);
  pinMode(echoHCPin, INPUT);
  ...
}
...
```

The Ultrasonic sensor HC-S04 integration takes place by handling signals over two different channels, Trigger and Echo. Passing a signal to the Trigger channel with a time delay of 10 µs makes possible to register on the Echo channel the time the signal needed to return back. At this point we can calculate the distance in centimeters of the nearest obstacle by multiplying the duration with the speed of sound (rounded 0.034 cm/s) and dividing by 2 since the sound travels the distance twice before returning to its origin. The sensor function can provide different values based on the input.

LDR-LM3939 sensor handling

```
...
```

```
//variable LDR
boolean light;
...
void getlLDR393Data(){
  light=digitalRead(ldrPin);
  if(light==HIGH){
    Esp_serial.print("1");Esp_serial.println("LOW Light");
  }else{
    Esp_serial.print("1");Esp_serial.println("HIGH
        Light");
  }
}
...
void setup() {
  pinMode(ldrPin, INPUT);
  ...
}
...
```

The LDR module has the simplest implementation, requiring only to read the input coming from its digital port. The module returns an HIGH signal if the light in the environment is under a certain threshold while returns LOW if there's sufficient light. The threshold is defined by a regulator on the module.

3.2.2 ESP-01 code

The ESP-01 WiFi module has been programmed to implement both CoAP client and server capabilities, to communicate with Arduino and another CoAP server to provide data about the exposed resources, and, with the activation of a special routine, upload constantly data of all its resources to the cloud (in particular to a Firebase Realtime Database).
First of all lets start by describing how the CoAP server and client are set up.

```
...
#include <WiFiUdp.h>
#include <coap-simple.h>
...
//Definition of coap classes
WiFiUDP udp;
Coap coap(udp);
...
void setup() {
  ...
  // Init of coap callbacks for the server
  coap.server(callback_wn, ".well-known/core");
  coap.server(callback_loop_update, "loop_update");
```

```
  coap.server(callback_time, "time");
  coap.server(callback_hcsr04, "hcsr04");
  coap.server(callback_dht22, "dht22");
  coap.server(callback_lm393,"lm393");
  coap.response(callback_response);
  coap.start(my_port);
}
void loop() {
  ...
  coap.loop();//Coap checks
  ...
}
```

The CoAP server and client have been set up using the CoAP-simple-library by initializing the Coap class object with WiFiUDP object handler. In this way UDP packages' creation and management can be easily implemented in the code by only calling the library functions. In the setup function server resources are defined giving a function callback reference and a string definition for each element that we want to expose to possible clients to the "*server*" function of the Coap class object. In particular in the code we can identify "*.well-known/core*", a particular resource required by the standard that returns all the available elements on the node, a resource for each "local" sensor, the resource "*time*", a reference to a remote resource that resides on a different CoAP node and that is requested directly to it by the server (the actual P2P implementation), and finally "*loop_update*", a particular resource that returns to the requester its actual state (ON or OFF) and allows through a PUT CoAP request to control a routine that with a minimum delay uploads the values of the other resources on the Cloud. Here a *response callback* is also defined in order to catch the responses that come after a CoAP request is sent by the server (The function covers all the possible incoming response, but in our case refers only to the "time" resource). After starting the server on a defined port, the "*loop*" function allows the periodically check if any incoming CoAP request or response has reached the server, activating the callback routines.

Local sensor callbacks

The callbacks for the three sensors have all been implemented following the same pattern as it follows:

```
...
void callback_dht22(CoapPacket &packet, IPAddress ip,
    int port) {//0 request to arduino for the DHT22 data
  serialFlush();
  Serial.println("0");
  delay(200);
  if(Serial.available()){
```

```
    String result=Serial.readStringUntil('\n');
    String s=result.substring(1);
    char data[s.length() + 1];
    s.toCharArray(data, s.length());
    data[s.length()] = '\0';
    if(result.charAt(0)=='0')
      coap.sendResponse(ip, port,packet.messageid, data);
    else
      coap.sendResponse(ip, port,
          packet.messageid,"Error");
  }else{
    coap.sendResponse(ip, port, packet.messageid,"Error");
  }
}
...
void serialFlush(){
  while(Serial.available() > 0) {
    String s=Serial.readStringUntil('\n');
  }
}
...
```

We can identify the counterpart of the "simple protocol" of Serial communication that we saw on Arduino:

1. Any previous message on the Serial buffer is deleted (A precaution for asynchronous routines between the microcontrollers);
2. Sensor data are requested via a "println" call on the Serial channel by their ID;
3. the result is read from the Serial buffer if available within a defined delay;
4. the message is prepared for the requester and sent only if it matches the sensor resource ID;
5. if any control fails an error message is sent back to the requester.

The message is sent back directly with the "*sendResponse*" function that takes as input data of the requester available in the inputs of the callback and the actual response message.

Remote resource callback and the P2P request

```
...
#include <Queue.h>
...
```

```
4  //Class used to define a queue of requests on remote data
5  class request{
6    public:
7    IPAddress ip;
8    int port;
9    uint8_t messageid;
10 };
11 DataQueue<request> requestsQueue(10);//Queue of pending
       requests up to 10
12 ...
13 void callback_time(CoapPacket &packet, IPAddress ip, int
       port) {
14    request req;
15    req.ip=ip;
16    req.port=port;
17    req.messageid=packet.messageid;
18    requestsQueue.enqueue(req);
19    coap.get(coap_peer, peer_port, "time");
20 }
21
22 void callback_response(CoapPacket &packet, IPAddress ip,
       int port) { //In response of a request
23   //Retrieves payload
24   char p[packet.payloadlen + 1];
25   memcpy(p, packet.payload, packet.payloadlen);
26   p[packet.payloadlen] = '\0';
27   String message(p);
28   while(!requestsQueue.isEmpty()){
29     request r=requestsQueue.dequeue();
30     coap.sendResponse(r.ip, r.port, r.messageid,p);
31   }
32   ...
33 }
34 ...
```

The "time" resource can be requested by any CoAP client to the coded server resulting in the request callback (*callback_time*) to make a CoAP request acting as a client to the peer server in order to query the real resource. During this operation some data of the original requester are saved in a buffer class object and then in a queue in order to be reused later. When a response from the CoAP peer comes and activates the response callback, then all the responses for that resource are sent back to the respective clients by emptying the request-queue. In this project in the response callback the message content is simply sent back, having only one peer CoAP server and only one remote resource. It is possible in a more complex situation to use the protocol tokens and different

data structures for buffering the requests (like dictionary implementations as an example) in order to recognize the remote resources.

Loop_update resource and Firebase upload

```
#include <FirebaseESP8266.h>
...
//Define FirebaseESP8266 data object
FirebaseData firebaseData;
String FPath = "/Coap";
...
//Timeouts for automatic sensor readings (deactivated by default)
unsigned long timer=0;
boolean timer_active=false;
int action=-1;
...
void callback_loop_update(CoapPacket &packet, IPAddress ip, int port){//Callback for /loop_update resource
  if(packet.code==3){//PUT request
    // extract payload
    char p[packet.payloadlen + 1];
    memcpy(p, packet.payload, packet.payloadlen);
    p[packet.payloadlen] = '\0';
    String message(p);

    if (message.equals("0")){
      timer_active=false;
    }else if(message.equals("1")){
      timer_active=true;
    }

  }
  if(timer_active){
    coap.sendResponse(ip, port,packet.messageid, "ON");
  }else{
    coap.sendResponse(ip, port,packet.messageid, "OFF");
  }

}
...
void loop_update(int i){//Function called in loop that when active sends calls to the other CoAP servers and data to Firebase
  if(i>=0){
```

```
37    serialFlush();
38    Serial.println(i);
39    delay(200);
40    if(Serial.available()){
41      String result=Serial.readStringUntil('\n');
42      String s=result.substring(1);
43      char data[s.length() + 1];
44      s.toCharArray(data, s.length());
45      data[s.length()] = '\0';
46      if(i==0){
47        if(result.charAt(0)=='0')
48          Firebase.setString(firebaseData,FPath+"/dht22/data/string",
                String(data));
49        ...
50      }else if(i==1){
51        if(result.charAt(0)=='1')
52          Firebase.setString(firebaseData,FPath+"/lm393/data",
                String(data));
53      }else{
54        if(result.charAt(0)=='2')
55          Firebase.setString(firebaseData,
                FPath+"/hcsr04/data/string", String(data));
56        ...
57      }
58    }
59  }else{
60    coap.get(coap_peer, peer_port, "time");
61  }
62 }
63 void callback_response(CoapPacket &packet, IPAddress ip,
      int port) { //In response of a request
64  ...
65  if(timer_active){//send to Firebase
66    Firebase.setString(firebaseData,FPath+"/time/data",
        message);
67  }
68 }
69 ...
70 void loop(){
71  ...
72  if(timer_active){//Update function
73    if(millis()>timer){
74      timer=millis()+300;//Timer
75      loop_update(action);
76      action+=1;
77      if(action>2){
```

```
78          action=-1;
79        }
80      }
81    }
82  }
```

The callback for the *loop_update* function is very standard and on any CoAP request different from the PUT one only returns as response message the state of a global variable that defines if the "upload to the cloud" routine is active or not. When a PUT routine activates the callback with the correct payload that value is updated having effectes on what is executed on the loop routine. In particular with a minimum delay each "*real*" resource is queried and the actual value is sent to the Firebase Realtime Database of the project. The connection to the cloud database is mediated through the *FirebaseESP8266* library, sending updating requests to values previously defined for the resources on the online database.

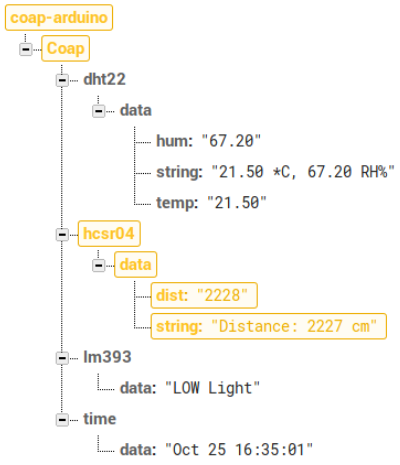


Figure 3.5: View of the Firebase database during data upload.

3.3 Test setup

All the project architecture has been tested on a real WiFi network using a PC with the libcoap[10] as second server and client tester. In particular, using two terminals a CoAP peer server (the one that exposes the actual "time" resource) has been setup and different requests have been called to the microcontrollers. Below there's a brief list of the used commands:

```
#In the test the machine had 192.168.43.14 on the WiFi network
#Launch the server on the selected port with max verbosity on
    the terminal
coap-server -v 9 -p 11111
#Query the server for the exposed resources
coap-client -m get coap://192.168.43.18:5865/.well-known/core
#Get a message containing data of the searched resource
coap-client -m get coap://192.168.43.18:5865/{resource}
#Activate or deactivate the loop update routine
coap-client -m put coap://192.168.43.18:5865/loop_update -e 1
coap-client -m put coap://192.168.43.18:5865/loop_update -e 0
```

[10] A C-implementation of CoAP server and client. Available at: `https://libcoap.net/`.

Chapter 4

Conclusions

The project proposed offers one of many possibilities of peer-to-peer implementations of machine-to-machine protocols with additional cloud integration, giving a simple way of creating a network of microcontrollers that can share data and make clients able to query them from an access node. It has been possible to check the performances of the server and in the majority of cases it managed all the incoming and exiting connections without a significant error rate or important delays even with the Firebase automatic update routine active. Only few messages on the Serial channel were lost due to the asynchronous computations that took place on the two modules. This aspect was also taken in consideration while coding in order to cope the availability of precise data with the reactivity in the response of the server.
Not any possibility has been explored due to the presence of only one microcontroller able to connect directly to the network and the usage of only two REST main requests (PUT and GET). As an example for a future project, more than one microcontroller may be configured to cooperate in a CoAP network over different physical networks exploiting in different ways the REST possibilities given by the protocol. In this way other aspects about security and performances would have the possibility to be investigated over a more expanded environment.
As a final note, it would be interesting to try something similar with the M2M protocol OPC UA since it shares the same communication pattern as CoAP[1], taking in consideration also the availability of libraries and its limits over the types of data that can be shared.

[1] The client/server pattern.

Chapter 5

Appendixes

5.1 Data sheets

In the next pages are reported the first three pages of each component data sheet. A numbered list of each document content is provided below:

1. Arduino Uno's Atmega 328P data sheet available at: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-Datasheet.pdf;
2. ESP8266-01's data sheet available at: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf;
3. DHT22 data sheet available at: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf;
4. HC-SR04 data sheet available at: https://www.mouser.com/datasheet/2/813/HCSR04-1022824.pdf;
5. LDR - LM393 data sheet available at: https://www.sunrom.com/get/885000;
6. AMS117 data sheet available at: http://www.advanced-monolithic.com/pdf/ds1117.pdf.

Atmel

ATmega328P

8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash

DATASHEET

Features

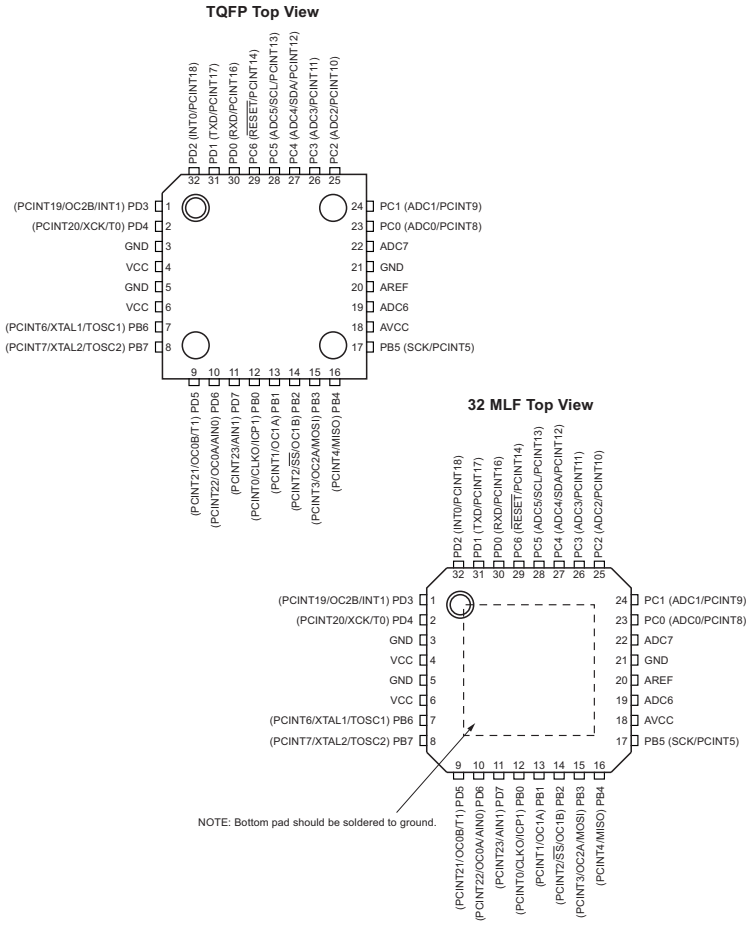
- High performance, low power AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single clock cycle execution
  - 32 × 8 general purpose working registers
  - Fully static operation
  - Up to 16MIPS throughput at 16MHz
  - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
  - 32K bytes of in-system self-programmable flash program memory
  - 1Kbytes EEPROM
  - 2Kbytes internal SRAM
  - Write/erase cycles: 10,000 flash/100,000 EEPROM
  - Optional boot code section with independent lock bits
    - In-system programming by on-chip boot program
    - True read-while-write operation
  - Programming lock for software security
- Peripheral features
  - Two 8-bit Timer/Counters with separate prescaler and compare mode
  - One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
  - Real time counter with separate oscillator
  - Six PWM channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature measurement
  - Programmable serial USART
  - Master/slave SPI serial interface
  - Byte-oriented 2-wire serial interface (Phillips I^2C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - Power-on reset and programmable brown-out detection
  - Internal calibrated oscillator
  - External and internal interrupt sources
  - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby

7810D-AVR-01/15

- I/O and packages
  - 23 programmable I/O lines
  - 32-lead TQFP, and 32-pad QFN/MLF
- Operating voltage:
  - 2.7V to 5.5V for ATmega328P
- Temperature range:
  - Automotive temperature range: –40°C to +125°C
- Speed grade:
  - 0 to 8MHz at 2.7 to 5.5V (automotive temperature range: –40°C to +125°C)
  - 0 to 16MHz at 4.5 to 5.5V (automotive temperature range: –40°C to +125°C)
- Low power consumption
  - Active mode: 1.5mA at 3V - 4MHz
  - Power-down mode: 1µA at 3V

1. Pin Configurations

Figure 1-1. Pinout



ESP8266EX

Datasheet

Version 6.2
Espressif Systems

About This Guide

This document introduces the specifications of ESP8266EX.

Release Notes

Date	Version	Release Notes
2015.12	V4.6	Updated Chapter 3.
2016.02	V4.7	Updated Section 3.6 and Section 4.1.
2016.04	V4.8	Updated Chapter 1.
2016.08	V4.9	Updated Chapter 1.
2016.11	V5.0	Added Appendix II “Learning Resources”.
2016.11	V5.1	Changed the power consumption during Deep-sleep from 10 μA to 20 μA in Table 5-2.
2016.11	V5.2	Changed the crystal frequency range from “26 MHz to 52 MHz” to “24 MHz to 52 MHz” in Section 3.3.
2016.12	V5.3	Changed the minimum working voltage from 3.0V to 2.5V.
2017.04	V5.4	Changed chip input and output impedance from 50Ω to 39+j6 Ω.
2017.10	V5.5	Updated Chapter 3 regarding the range of clock amplitude to 0.8 ~ 1.5V.
2017.11	V5.6	Updated VDDPST from 1.8V ~ 3.3V to 1.8V ~ 3.6V.
2017.11	V5.7	• Corrected a typo in the description of SDIO_DATA_0 in Table 2-1; • Added the testing conditions for the data in Table 5-2.

Date	Version	Release Notes
2018.02	V5.8	• Updated Wi-Fi protocols in Section 1.1; • Updated description of the integrated Tensilica processor in 3.1.
2018.09	V5.9	• Update document cover; • Added a note for Table 1-1; • Updated Wi-Fi key features in Section 1.1; • Updated description of the Wi-Fi function in 3.5; • Updated pin layout diagram; • Fixed a typo in Table 2-1; • Removed Section AHB and AHB module; • Restructured Section Power Management; • Fixed a typo in Section UART; • Removed description of transmission angle in Section IR Remote Control; • Other optimization (wording).
2018.11	V6.0	• Added an SPI pin in Table 4-2; • Updated the diagram of packing information.
2019.08	V6.1	Removed description of the GPIO function in Section 4.1.
2019.08	V6.2	Updated notes on CHIP_EN in Section 5.1

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

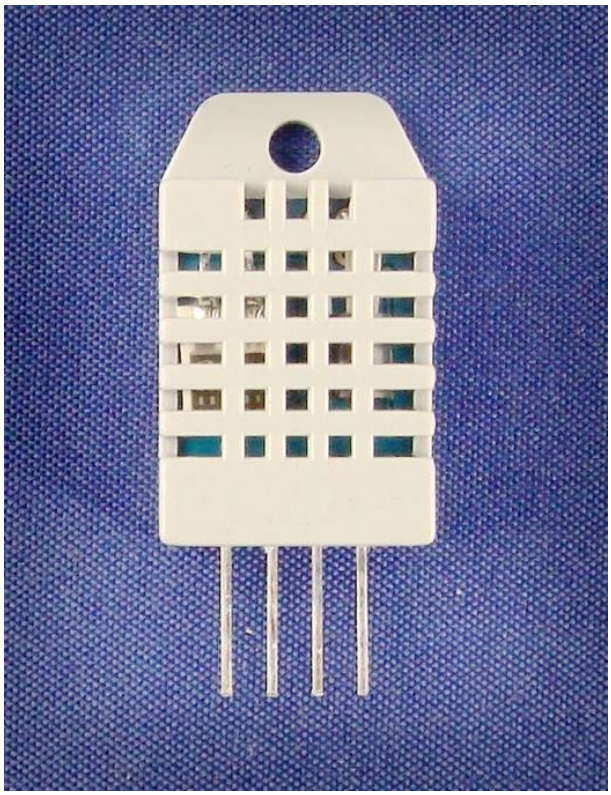
Download certificates for Espressif products from *https://www.espressif.com/en/certificates*.

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

Digital-output relative humidity & temperature sensor/module

DHT22 (DHT22 also named as AM2302)



Capacitive-type humidity and temperature module/sensor

1

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

1. Feature & Application:

* Full range temperature compensated * Relative humidity and temperature measurement
* Calibrated digital signal *Outstanding long-term stability *Extra components not needed
* Long transmission distance * Low power consumption *4 pins packaged and fully interchangeable

2. Description:

DHT22 output calibrated digital signal. It utilizes exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability.Its sensing elements is connected with 8-bit single-chip computer.

Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory.

Small size & low consumption & long transmission distance(20m) enable DHT22 to be suited in all kinds of harsh application occasions.

Single-row packaged with four pins, making the connection very convenient.

3. Technical Specification:

Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

4. Dimensions: (unit----mm)

1) Small size dimensions: (unit----mm)

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

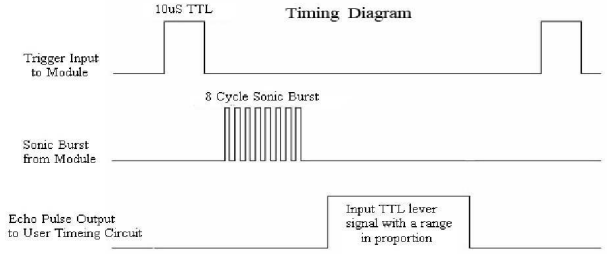
Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Vcc Trig Echo GND

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

www.Elecfreaks.com

LM193, LM293, LM393

Low power dual voltage comparators

Datasheet – production data

Features

- Wide single-supply voltage range or dual supplies: +2 V to +36 V or ±1 V to ±18 V
- Very low supply current (0.45 mA) independent of supply voltage (1 mW/comparator at +5 V)
- Low input bias current: 20 nA typ.
- Low input offset current: ±3 nA typ.
- Low input offset voltage: ±1 mV typ.
- Input common-mode voltage range includes ground
- Low output saturation voltage: 80 mV typ. (I_{sink} = 4 mA)
- Differential input voltage range equal to the supply voltage
- TTL, DTL, ECL, MOS, CMOS compatible outputs
- Available in DIP8, SO-8, TSSOP8, MiniSO-8, and DFN8 2 x 2 mm packages

Description

The LM193, LM293, and LM393 devices consist of two independent low voltage comparators designed specifically to operate from a single supply over a wide range of voltages. Operation from split power supplies is also possible.

These comparators also have a unique characteristic in that the input common-mode voltage range includes ground even though operated from a single power supply voltage.

DIP8
(plastic package)

SO-8
(plastic micropackage)

TSSOP8
(thin shrink small outline package)

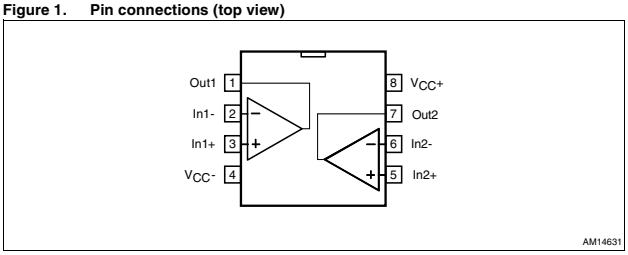
MiniSO-8
(plastic micropackage)

DFN8 2 x 2 mm
(plastic micropackage)

Contents

1 Pin connections

Figure 1. Pin connections (top view)



Advanced Monolithic Systems

AMS1117

1A LOW DROPOUT VOLTAGE REGULATOR

RoHs Compliant

FEATURES

- **Three Terminal Adjustable or Fixed Voltages* 1.5V, 1.8V, 2.5V, 2.85V, 3.3V and 5.0V**
- **Output Current of 1A**
- **Operates Down to 1V Dropout**
- **Line Regulation: 0.2% Max.**
- **Load Regulation: 0.4% Max.**
- **SOT-223, TO-252 and SO-8 package available**

APPLICATIONS

- **High Efficiency Linear Regulators**
- **Post Regulators for Switching Supplies**
- **5V to 3.3V Linear Regulator**
- **Battery Chargers**
- **Active SCSI Terminators**
- **Power Management for Notebook**
- **Battery Powered Instrumentation**

GENERAL DESCRIPTION

The AMS1117 series of adjustable and fixed voltage regulators are designed to provide up to1A output current and to operate down to 1V input-to-output differential. The dropout voltage of the device is guaranteed maximum 1.3V, decreasing at lower load currents.

On-chip trimming adjusts the reference voltage to 1.5%. Current limit is set to minimize the stress under overload conditions on both the regulator and power source circuitry.

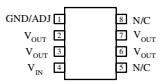
The AMS1117 devices are pin compatible with other three-terminal SCSI regulators and are offered in the low profile surface mount SOT-223 package, in the 8L SOIC package and in the TO-252 (DPAK) plastic package.

ORDERING INFORMATION:

PACKAGE TYPE			OPERATING JUNCTION
TO-252	SOT-223	8L SOIC	TEMPERATURE RANGE
AMS1117CD	AMS1117	AMS1117CS	-40 to 125° C
AMS1117CD-1.5	AMS1117-1.5	AMS1117CS-1.5	-40 to 125° C
AMS1117CD-1.8	AMS1117-1.8	AMS1117CS-1.8	-40 to 125° C
AMS1117CD-2.5	AMS1117-2.5	AMS1117CS-2.5	-40 to 125° C
AMS1117CD-2.85	AMS1117-2.85	AMS1117CS-2.85	-40 to 125° C
AMS1117CD-3.3	AMS1117-3.3	AMS1117CS-3.3	-40 to 125° C
AMS1117CD-5.0	AMS1117-5.0	AMS1117CS-5.0	-40 to 125° C

*For additional available fixed voltages contact factory.

8L SOIC Top View



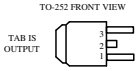
PIN CONNECTIONS

3 PIN FIXED/ADJUSTABLE VERSION

1- Ground/Adjust
2- V_{OUT}
3- V_{IN}

SOT-223 Top View

TO-252 FRONT VIEW



ABSOLUTE MAXIMUM RATINGS (Note 1)

Power Dissipation	Internally limited
Input Voltage	15V
Operating Junction Temperature :	
Control Section	-40°C to 125°C
Power Transistor	-40°C to 125°C
Storage temperature	- 65°C to +150°C

Soldering information	
Lead Temperature (25 sec)	265°C
Thermal Resistance	
SO-8 package	φ_{JA}= 160°C/W
TO-252 package	φ_{JA}= 80°C/W
SOT-223 package	φ_{JA}= 90°C/W*

* With package soldering to copper area over backside ground plane or internal power plane φ_{JA} can vary from 46°C/W to >90°C/W depending on mounting technique and the size of the copper area.

ELECTRICAL CHARACTERISTICS

Electrical Characteristics at I_{OUT} = 0 mA, and T_J = +25°C unless otherwise specified.

Parameter	Device	Conditions	Min	Typ	Max	Units
Reference Voltage (Note 2)	AMS1117	I_{OUT} = 10 mA 1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V	1.232 **1.2125**	1.250 **1.250**	1.268 **1.2875**	V V
Output Voltage (Note 2)	AMS1117-1.5	V_{IN} = 3V	1.478 **1.455**	1.500 **1.500**	1.522 **1.545**	V V
	AMS1117-1.8	V_{IN} = 3.3V	1.773 **1.746**	1.800 **1.800**	1.827 **1.854**	V V
	AMS1117-2.5	V_{IN} = 4V	2.463 **2.425**	2.500 **2.500**	2.537 **2.575**	V V
	AMS1117-2.85	V_{IN} = 4.35V	2.808 **2.7645**	2.850 **2.850**	2.892 **2.9355**	V V
	AMS1117-3.3	V_{IN} = 4.8V	3.251 **3.201**	3.300 **3.300**	3.349 **3.399**	V V
	AMS1117-5.0	V_{IN} = 6.5V	4.925 **4.850**	5.000 **5.000**	5.075 **5.150**	V V
Line Regulation	AMS1117	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.015 **0.035**	0.2 **0.2**	% %
	AMS1117-1.5	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.3 **0.6**	5 **6**	mV mV
	AMS1117-1.8	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.3 **0.6**	5 **6**	mV mV
	AMS1117-2.5	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.3 **0.6**	6 **6**	mV mV
	AMS1117-2.85	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.3 **0.6**	6 **6**	mV mV
	AMS1117-3.3	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.5 **1.0**	10 **10**	mV mV
	AMS1117-5.0	1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		0.5 **1.0**	10 **10**	mV mV
Load Regulation (Notes 2, 3)	AMS1117	(V_{IN} - V_{OUT}) =1.5V, 10mA ≤ I_{OUT} ≤ 0.8A		0.1 **0.2**	0.3 **0.4**	% %
	AMS1117-1.5	V_{IN} = 3V, 0 ≤ I_{OUT} ≤ 0.8A		3 **6**	10 **20**	mV mV
	AMS1117-1.8	V_{IN} = 3.3V, 0 ≤ I_{OUT} ≤ 0.8A		3 **6**	10 **20**	mV mV
	AMS1117-2.5	V_{IN} = 5V, 0 ≤ I_{OUT} ≤ 0.8A		3 **6**	12 **20**	mV mV

ELECTRICAL CHARACTERISTICS

Electrical Characteristics at I_{OUT} = 0 mA, and T_J = +25°C unless otherwise specified.

Parameter	Device	Conditions	Min	Typ	Max	Units
Load Regulation (Notes 2, 3)	AMS1117-2.85	V_{IN} = 4.35V, 0 ≤ I_{OUT} ≤ 0.8A		3 **6**	12 **20**	mV mV
	AMS1117-3.3	V_{IN} = 4.75V, 0 ≤ I_{OUT} ≤ 0.8A		3 **7**	15 **25**	mV mV
	AMS1117-5.0	V_{IN} = 6.5V, 0 ≤ I_{OUT} ≤ 0.8A		5 **10**	20 **35**	mV mV
Dropout Voltage (V_{IN} - V_{OUT})	AMS1117-1.5/-1.8/-2.5/-2.85/-3.3/-5.0	ΔV_{OUT}, ΔV_{REF} = 1%, I_{OUT} = 0.8A (Note 4)		**1.1**	**1.3**	V
Current Limit	AMS1117-1.5/-1.8/-2.5/-2.85/-3.3/-5.0	(V_{IN} - V_{OUT}) = 1.5V	**900**	**1,100**	**1,500**	mA
Minimum Load Current	AMS1117	(V_{IN} - V_{OUT}) = 1.5V (Note 5)		**5**	**10**	mA
Quiescent Current	AMS1117-1.5/-1.8/-2.5/-2.85/-3.3/-5.0	(V_{IN} - V_{OUT}) = 1.5V		**5**	**11**	mA
Ripple Rejection	AMS1117	f =120Hz , C_{OUT} = 22µF Tantalum, I_{OUT} = 1A, (V_{IN}-V_{OUT}) = 3V, C_{ADJ} =10µF	**60**	**75**		dB
	AMS1117-1.5/-1.8/-2.5/-2.85	f =120Hz , C_{OUT} = 22µF Tantalum, I_{OUT} = 1A, V_{IN} = 4.35V	**60**	**72**		dB
	AMS1117-3.3	f =120Hz , C_{OUT} = 22µF Tantalum, I_{OUT} = 1A V_{IN} = 4.75V	**60**	**72**		dB
	AMS1117-5.0	f =120Hz , C_{OUT} = 22µF Tantalum, I_{OUT} = 1A V_{IN} = 6.5V	**60**	**68**		dB
Thermal Regulation	AMS1117	T_A = 25°C, 30ms pulse		0.008	0.04	%W
Adjust Pin Current	AMS1117	I_{OUT} =10mA , 1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		55	 **120**	µA µA
Adjust Pin Current Change	AMS1117	I_{OUT} =10mA, 1.5V≤ (V_{IN} - V_{OUT}) ≤ 12V		**0.2**	**5**	µA
Temperature Stability				**0.5**		%
Long Term Stability		T_A =125°C, 1000Hrs		0.3	1	%
RMS Output Noise (% of V_{OUT})		T_A = 25°C , 10Hz ≤ f ≤ 10kHz		0.003		%
Thermal Resistance Junction-to-Case		All packages			15	°C/W

Parameters identified with **boldface type** apply over the full operating temperature range.

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. For guaranteed specifications and test conditions, see the Electrical Characteristics. The guaranteed specifications apply only for the test conditions listed.

Note 2: Line and Load regulation are guaranteed up to the maximum power dissipation of 1.2 W for SOT-223, 2.2W for TO-252 and 780mW for 8-Lead SOIC. Power dissipation is determined by the input/output differential and the output current. Guaranteed maximum power dissipation will not be available over the full input/output range.

Note 3: See thermal regulation specifications for changes in output voltage due to heating effects. Line and load regulation are measured at a constant junction temperature by low duty cycle pulse testing. Load regulation is measured at the output lead ~1/8" from the package.

Note 4: Dropout voltage is specified up to 0.8A load. For currents over 0.8A dropout will be higher

Note 5: Minimum load current is defined as the minimum output current required to maintain regulation. When 1.5V ≤ (V_{IN} - V_{OUT}) ≤ 12V the device is guaranteed to regulate if the output current is greater than 10mA.

5.2 Prototype photos

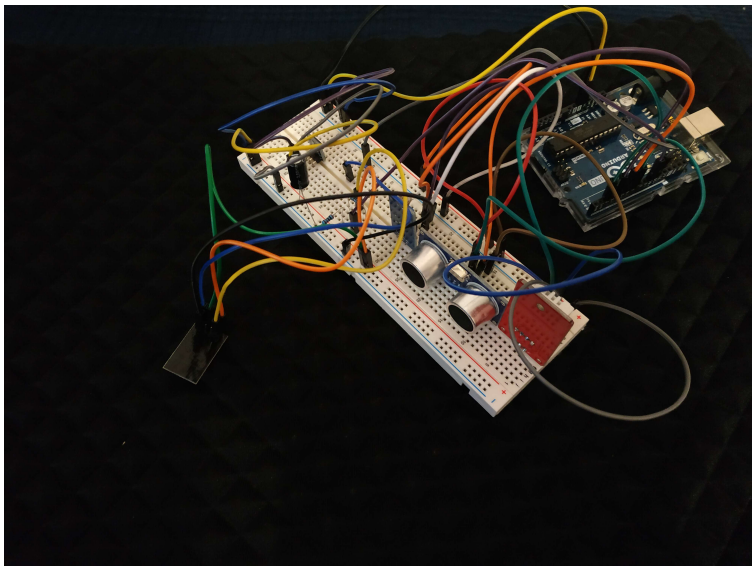


Figure 5.1: Left angle view of the prototype.

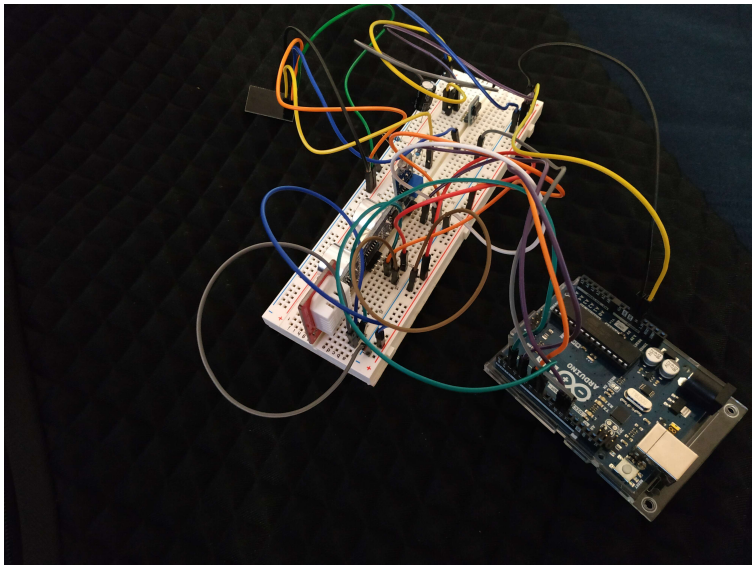


Figure 5.2: Right angle view of the prototype.

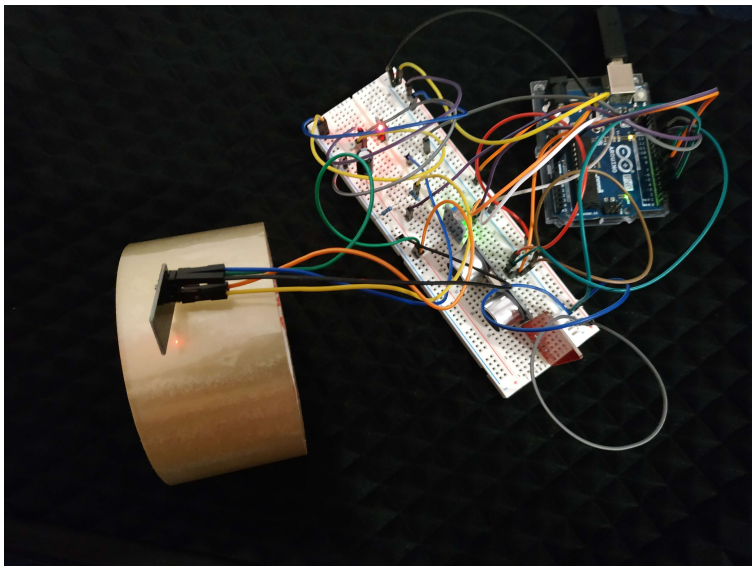


Figure 5.3: Prototype on and working.

5.3 Software routines

5.3.1 *Arduino_Uno_Coap_p2p.ino*

```
#include <SoftwareSerial.h>
#include <SimpleDHT.h>

#define pinRXesp 10
#define pinTXesp 11
#define pinDHT22 2
#define trigHCPin 7
#define echoHCPin 6
#define ldrPin 4

//variables HC-SR04
long duration;
int distance;
//variables DHT-22
float temperature = 0;
float humidity = 0;
//variable LDR
boolean light;

SimpleDHT22 dht22(pinDHT22);
```

```
21
22 SoftwareSerial Esp_serial(pinTXesp,pinRXesp); // TX,RX
23
24 void getDHT22Data(String type){//0 string, 3 temp, 4 hum
25   float temperature_l = 0;
26   float humidity_l = 0;
27   int err = SimpleDHTErrSuccess;
28   if ((err = dht22.read2(&temperature_l, &humidity_l,
         NULL)) == SimpleDHTErrSuccess) {
29     temperature=temperature_l;
30     humidity=humidity_l;
31   }
32   if(type.charAt(0)=='0'){
33     Esp_serial.print("0");Esp_serial.print((float)temperature);
         Esp_serial.print(" *C,
         ");Esp_serial.print((float)humidity);
         Esp_serial.println(" RH%");
34   }else if(type.charAt(0)=='3'){
35     Esp_serial.print("3");Esp_serial.println((float)temperature);
36   }else if(type.charAt(0)=='4'){
37     Esp_serial.print("4");Esp_serial.println((float)humidity);
38   }
39 }
40
41 void getlLDR393Data(){
42   light=digitalRead(ldrPin);
43   if(light==HIGH){
44     Esp_serial.print("1");Esp_serial.println("LOW Light");
45   }else{
46     Esp_serial.print("1");Esp_serial.println("HIGH
         Light");
47   }
48 }
49
50 void getHCS04Data(String type){
51   // Clears the trigPin
52   digitalWrite(trigHCPin, LOW);
53   delayMicroseconds(2);
54   // Sets the trigPin on HIGH state for 10 micro seconds
55   digitalWrite(trigHCPin, HIGH);
56   delayMicroseconds(10);
57   digitalWrite(trigHCPin, LOW);
58   // Reads the echoPin, returns the sound wave travel
         time in microseconds
59   duration = pulseIn(echoHCPin, HIGH);
60   // Calculating the distance
```

```
61   distance = duration*0.034/2;
62   if(type.charAt(0)=='2'){
63     Esp_serial.print("2");Esp_serial.print("Distance:
         ");Esp_serial.print(distance);
         Esp_serial.println(" cm");
64   }else if(type.charAt(0)=='5'){
65     Esp_serial.print("5");Esp_serial.println(distance);
66   }
67 }
68
69 void setup() {
70   pinMode(trigHCPin, OUTPUT);
71   pinMode(echoHCPin, INPUT);
72   pinMode(ldrPin, INPUT);
73   Esp_serial.begin(115200);
74   Serial.begin(115200);
75 }
76
77 void loop() {
78    if(Esp_serial.available()){
79      String s=Esp_serial.readStringUntil('\n');
80      Serial.println(s);
81      if(s.charAt(0)=='0' || s.charAt(0)=='3' ||
           s.charAt(0)=='4'){
82       getDHT22Data(s);
83      } else if(s.charAt(0)=='1'){
84       getlLDR393Data();
85      } else if(s.charAt(0)=='2' || s.charAt(0)=='5'){
86       getHCS04Data(s);
87      }
88    }
89 }
```

5.3.2 *ESP_Coap_p2p.ino*

```
1 #include <FirebaseESP8266.h>
2 #include <ESP8266WiFi.h>
3 #include <WiFiUdp.h>
4 #include <coap-simple.h>
5 #include <Queue.h>
6 //Definition of the WiFi SSID, password and
     configurations of cloud storage
7 #ifndef STASSID
8 #define STASSID "*******"
```

```
#define STAPSK "********"
#define FIREBASE_HOST "https://*******.firebaseio.com/"
#define FIREBASE_AUTH "*************"
#endif
const char* ssid = STASSID;
const char* password = STAPSK;
IPAddress my_ip(192, 168, 43, 18);
IPAddress gateway(192, 168, 43, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress dns(8, 8, 8, 8);
//Define FirebaseESP8266 data object
FirebaseData firebaseData;
String FPath = "/Coap";
//Definition of coap classes
WiFiUDP udp;
Coap coap(udp);
const int my_port = 5865;
//Definition of IP and port of the second CoAP server
const int peer_port = 11111;
IPAddress coap_peer(192, 168, 43, 14);
//Timeouts for automatic sensor readings (deactivated by
    default)
unsigned long timer=0;
boolean timer_active=false;
int action=-1;
//Class used to define a queue of requests on remote data
class request{
  public:
  IPAddress ip;
  int port;
  uint8_t messageid;
};
DataQueue<request> requestsQueue(10);//Queue of pending
    requests up to 10

//Callbacks
void callback_wn(CoapPacket &packet, IPAddress ip, int
    port){//Callback for /.well-known/core resources
  coap.sendResponse(ip, port,
      packet.messageid,"loop_update, dht22, lm393,
      hcsr04, time"); //Gives the resources as a responce
      to the client (</time> is the default resource on
      the remote server)
}

```

```
47 void callback_loop_update(CoapPacket &packet, IPAddress
       ip, int port){//Callback for /loop_update resource
48   if(packet.code==3){//PUT request
49     // extract payload
50     char p[packet.payloadlen + 1];
51     memcpy(p, packet.payload, packet.payloadlen);
52     p[packet.payloadlen] = '\0';
53     String message(p);
54
55     if (message.equals("0")){
56       timer_active=false;
57     }else if(message.equals("1")){
58       timer_active=true;
59     }
60
61   }
62   if(timer_active){
63     coap.sendResponse(ip, port,packet.messageid, "ON");
64   }else{
65     coap.sendResponse(ip, port,packet.messageid, "OFF");
66   }
67
68 }
69
70 void callback_dht22(CoapPacket &packet, IPAddress ip,
       int port) {//0 request to arduino for the DHT22 data
71   serialFlush();
72   Serial.println("0");
73   delay(200);
74   if(Serial.available()){
75     String result=Serial.readStringUntil('\n');
76     String s=result.substring(1);
77     char data[s.length() + 1];
78     s.toCharArray(data, s.length());
79     data[s.length()] = '\0';
80     if(result.charAt(0)=='0')
81       coap.sendResponse(ip, port,packet.messageid, data);
82     else
83       coap.sendResponse(ip, port,
             packet.messageid,"Error");
84   }else{
85     coap.sendResponse(ip, port, packet.messageid,"Error");
86   }
87 }
88
```

```
89 void callback_lm393(CoapPacket &packet, IPAddress ip,
       int port) {//1 request to arduino lm393
90   serialFlush();
91   Serial.println("1");
92   delay(200);
93   if(Serial.available()){
94     String result=Serial.readStringUntil('\n');
95     String s=result.substring(1);
96     char data[s.length() + 1];
97     s.toCharArray(data, s.length());
98     data[s.length()] = '\0';
99     if(result.charAt(0)=='1')
100      coap.sendResponse(ip, port,packet.messageid, data);
101    else
102      coap.sendResponse(ip, port,
           packet.messageid,"Error");
103  }else{
104    coap.sendResponse(ip, port, packet.messageid,
         "Error");
105  }
106 }
107
108 void callback_hcsr04(CoapPacket &packet, IPAddress ip,
       int port) {//2 request to arduino hcsr04
109   serialFlush();
110   Serial.println("2");
111   delay(200);
112   if(Serial.available()){
113     String result=Serial.readStringUntil('\n');
114     String s=result.substring(1);
115     char data[s.length() + 1];
116     s.toCharArray(data, s.length());
117     data[s.length()] = '\0';
118     if(result.charAt(0)=='2')
119       coap.sendResponse(ip, port,packet.messageid, data);
120     else
121       coap.sendResponse(ip, port,
            packet.messageid,"Error");
122   }else{
123     coap.sendResponse(ip, port, packet.messageid,
         "Error");
124   }
125 }
126
127 void callback_time(CoapPacket &packet, IPAddress ip, int
       port) {
```

```
128     request req;
129     req.ip=ip;
130     req.port=port;
131     req.messageid=packet.messageid;
132     requestsQueue.enqueue(req);
133     coap.get(coap_peer, peer_port, "time");
134 }
135
136 void callback_response(CoapPacket &packet, IPAddress ip,
        int port) { //In response of a request
137   //Retrieves payload
138   char p[packet.payloadlen + 1];
139   memcpy(p, packet.payload, packet.payloadlen);
140   p[packet.payloadlen] = '\0';
141   String message(p);
142   while(!requestsQueue.isEmpty()){
143     request r=requestsQueue.dequeue();
144     coap.sendResponse(r.ip, r.port, r.messageid,p);
145   }
146   if(timer_active){//send to Firebase
147     Firebase.setString(firebaseData,FPath+"/time/data",
          message);
148   }
149 }
150
151 void loop_update(int i){//Function called in loop that
      when active sends calls to the other CoAP servers and
      data to Firebase
152   if(i>=0){
153     serialFlush();
154     Serial.println(i);
155     delay(200);
156     if(Serial.available()){
157       String result=Serial.readStringUntil('\n');
158       String s=result.substring(1);
159       char data[s.length() + 1];
160       s.toCharArray(data, s.length());
161       data[s.length()] = '\0';
162       if(i==0){
163         if(result.charAt(0)=='0')
164           Firebase.setString(firebaseData,FPath+"/dht22/data/string",
              String(data));
165         serialFlush();
166         Serial.println(3);//Only temperature
167         delay(200);
168         if(Serial.available()){
```

```
169         result=Serial.readStringUntil('\n');
170         s=result.substring(1);
171         data[s.length() + 1];
172         s.toCharArray(data, s.length());
173         data[s.length()] = '\0';
174         if(result.charAt(0)=='3')
175           Firebase.setString(firebaseData,
                FPath+"/dht22/data/temp", String(data));
176         serialFlush();
177         Serial.println(4);//Only humidity
178         delay(200);
179         if(Serial.available()){
180           result=Serial.readStringUntil('\n');
181           s=result.substring(1);
182           data[s.length() + 1];
183           s.toCharArray(data, s.length());
184           data[s.length()] = '\0';
185           if(result.charAt(0)=='4')
186             Firebase.setString(firebaseData,
                  FPath+"/dht22/data/hum", String(data));
187         }
188       }
189     }else if(i==1){
190       if(result.charAt(0)=='1')
191         Firebase.setString(firebaseData,FPath+"/lm393/data",
              String(data));
192     }else{
193       if(result.charAt(0)=='2')
194         Firebase.setString(firebaseData,
              FPath+"/hcsr04/data/string", String(data));
195       serialFlush();
196       Serial.println(5);//Only distance
197       delay(200);
198       if(Serial.available()){
199         result=Serial.readStringUntil('\n');
200         s=result.substring(1);
201         data[s.length() + 1];
202         s.toCharArray(data, s.length());
203         data[s.length()] = '\0';
204         if(result.charAt(0)=='5')
205           Firebase.setString(firebaseData,
                FPath+"/hcsr04/data/dist", String(data));
206       }
207     }
208   }
209 }else{
```

```
210     coap.get(coap_peer, peer_port, "time");
211   }
212 }
213
214 void serialFlush(){
215   while(Serial.available() > 0) {
216     String s=Serial.readStringUntil('\n');
217   }
218 }
219
220 void setup() {
221   //Serial is used to communicate wih Arduino and
        retrieve data about its connected sensors
222   Serial.begin(115200);
223   // We start by connecting to the WiFi network with a
        static IP
224   WiFi.mode(WIFI_STA);
225   WiFi.setAutoConnect(false); // Not connecting by its own
226   WiFi.disconnect(); //Prevent connecting to wifi based
        on previous configuration
227   WiFi.config(my_ip, gateway, subnet, dns);
228   WiFi.begin(ssid, password);
229   while (WiFi.status() != WL_CONNECTED) {
230     delay(500);
231   }
232   // Init Firebase data
233   Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
234   Firebase.reconnectWiFi(true);
235   Firebase.setReadTimeout(firebaseData, 1000 * 60);
236   //Size and its write timeout e.g. tiny (1s), small
        (10s), medium (30s) and large (60s).
237   Firebase.setwriteSizeLimit(firebaseData, "tiny");
238   // Init of coap callbacks for the server
239   coap.server(callback_wn, ".well-known/core");
240   coap.server(callback_loop_update, "loop_update");
241   coap.server(callback_time, "time");
242   coap.server(callback_hcsr04, "hcsr04");
243   coap.server(callback_dht22, "dht22");
244   coap.server(callback_lm393,"lm393");
245   coap.response(callback_response);
246   coap.start(my_port);
247
248 }
249
250 void loop() {
```

```
251   if(WiFi.status() != WL_CONNECTED){//If disconnected
          from Wifi, retry to connect again
252     WiFi.disconnect();
253     WiFi.config(my_ip, gateway, subnet, dns);
254     WiFi.begin(ssid, password);
255     while (WiFi.status() != WL_CONNECTED) {
256       delay(500);
257     }
258   }
259   coap.loop();//Coap checks
260   if(timer_active){//Update function
261     if(millis()>timer){
262       timer=millis()+300;//Timer
263       loop_update(action);
264       action+=1;
265       if(action>2){
266         action=-1;
267       }
268     }
269   }
270 }
```

Bibliography

[1] Adafruit. Adafruit-mqtt-library github project. Available from: https://github.com/adafruit/Adafruit_MQTT_Library.

[2] Antunes, M., Gomes, D., and Aguiar, R. L. Scalable semantic aware context storage. *Future Gener. Comput. Syst.*, **56** (2016), 675. Available from: https://doi.org/10.1016/j.future.2015.09.008.

[3] Antunes, M., Jesus, R., Gomes, D., and Aguiar, R. Improve iot/m2m data organization based on stream patterns (2017).

[4] Bormann, C. Coap, rfc 7252 constrained application protocol (2016). Available from: https://coap.technology/.

[5] Cayenne. Cayenne-mqtt-library github project. Available from: https://github.com/myDevicesIoT/Cayenne-MQTT-Arduino.

[6] Cloud, A. Create your own im social app with mqtt (2017). Available from: https://medium.com/@Alibaba_Cloud/create-your-own-im-social-app-with-mqtt-703893695103.

[7] Collina, M., Corazza, G., and Vanelli-Coralli, A. Introducing the qest broker: scaling the iot by bridging mqtt and rest. pp. 36–41 (2012). ISBN 978-1-4673-2566-0.

[8] Dürkop, L., Czybik, B., and Jasperneite, J. Performance evaluation of m2m protocols over cellular networks in a lab environment (2015).

[9] Farooqi, N., Gutub, A., and Khozium, O. Smart community challenges: Enabling iot/m2m technology case study. *Life Science Journal*, **16** (2019), 11.

[10] Frigieri, E., Mazzer, D., and Parreira, L. M2m protocols for constrained environments in the context of iot: A comparison of approaches (2015).

[11] Gaehwiler, J. arduino-mqtt github project. Available from: https://github.com/256dpi/arduino-mqtt.

[12] Gaehwiler, J. lwmqtt github project. Available from: https://github.com/256dpi/lwmqtt.

[13] Hunkeler, U., Linh Truong, H., and Stanford-Clark, A. Mqtt-s — a publish/subscribe protocol for wireless sensor networks. pp. 791 – 798 (2008).

[14] (IBM), I. B. M. C. Mqtt: Message queuing telemetry transport, version 3.1, protocol specification. *Eurotech*, (2010), 1.

[15] Jouni Mäenpää, J. J. B. and Loreto, S. Using reload and coap for wide area sensor and actuator networking (2012). Available from: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/1687-1499-2012-121.

[16] Kalyani, V., Gaur, P., and Priya, S. Iot: 'machine to machine' application a future vision. *Journal of Management Engineering and Information Technology*, **2** (2015), 2394.

[17] Kazmi, S., U. Khan, L., Tran, N., and Hong, C. S. *5G Networks*, pp. 1–12 (2019). ISBN 978-3-030-16169-9.

[18] Levä, T., Mazhelis, O., and Suomi (née Warma, H. Comparing the cost-efficiency of coap and http in web of things applications. *Decision Support Systems*, **In press** (2013), 23.

[19] Losant. Losant-mqtt-arduino github project. Available from: https://github.com/Losant/losant-mqtt-arduino.

[20] Martinez, I. Opc library github project. Available from: https://github.com/SoftwareTools4Makers/OPC.

[21] Niisato, H. Coap simple library github project. Available from: https://github.com/hirotakaster/CoAP-simple-library.

[22] RF Wireless World 2012, R. . W. V. and Resources. Home of rf and wireless vendors and resources. Available from: https://www.rfwireless-world.com/Articles/M2M.html.

[23] Shrestha, G., Imtiaz, J., and Jasperneite, J. An optimized opc ua transport profile to bringing bluetooth low energy device into ip networks. pp. 1–5 (2013). ISBN 978-1-4799-0862-2.