A safer smart home? Feasibility analysis of privacy mechanisms for MQTT and CoAP in smart home

Trindade, Fernando
contato@fernandohenrique.com

Ribeiro, Admilson
admilson@dcomp.ufs.br

Bispo, Kalil
kalilbispo@gmail.com

Júnior, Rubens
rubens.matos@gmail.com

August 16, 2022

Abstract

Smart Home technologies are among the fastest-growing markets in consumer electronics over the years. It is necessary to use lightweight communication protocols between smart objects to avoid overloading them. Due to limited hardware resources in SHS (Smart Home System), requirements such as response time, low processing load, and low energy consumption tend to be prioritized over security and privacy. However, insecure communication technologies might expose users privacy and intimacy. Given the above, it is necessary to assess the feasibility of adding protection to the privacy of the IoT application layer. This work employs testbed experiments to measure whether the addition of authenticity and confidentiality in MQTT (Message Queue Telemetry Transport) and CoAP protocols is effective in SHS. The experiments are carried out in an ESP32 microcontroller and the results are analyzed with statistical methods regarding performance, energy efficiency, and usability.This evaluation concludes that although the overload adding security in SHS is feasible in any aspect.

1 Introduction

The Internet of Things (IoT) encompasses a set of technologies under active development and that makes the world more and more connected. This new category of computer networks expands the capacity for communication and information between devices and humans. As a result, previously supposed networks meet the requirements: at any time, any place, now also meet at any things. However, on the other hand, information security and privacy becomes a critical concern [SAR15].

The IoT has influenced a massive explosion in the volume of data trafficked on the Internet and stored in the cloud. The Norwegian research organization *Stiftelsen for industriell og teknisk forskning (SINTEF)* pointed out that in mid-2015, 90% of the world's data was generated at a speed of over 205,000 gigabytes per second, which was approximately equivalent to 150 million books [DR15].

The Internet of Things is quite broad and can be subdivided into several application areas, such as smart cities, smart homes, health care, industry automation, agricultural automation, among many others. According to Gartner [Gar19] among the many sub-areas of IoT, the fastest growing ones are smart homes and health care.

Smart home is a house that incorporates a communication network connecting electrical appliances and essential services, in order to allow them to be controlled, monitored, and accessed remotely [Kin03]. The development of microcontrollers integrated with low-cost network cards empowered a growing demand for smart and connected appliances. In addition, the mass production of those electronics popularized the use of objects that bring convenience and comfort through communication, monitoring, and interaction with ordinary people in their homes. The benefits of using those technologies range from physical access control through biometric authentication to the activation of household appliances by voice commands.

Faced with this scenario, where technology is inserted in people's daily lives, even in the most intimate moments inside their own homes and that these low-power devices processing capabilities are limited to render a robust layer of security, it is necessary be very careful about data privacy and

security. Several researches, from different perspectives, have already been carried out on security in home IoT networks, but some gaps still remain and must be explored. With the aim of better understanding problems related to security and privacy, in the aforementioned context, was carried out first a systematic mapping where the privacy challenge was detected in IoT networks. Subsequently, a systematic review was carried out to identify which works are addressing this issue and whether there are still issues that need to be addressed.

The result of mapping and review served as the basis for this research, which aims to to analyze the feasibility of adding privacy protection mechanisms, in the application of IoT networks in smart homes. This research intends to demonstrate that a solution for data privacy is viable in smart homes, but it is necessary to achieve a satisfactory balance for the triad: energy consumption, performance, and usability.

One of the fundamental principles of the Internet of Things concept is building a solution suited to the needs of applications considered as designated non-functional requirements. IoT is a feature of working with limited resources, or has systems embedded in devices with little computing power and interconnected in a network. In this way, the choice of protocols drastically influenced the energy consumption and performance of devices in an Internet of Things network. It can be noted when realizing the expressiveness of IoT in the current scenario. According to [Gar19], IoT is among the 8 technologies highlighted for 2019. In addition, it is estimated that 26 billion of connected IoT devices by value in 2020 $1.9 trillion global economic aggregates, moreover, IoT product and service providers have generated incremental revenue in excess of US$300 billion, primarily in services. Already Lueth (2020), believes that by 2025 there will be 22 billion devices. In this way, it becomes essential assertively choose which communication protocol and security mechanisms should be used, since energy efficiency, are categorical transmission rate and coverage unsuccessful application in the context of IoT.

The need to protect smart home networks in terms of privacy is evident, given their relevance. Thus, there is a need for new forms of protection to be developed that have the characteristic of taking into account performance requirements, energy consumption, and usability.

For this, it is necessary to answer questions about: Does the smart home IoT environment support, from a performance point of view, high levels of privacy protection?; Is the increased level of privacy protection feasible from a performance point of view? (response time); Is the increase in the level of privacy protection feasible from an economic point of view? on which devices? (Can energy-saving device be safe?) and Is the security enhancement feasible from the user's point of view? (usability, complex settings). In this way, it becomes crucial to answer these questions, to ensure that safe smart home environments allow users to use their devices with an acceptable level of privacy.

This work was carried out in order to resolve some questions about the privacy protection mechanisms in application layer protocols in IoT networks. For this, it was essential to define objectives which guided the answers to the aforementioned questions.

The general objective of this work is to verify the feasibility of adding privacy protection mechanisms in the application layer, in IoT networks in the smart home context. This increase in security aims to guarantee the confidentiality of the information transmitted, without compromising the quality requirements related to performance and energy efficiency to the point of making a smart home solution unfeasible.To achieve the main objective, the following specific objectives were defined:

- Raise the main privacy-related vulnerabilities in the application layer of IoT networks IoT.
- Raise the main application layer protocols (which run over TCP and UDP) and their confidentiality options for smart home networks.
- Analyze the possible impacts of vulnerable traffic in the smart home context.
- Verify, by prototyping, the evaluation of privacy mechanisms using a methodology for measuring response time and increase in electric current consumption.
- Determine the viable scenario so that the application layer can provide privacy on smart home devices.

In this paper we will focus on the assessment of those factors, by means of an experimental smart home setup that employs low cost hardware (ESP32) running a publish-subscribe service enabled by MQTT and MQTT+TLS (Transport Layer Security) over TCP. We also did the same experiments using client/server CoAP and CoAP + DTLS over UDP.

For this, we will make a brief introduction about the context in which smart home networks involve in section I.In section II, we present an analysis of some related works. In section III, we demonstrate the experiments setup and configurations that were carried out in this research. Section IV highlights the main results of our experiments, and Section V draws conclusions and open challenges for future work.

2 Analysis of Related Research Papers

In this section, we carry out an analysis of primary related studies by means a systematic review of the theme. In addition, we performed a search in the main knowledge bases on the security and privacy mechanisms at the application layer of IoT networks, considering SHS environments.

In the work of [PKBT18] a performance comparison is made between some of the main application layer IoT protocols (AMQP, MQTT, and XMPP). The authors address an automation environment in business buildings, which often resemble the devices used in smart homes. Still, they do not consider non-functional requirements such as security, privacy, and energy consumption. The authors suggest future work to do the same analysis using different settings of QoS and security.

[PMY19] proposed a comparison between the protocols: MQTT and CoAP, both from the IoT application layer. The authors used NodeMCU ESP8266, a microcontroller compatible with smart home devices. Although this performance analysis is interesting for comparing protocols of different architectures (one runs on TCP and the other on UDP), the protection mechanisms and their overload on this type of communication are not considered. Due to the characteristics of each protocol, the most appropriate use is evident in the situation in which they will be used.

[PV19] They present a form of anonymous communication in Iot networks using the MQTT protocol with the dynamic bridge mechanism (dynamic bridging). This solution is pertinent concerning privacy since it consists of onion type routing among the brokers; however, an elaborate study on the impact on performance and the usability of the scheme has not been proposed. Furthermore, although it provides a very significant contribution in the context of IoT networks, for it to be applied in an innovative home environment, a more in-depth study of the costs of acquiring various devices and their energy consumption would be necessary.

The research of [JKH20] presents a framework for application development to run on a smart home hub. The purpose of the algorithm is to pre-process raw data and remove information that could compromise users' privacy. This proposal aims at guaranteeing privacy, but a detailed analysis of the impact on communication is not presented. The framework Peekaboo adds considerable complexity to the user of smart home and does not make data on energy consumption explicit.

The [PWDC18] project proposes an architecture with an encapsulation layer that abstracts the discovery of new devices and access to them through doors. This proposal can be promising concerning the orchestration of devices and standardize the way they are accessed. However, the authors clarify that it is necessary to evolve considerably in terms of authentication, authorization, performance, and energy consumption. An in-depth performance analysis could show whether this mechanism would be viable for the IoT network environment in the context of smart home.

The article developed by [SBZB19] proposes to use the open source implementation of CoAP and integrate it with Datagram Transport Layer Security (DTLS), in order to create a secure data transfer channel between IoT devices. The impact of adding DTLS to COAP was studied, through experiments using real data, on IoT devices with limited resources and open source software. Testing has shown that using a CoAP-DTLS implementation with a symmetric-key cipher suite resulted in noticeable performance costs. A secure connection with DTLS over CoAP consumed approximately 10% more power than a non-secure connection. In addition, latency tests revealed a more than 100% increase in average latency time for secure messages compared to non-secure messages. We also encountered some of implementation challenges when developing a real IoT test environment for safe experimentation.

[RHPV17] implemented secure CoAP, protected by DTLS, for resource-constrained IoT devices and a cloud backend to evaluate the three security modes of CoAP: pre-shared key, raw public key, and certificate-based. For this purpose, a configuration based on real cloud-connected IoT applications was used. The authors have extended SicsthSense, a cloud platform for the IoT, which uses the CoAP protocol along with a DTLS implementation for resource-constrained IoT devices. The tests were performed using symmetric and asymmetric cryptography. For this work, the authors created Secure-Sense, which is an open-source, BSD-licensed End-to-End (E2E) secure communication architecture

for the IoT. An evaluation benchmark was carried out that makes it possible for IoT service and product providers to account for the security overhead when using the protocols. The main contributions of this article are full implementation of CoAP security modes for E2E IoT security; IoT security and communication protocols for a cloud platform for the IoT; in-depth experimental evaluation and benchmarking of E2E security between a network of smart things and a cloud platform.

As can be seen in the text above, none of the works provide a complete analysis of privacy in the IoT application layer. [PKBT18] do not consider the main QoS requirements. In [PMY19], the authors do not analyze energy consumption costs. In the works of [SBZB19] and [RHPV17], energy efficiency is considered, however, the combination of CoAP + DTLS protocols is limited without comparing them with other protocols of the Application Layer. [PV19] and [JKH20] do not conduct a study on the impact on performance and usability. However, [PWDC18] talks about usability but does not delve into other non-functional and essential requirements in smart home environments.

3 Experiments

To demonstrate the experiment, it is necessary to present the environment in which the tests were performed. This section will cover hardware, libraries, and firmware, as well as the necessary configurations to carry out the experimentation.

3.1 Hardware

Two ESP32 NodeMCUs were used to execute the client codes, one for the MQTT protocol and the other for the CoAP, as well as the TLS and DTLS protections respectively. Thus, the hardware used as a client was intended to collect response time measurements. Aiming at non-interference in the processing on the client side and consequently, in the response time of the requests, the energy measurement was performed using an Arduino Nano V3. The server-side applications were run on a Raspberry Pi 3 Model B cabled to a mesh Intelbras Twibi Giga router. ESP32 microcontrollers are connected to the same router via WiFi connection.

According to [Nan21], the Arduino Nano is a development and prototyping board with the embedded ATmega328 microcontroller. The Arduino Nano works with a clock speed of 16 MHz, has a power consumption of 19 mAh, and can be purchased for less than two dollars. Additionally, it can be powered via the USB Mini-B connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected for the highest voltage source. The ATmega328 microcontroller has 32 KB of internal memory, of which 2 KB is used for the bootloader. It also has 2KB of SRAM and 1KB of EEPROM.

The Arduino Nano can be programmed with the Arduino IDE software. In addition, the ATmega328 on the Arduino Nano comes pre-programmed with a bootloader, which is a program installer that allows you to load new program code without using an external hardware programmer.

Nano, which can be viewed in Figure 1, was used in this work to process separately the data collected from the energy consumption module. The choice of this board was due to its low cost, sufficient processing power for the experiment, ease of integration with the measurement module, and ease of code development for this purpose.

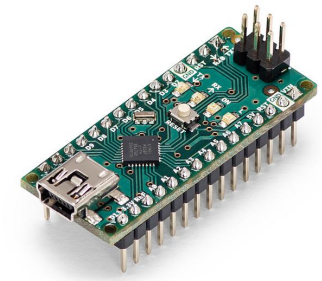

Figure 1: Arduino Nano V3

The company [Tex22] describes this module as a current meter and energy monitor with an interface compatible with I2C or SMBUS. The device monitors voltage drop and bus supply voltage, with programmable conversion and filtering times. Thus, a programmable calibration value, combined with an internal multiplier, allows direct readings of current in amps and an additional multiplication register calculates power in watts. In addition, the interface is compatible with I2C or SMBUS and has 16 programmable addresses.

This module was chosen because it is easy to use and integrates with Arduino, in addition to having a good resolution in measuring DC voltage and current. With this, it became possible to isolate the consumption measurement circuit of the NodeMCU microcontroller, which was used in overhead experiments applying security protocols. In this way, the measurement becomes reliable and does not interfere with the processing of the experiment.

The INA219 I2C DC Current Sensor board, which can be seen at Figure 2, facilitates projects involving this component, as it already includes the integrated circuit and the passive components necessary for its operation.

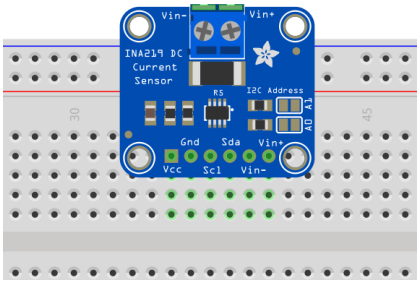

Figure 2: INA219 Texas Intruments

ESP32 is a System On Chip (SoC) or Microcontroller System on a Chip with several built-in features: Wi-Fi (2.4GHz band), Bluetooth, two 32-bit Xtensa LX6 cores, and various peripherals. The Figure 3 shows the input/output pins of the NodeMCU ESP32. The version used in the experiments was a development kit that integrates the ESP32-D0WD-V3 version of the microcontroller.

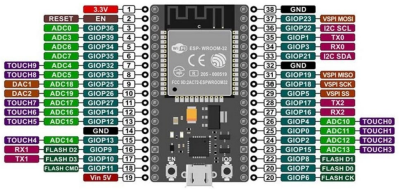

Figure 3: NodeMCU ESP32 I/O Pin Datasheet

Manufactured by the company Raspberry Pi Foundation, the Raspberry pi 3 model B is a Single Board Computer (SBC), that is, a single board computer, with a Broadcom BCM2837 quad-core 1.2 GHz 64-bit processor, 1 GB RAM, BCM43438 wireless LAN, and Bluetooth Low Energy (BLE). The operating system used was Raspbian. In this research, Raspberry was used as a server and the Mosquitto broker was installed on it, to use MQTT, and the LibCoAP library to run a CoAP server code. The Figure 4 shows the Raspberry pi 3 model B [Fou21].

The Wi-Fi Router 5 Mesh - Twibi Giga by Intelbras uses mesh network technology to improve the connection between devices. Wi-Fi mesh is a wireless networking technology that uses a mesh topology, where all nodes are connected, acting as a single wireless network with several scattered points. In this model, the transfer rate is up to 867 Mbps at 5 GHz (802.11ac) and 300 Mbps at 2.4 GHz (802.11n), in addition to guaranteeing up to 60 connected users, and having a Wi-Fi signal coverage area. of up to 180m^2 in open spaces [Int21].An illustrative image of the router can be seen at Figure 5

Figure 4: Raspberry pi 3 model B

Figure 5: Twibi Giga

3.2 Libcoap client application

To experiment with the arguments proposed for this dissertation, the coap_client example provided by the ESP-IDF framework was used as a base, since it was the only functional CoAP + DTLS example found. The code used was modified so that instead of sending a single request to the server, it would send a request every second, repeating this process 100 times. When sending a request, a time counter is started that only ends counting as soon as the response returns from the server. This request and response time are printed on the console and collected for Round-trip time (RTT) analysis. Figure 6 shows the main changes highlighted in red.

The second version of this code was used by adding PSK and PKI settings with a self-signed certificate to provide secure communication between client and server. In addition, a pair of 2048-bit RSA keys was used, which were generated with the help of the OpenSSL framework through the OpenSSL command line in Figure 7.

It is important to note that the generated certificate is self-signed, and considered sufficient to be used in the experiments. Ideally, it would involve a valid certification authority signature, but for cost reasons, it was not used. For this reason, the coap_ca.pem and coap_client.crt certificates are identical. For organization purposes, these files were stored in the certs directory, in the project's root folder.

3.3 MQTT client application

In order to investigate the arguments proposed for this dissertation, the mqtt_ssl and mqtt_tcp examples provided by the ESP-IDF platform were used as a basis for the experiments. The codes used were modified so that instead of sending a single request to the server, it sent a request every second, repeating this process 100 times. When sending a request, a time counter is started that only ends counting as soon as the response returns from the server. This request and response time are printed on the console and collected for RTT analysis. The MQTT client shows the main changes in the code snippet between the comments: *//–EXPERIMENT—* and *//—END OF EXPERIMENT—*.

3.4 MQTT Broker

The Mosquitto broker is used to implement the MQTT protocol and because it is light, it makes it compatible to be used in the Internet of Things. It is responsible for managing MQTT publications and subscriptions, that is, its function is to intermediate and automate incoming messages and then

```c
int cont = 0;


while(cont <= 99){
    request = coap_new_pdu(session);
    if (!request) {
        ESP_LOGE(TAG, "coap_new_pdu() failed");
        goto clean_up;
    }
    request->type = COAP_MESSAGE_CON;
    request->tid = coap_new_message_id(session);
    request->code = COAP_REQUEST_GET;
    coap_add_optlist_pdu(request, &optlist);

    resp_wait = 1;
    tempo_inicial = esp_timer_get_time();
    coap_send(session, request);

    wait_ms = COAP_DEFAULT_TIME_SEC * 1000;

    while (resp_wait) {
        int result = coap_run_once(ctx, wait_ms > 1000 ? 1000 : wait_ms);

        if (result >= 0) {
            if (result >= wait_ms) {
                ESP_LOGE(TAG, "select timeout");
                break;
            } else {
                wait_ms -= result;
            }
        }
    }
    sleep(1);
    cont++;
```

Figure 6: CoAP Client Sample

```
C:\ESP_IDF\examples\protocols\coap_client> openssl req -x509 -sha256 -nodes -newkey rsa:2048 -keyout keyfile.pem -out certfile.pem
```

Figure 7: Command to generate certificate.

send them to their recipients [Mos21]. In studying the performance of MQTT and MQTT + TLS, Esp-IDF is used to run client code on NodeMCU ESP32. Thus, the NodeMCU acts as a device that sends a message, registers the moment of sending, and waits for the receipt of the same message, counting the duration time (sending + receiving) when receiving the message.

To configure TLS security in Mosquitto it is essential to create a certificate authority, server keys, and certificates. OpenSSL software is used to generate the keys and certificates. The first step is to create a key to simulate the certification authority that will be responsible for validating the certificate. After that, a certificate is created for the certificate authority. And then we create the key destined for the server. Upon completion of the process, we will configure the server to require the certificate.

By default, Mosquitto is configured with port 1883 to be used in pure MQTT. After that, to use TLS encryption, the configuration file that is present in the Mosquitto directory must be changed to

```c
static void mqtt_app_start(void)
{
    const esp_mqtt_client_config_t mqtt_cfg = {
        .uri = CONFIG_BROKER_URI,
        .cert_pem = (const char *)mqtt_eclipse_org_pem_start,
    };

    ESP_LOGI(TAG, "[APP] Free memory: %d bytes", esp_get_free_heap_size());
    esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
    esp_mqtt_client_start(client);
    //--------------EXPERIMENTO-----------------------------------------------------------
    sleep(2);
    cont = 0;
    // Como a resposta da requisição é assíncrona e interrmpre o fluxo de processamento,
    // inicia-se o tempo do contador, faz-se a requisição. No momento da chegada da resposta
    // calcula-se a diferença entre o tempo da requisição e o da resposta. Entre uma (requisição+resposta)
    // executa-se o comando para dormir por um segundo.
    while  (cont < 1000 ){
        sleep(1); // Dorme por 1s entre uma mensagem (requisição + resposta) e outra.
        tempo_inicial = esp_timer_get_time();
        msg_id = esp_mqtt_client_publish(client, "/topic/tls/qos2", msg , 0, 2, 0);
        cont++;



    }
    //-------------FIM DO EXPERIMENTO----------------------------------------------------------
}
```

Figure 8: MQTT Client Sample

run on port 8883. When opening the Config file, it is necessary to change the port to 8883, as shown in Figure 9.

```
200 # ================================================================
201 # Default listener
202 # ================================================================
203
204 # IP address/hostname to bind the default listener to. If not
205 # given, the default listener will not be bound to a specific
206 # address and so will be accessible to all network interfaces.
207 # bind_address ip-address/host name
208 #bind_address
209
210 # Port to use for the default listener.
211 port 8883
```

Figure 9: Port configuration Mosquitto.config

Then we need to add the path where the certificate issued by the certificate authority called "ca.crt" is, the server key "server.key" and the file to validate the server certificate "server.crt", as shown in Figure 10.

```
252 # ----------------------------------------------------------------------
253 # Certificate based SSL/TLS support
254 # ----------------------------------------------------------------------
255 # The following options can be used to enable SSL/TLS support for
256 # this listener. Note that the recommended port for MQTT over TLS
257 # is 8883, but this must be set manually.
258 #
259 # See also the mosquitto-tls man page.
260
261 # At least one of cafile or capath must be defined. They both
262 # define methods of accessing the PEM encoded Certificate
263 # Authority certificates that have signed your server certificate
264 # and that you wish to trust.
265 # cafile defines the path to a file containing the CA certificates.
266 # capath defines a directory that will be searched for files
267 # containing the CA certificates. For capath to work correctly, the
268 # certificate files must have ".crt" as the file ending and you must run
269 # "openssl rehash <path to capath>" each time you add/remove a certificate.
270 #cafile
271 #capath
272 cafile C:\Program Files\mosquitto\certs\ca.crt
273
274 # Path to the PEM encoded server certificate.
275 #certfile
276 certfile C:\Program Files\mosquitto\certs\server.crt
277
278 # Path to the PEM encoded keyfile.
279 #keyfile
280 keyfile C:\Program Files\mosquitto\certs\server.key
281
```

Figure 10: Mosquitto.config Certificates

Once this is done, we need to inform which version of TLS, as seen in Figure 11. In the end, the certificate must be copied to the client.

```
310 # This option defines the version of the TLS protocol to use for this listener.
311 # The default value allows all of v1.3, v1.2 and v1.1. The valid values are
312 # tlsv1.3 tlsv1.2 and tlsv1.1.
313 tls_version tlsv1.2
```

Figure 11: TLS - Mosquitto.config

4 Results

In this section, details of the execution of the experiments will be discussed, as well as a statistical analysis of the measured results. Given the questions proposed by the work, studies were carried out to assess the feasibility from an economic point of view, performance, and ease of use. The objective is to answer if the security overload in relation to privacy, in the communication using the chosen protocols, can make unfeasible the use of low-cost microcontrollers in a smart home network.

4.1 Analysis of energy consumption measurement

To measure the energy consumption of the execution of the experiments on the client side (NodeMCU), a second microcontroller of the Arduino Nano model was used together with the Ina219 module, a DC current meter, using the I2C communication protocol. The choice of using a second microcontroller only for current measurement was due to the need to isolate the measurement circuit in relation to

the experiment circuit. In this way, it is possible to guarantee that all consumption measured will only be from the execution of protocols and their security overloads in ESP32. Furthermore, in order to check for any kind of calibration problem in the Ina219 sensor, a multimeter was used to check the measurement between the power supply and the ESP32, which showed identical results in both measurements.

The Figure 12 shows the ESP32 connected to an external power supply, represented by the 600mAh battery, and the Ina219 sensor connected in series, thus measuring the entire load that the microcontroller demands from the battery. The Arduino in turn is powered by the USB port of a computer, which in addition to supplying power to it also receives the data collected by the sensor.

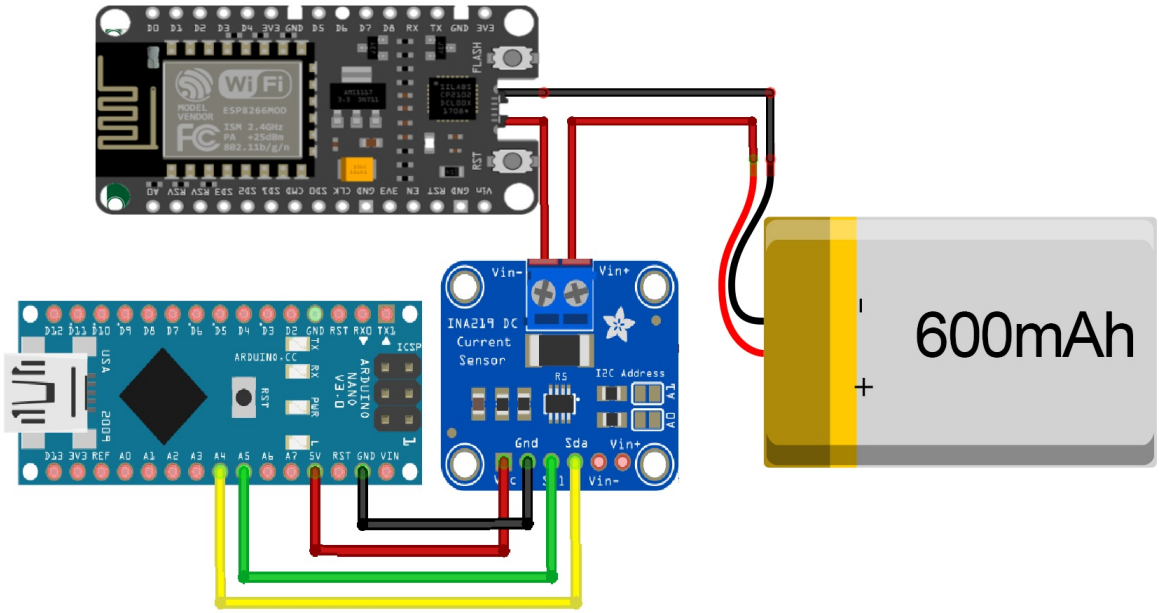

Figure 12: Current measurement circuit

4.1.1 Comparative analysis of energy consumption in the CoAP protocol

The electrical current measurements measured through the Ina219 module use the mA (milliampere) measured unit. 100 mA observations were collected for each sample of each experiment. For a total of 5 samples per experiment, the mean between the observations and finally the mean between the samples were calculated. The average index was chosen because it is sensitive to extreme values, which reflects the reality when referring to the financial cost for energy consumption. The Figure 13 denotes the means in each experiment with the CoAP protocol.

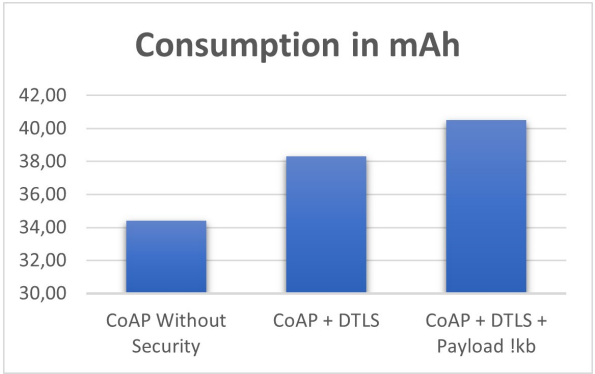

Figure 13: CoAP Energy Consumption

In order to calculate the total energy load consumed in the experiments, it was necessary to apply some scale conversions and measurement units. To exemplify the calculations performed in each experiment, the step-by-step procedure using the first average (unsafe CoAP protocol) is listed below:

- 34.4mA / 1000 = 0.0344 A (milliampere to ampere conversion);
- 3.3 v * 0.35 A = 0.11352 W (power calculation in watts using ohm's law);
- 0.1155 W / 1000 = 0.00011352 kW (watt to kilowatt conversion);
- 0.0001155 kW *24h = 0.00272448 kWh (daily consumption in kilowatt hours);
- 0.002772 kWh * 30 days = 0.0817344 kWh (monthly consumption);
- 0.08316 kwh * R$ 1.2 (rate applied by the energy company) = 0.09808 (approximately 9 cents of reais per month) .

Following the same steps for the other averages, the following results were obtained:

- For CoAP + DTLS, the cost in reais was: 0.091 KWh which costs 0.109 reais or 10 cents;
- For CoAP + DTLS + Payload 1 kb, the cost was: 0.0962 kWh which costs 0.115 reais or 11 cents.

4.1.2 Comparative analysis of energy consumption in the MQTT protocol

As in the consumption assessment of the CoAP protocol, 100 mA observations were collected for each sample of each experiment. With a total of 5 samples per experiment, the mean between the observations and finally the mean between the samples were calculated. The average index was chosen because it is sensitive to extreme values, which reflects the reality when referring to the financial cost for energy consumption. The Figure 14 denotes the means in each experiment with the MQTT protocol.

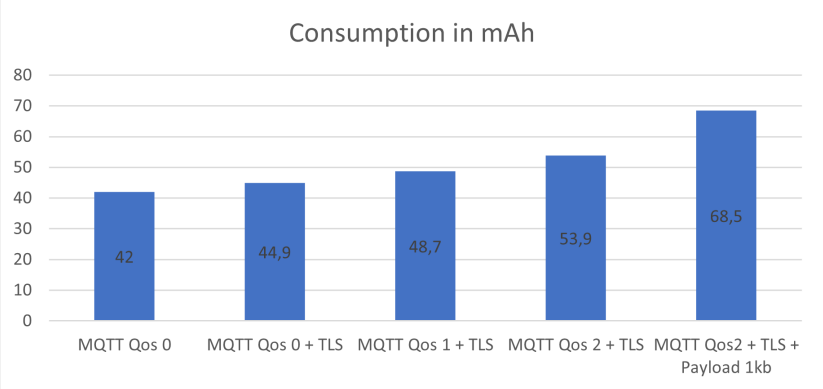

Figure 14: MQTT Energy Consumption

Following the same steps of conversions and calculations applied to CoAP, the following results were obtained for the experiments with MQTT:

- For MQTT QoS 0 the cost was: 0.09979kWh which costs 0.1197 reais, almost 12 cents of reais;
- For MQTT QoS 0 + TLS the cost was: 0.1066 kWh which costs 0.128 reais, almost 13 cents;
- For MQTT QoS 1 + TLS the cost was: 0.1157 kWh which costs 0.138 reais, almost 14 cents;
- For MQTT QoS 2 + TLS the cost was: 0.1280 kWh which costs 0.153 reais, 15 cents;
- For MQTT QoS 2 + TLS + Payload 1kb the cost was: 0.1627kWh which costs 0.195 reais, 19 cents.

4.2 Usability Assessment

In this section, an evaluation of the usability of the flow necessary to provide communication through the CoAP and MQTT protocols without security will be made, as well as, with the communication with security, using the TLS and DTLS protocols. Thus, on the client side ESP32 and on the server side the Raspberry Pi using the Mosquitto broker to work with MQTT and LibCoAP to work with CoAP.

Using the examples of experiments carried out in this work, a heuristic analysis of the necessary steps was carried out so that users can configure and use a proposed minimum environment. The objective of this evaluation is to measure the increase in the level of difficulty in creating: self-signed certificates, public and private keys and configuring them both on the client and server sides, in order to provide a message exchange with guaranteed privacy.

In this usability experiment, an analysis was carried out of the necessary steps for a maker user to be able, on his own, to generate the keys and certificates essential to the use of TLS/DTLS. The maker movement, or do-it-yourself movement, is a trend where users create their own devices and consequently generate alternatives to the low privacy solutions of large corporations. Therefore, the analysis of the difference in complexity between the use of protocols without protection and with protection through TLS/DTLS becomes relevant, even considering that some prior basic knowledge about these technologies is required. Figure 15 and Figure 16 show the necessary steps to implement a communication using MQTT and MQTT + TLS, respectively.

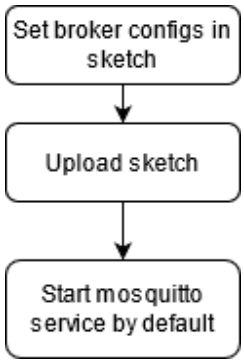

Figure 15: MQTT Flow

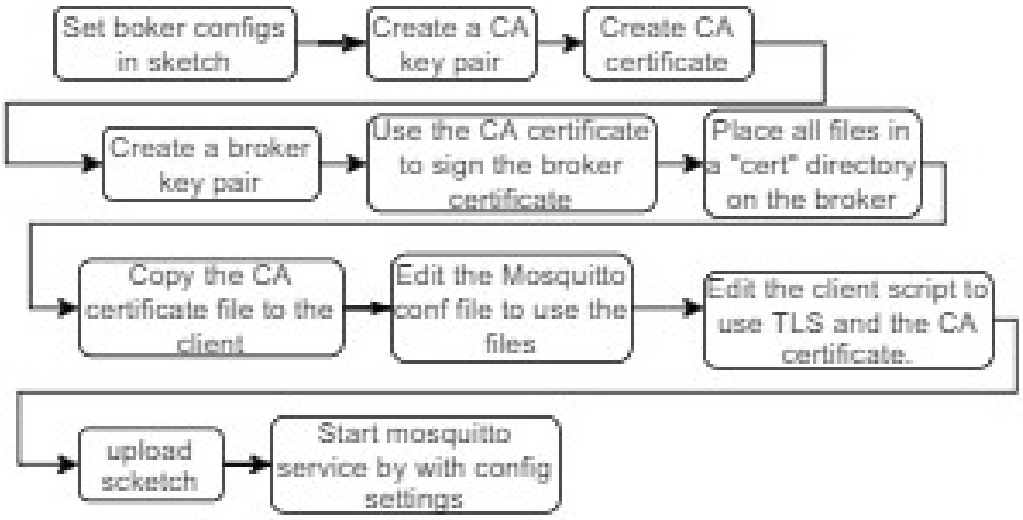

Figure 16: MQTT+TLS Flow

The first notable difference is in the number of steps, the first workflow had only three steps while the second one has eleven steps. To measure the complexity of the flows, a heuristic evaluation was performed based on the work of [NB13], the work of [TT06] and the Single Ease Question (SEQ) metric. In this way, a complexity score was used for each task, in this method from 1 to 5, where 1 would be little complex and 5 very complex. The Table 1 shows the distribution of scores in each of the tasks in the flows. The steps to configure a connection with a digital certificate in the CoAP protocol were the same, but replacing the broker with the server application.

Table 1: MQTT and CoAP usability score.

Steps without security	**Points**	**Steps with safety**	**Points**
Add broker/server/server URL on client	2	Add broker/server URL on client	2
Load firmware on client	1	Create self-signed key pair	3
Start broker/server	1	Create broker/server key pair	3
		Use self-signed certificate to sign broker/server certificate	4
		Add the keys and certificate files to the certs directory	1
		Copy public key to client firmware	1
		Edit broker/server settings to use TLS	4
		Edit client firmware to use TLS	3
		Load firmware on client	1
		Start the broker/server with custom settings	2
Total	4		24

4.3 Response Time Assessment

For the evaluation of the response time, a total of 8 experiments were carried out, reflecting 8 scenarios, each one with 5 samples and each sample had 100 observations of the response time. These experiments resulted in a total of 500 observations in each scenario, allowing to properly determine the distribution of the sample mean. This procedure was performed to evaluate the response time of a request from the client to the server, in the case of CoAP, or a request made from a client publishing in an MQTT broker and receiving the response by subscribing to the same published topic.

In this way, the scenarios where the request was made without the security layer in both protocols and with the security layer were analyzed, in order to verify the reliability of the information when applied to security. We also tested secure scenarios with the addition of QoS in MQTT and commitable messages in CoAP. Finally, more information was added to the payload, aiming to analyze whether its size could make secure communication unfeasible. The statistical data shown below will be from the 5 samples of each scenario, then a comparative analysis will be demonstrated between the average of the observations of each sample, thus making an average sample, defined as:

$$Average = \frac{Sample1 + Sample2 + Sample3 + Sample4 + Sample5}{5} \tag{1}$$

4.3.1 Comparative analysis of MQTT experiments

After finishing all experiments with the MQTT protocol, collecting the results and generating statistical data, the results were comparatively analyzed between the response times of each scenario. The table Table 2 displays the percentage frequency of observations in each response time category.

Looking at the table, you can clearly see an overhead in response times. As privacy protection is added using TLS, and the level of reliability in packet delivery through QoS levels is increased, the overhead becomes even more evident. In the experiment in which the payload size was increased, an even greater effort was observed to send and receive a message.

The data show that 80% of messages without encryption and Qos 0 quality of service had RTT between 8 and 12 ms and the remaining 20% between 12 and 25 ms. When adding the TLS protection

Table 2: Table of RTT frequency distribution.

Categories	Qos 0	Qos 0 + TLS	Qos 1 + TLS	QoS 2 + TLS	QoS 2 + TLS + Payload 1kb
8-12ms	80%	37%	-	-	-
12-25ms	20%	63%	81%	1%	-
25-50ms	-	-	-	2%	-
50-70ms	-	-	4%	-	-
70-100ms	-	-	1%	-	-
100-200ms	-	-	3%	-	-
200-350ms	-	-	11%	97%	79%
350-500ms	-	-	-	-	17%
500-700ms	-	-	-	-	2%
700-100ms	-	-	-	-	2%

layer, in the case of QoS 0 + TLS, only 37% had a response between 8 and 12 ms, the other 63% between 12 and 25 ms. Then, when setting the reliability level to QoS 1, keeping the TLS, there were no messages with a response time less than 12 ms and 81% of the messages returned between 12 and 25 ms. Still keeping the TLS and setting the QoS level to 2, the messages were returned, in almost all the observations (97%) between 200 and 350 ms. Finally adding a significant amount of data to the packet, almost 80% of the measurements were between 200 and 350 ms, however 21% were over half a second.

Considering only the averages of RTT in each experiment, the unsecured MQTT with zero quality of service suffers an overhead of 14.8% compared to the MQTT with Qos 0 and TLS, which in turn is 392.8% more faster than MQTT with QoS 1 and TLS, and this is 531.3% faster than MQTT with Qos 2 and TLS, which ultimately is 124.7% faster than this same configuration but with a payload of 1024 bytes. However, the average is a very sensitive index to the extremes and when modifying the QoS level, it is noticed that some observations are quite different from the others, in statistics this is called outliers. These discrepancies influence the value of the mean, making it an index that does not reflect most of the observations. The boxplot plot of Figure 17, which considers the median and the quartiles, clarifies the real distribution of the samples.

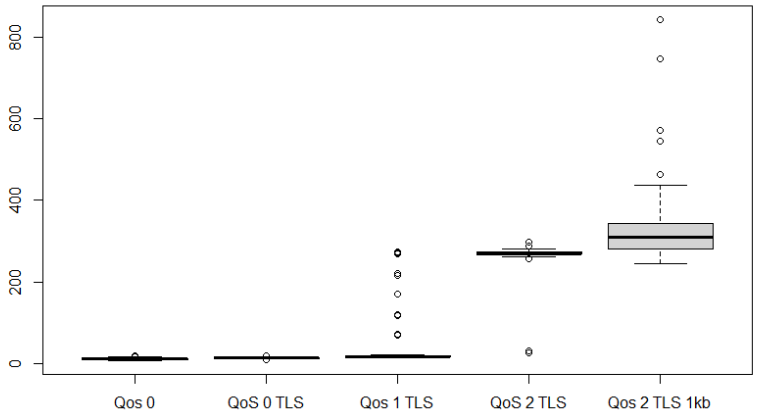

Figure 17: Boxplot MQTT

Based on the graph, where the darkest lines represent the median and the points are the outliers, it becomes evident that the median is the most adequate index to analyze the data, and the greatest overhead occurs from QoS 2 onwards.

4.3.2 Comparative analysis of CoAP experiments

When completing all experiments with the CoAP protocol and collecting the results and generating statistical data, the results were comparatively analyzed between the response times of each scenario. The Table 3 displays the percentage frequency of observations in each response time category. From it, it is possible to clearly identify an increase in the response time as a layer of protection is added, due to the use of DTLS, and the confidentiality level of the data transmitted in the packets is increased. However, there is an overhead and, in addition, in the experiment in which data was added to the payload size, an even greater effort was noticed to send and receive a message.

Table 3: Table of the frequency distribution of RTT CoAP.

RTT Categories	**CoAP**	**CoAP + DTLS**	**CoAP + DTLS + Payload 1kb**
4-8ms	73%	21%	-
8-16ms	26%	77%	39%
16-32ms	1%	2%	54%
32-64ms	-	-	5%
64-128ms	-	-	2%

Thus, the data show that 73% of unencrypted messages had an RTT between 4 and 8 ms and only 1% of the observations reached between 16 and 32 ms. When adding the protection layer, only 21% had a response between 4 and 8 ms, observations between 8 and 16 ms were 77% of the occurrences. Finally, adding a significant amount of data to the packet, there was no RTT less than 8 ms, 93% of measurements were between 8 and 32 ms, 5% between 32 and 64 ms and only 2% between 64 and 128 ms .

Thus, considering that in the three experiments using CoAP the values of central tendencies indices presented similar values, the mean was considered as the representative index of the samples. The boxplot plot of Figure 18 shows the distribution of the samples.

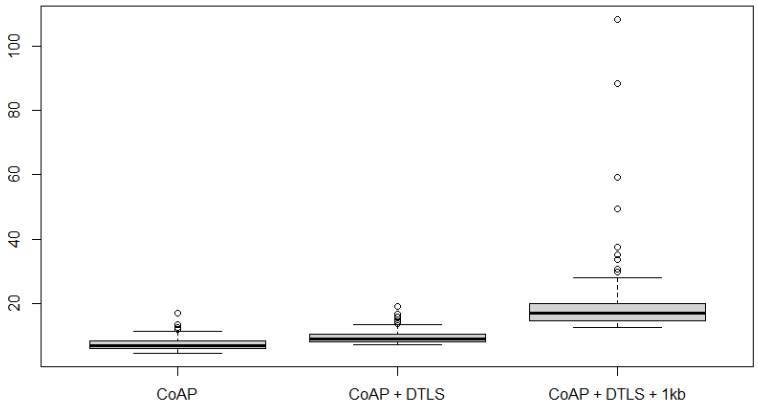

Figure 18: Boxplot CoAP

5 Conclusion

This work aimed to verify the feasibility of using privacy protection mechanisms in Application Layer protocols used in IoT networks in the smart home context. To properly meet the purpose of this study, it was necessary to analyze the communication using these protocols through the criteria of usability, performance, and energy efficiency. Thus, this protection has the purpose of guaranteeing privacy in the home environment, ensuring ease of use, good performance, and low cost. For this, the following specific objectives were defined:

- Survey the main privacy-related vulnerabilities in the Application Layer of IoT networks;
- Survey the main Application Layer protocols that run on TCP and UDP and your confidentiality options for smart home networks;
- Analyze the possible impacts of vulnerable traffic in the smart home context;
- Verify, by prototyping, the evaluation of privacy mechanisms, using a methodology for measuring response time, usability and increase in electric current consumption;
- Determine the feasible scenario so that the application layer can provide privacy on smart home devices.

First, a Systematic Mapping of Literature was carried out. In this way, works that used a mechanism that provided greater privacy or detected vulnerabilities were included in the Application Layer of smart home networks. In addition, works were sought that discussed the main IoT protocols used for home automation. This mapping was necessary to support the experiments that would be necessary to analyze the problem raised, which derived the objectives of this research. Then, a Systematic Review of the Literature was carried out and as a selection criterion, the works that used evaluation methods of the main protocols and their security layers were included. At the end of the review, it was decided to analyze an Application Layer protocol that used TCP. Thus, MQTT and another that used UDP to transport the packets were listed, and in this case, CoAP was defined.

The experiments were not carried out in order to compare MQTT with CoAP, but what would be the overhead on each of them when using TLS and DTLS, respectively, as their protective layers. The results showed that using CoAP, the increase in energy consumption in each connection, in the period of one month, was less than 1 cent of reais when using DTLS with a small payload. In addition, the difference between the adoption of security in communication in the scenario with the highest overhead (CoAP + DTLS + 1 kb payload) was 2 cents in a month, when compared to CoAP without security, which characterizes an increase of 20%, but financially derisory. In the experiments with MQTT, when comparing the use of MQTT with QoS 0 and without security with the most expensive scenario (MQTT with QoS 2 + TLS + 1 kb payload, the consumption increase was 7 cents of reais in one month. from 12 cents to 19 cents, an increase of 58.33% in a month of use of each device. Although in terms of percentage the increase has been significant, the cost of 7 cents to have security and quality of service is still viable.

To ensure that the increase in privacy protection does not make the use of security protocols unfeasible, it is necessary to analyze whether adding protection does not add any difficulty for the end user to use, which is intrinsic to the process of adding layers. To this end, a difficulty scoring methodology was created for each unsecured configuration step and compared with the steps for generating keys and certificates of asymmetric cryptography, as well as the configurations inherent to the process. The results showed that for both MQTT and CoAP, the difficulty level in points for unsafe use was 4 points, and when using security it increased to 24 difficulty points. This increase becomes significant for those who will provide the security solution, however, once configured, the protocols under protection are completely transparent to the end user. Thus, this makes the use of security viable when analyzed through the prism of usability.

Furthermore, although feasible through the observations of the criteria already analyzed, the response time of requests was also statistically evaluated. Thus, the purpose of this evaluation was to elucidate the questions about the use of TLS and DTLS as security of the MQTT and CoAP protocols in a smart home environment. The MQTT response time experiments showed that when using the sample medians as an index, it was observed:

1. Just adding TLS at Qos level 0, the overhead is 17%;
2. From QoS 0 with TLS to Qos 1 with TLS the overhead was 33%;
3. From QoS 1 with TLS to Qos 2 with TLS the overhead was 1500%;
4. Adding the payload to 1024 bytes and using Qos 2 with TLS the overhead was 15%.

Given the above, even though the difference in RTT time from the fastest to the slowest scenario was considerably high, analyzed through the prism of the context of a smart home, the response time in all cases was less than 1s. Thus, it is feasible to use security through TLS with guaranteed packet delivery and a payload of 1024 bytes.

The experiments performed for the CoAP protocol showed that:

1. Just adding the DTLS , the overhead is 32.43%;
2. Adding the payload to 1024 bytes and using DTLS, the overhead was 178%, compared to the previous one;
3. Comparing the use of DTLS, and DTLS + 1kb payload the overhead was 110%

That said, even though the difference in RTT time from the fastest to the slowest scenario was considerably high, analyzed from the point of view of a smart home, the response time in all cases was also less than one second. In this way, it is feasible to use security through DTLS with packet delivery confirmation and a payload of 1024 bytes.

5.1 Contributions

The main contributions of this work were to analyze the energy consumption through an isolated method of benchmarking and to correlate usability with the increase of security in the development process. In addition, calculate the (financial) cost of energy consumption of a secure device in the smart home environment and provide an analysis of the feasibility of using the security layer in the home environment through statistical analysis, using a low-cost and widely diffused SoC.

5.2 Future Works

In this section, some possibilities for future work are raised to continue and deepen the study:

- Analyze the feasibility (performance, cost and usability) using different cipher suites of TLS and DTLS;
- Analyze whether the use of Cryptographic Hardware Acceleration (CHA), a cryptographic acceleration core, in the ESPs32 architecture would lead to more performant results than when compared to other microcontrollers without this technology;
- Implement new experiments using other Application Layer protocols such as: XMPP, AMQP, DDS among others;
- Carry out a more detailed analysis on the server or broker side, as this work focused on the bottleneck on the client side of applications.

References

[DR15] Shawn DuBravac and Carlo Ratti. The internet of things: evolution or revolution? *AIG White Paper*, 2015.

[Fou21] Raspberry Pi Foundation. *Raspberry Pi 3 Model B*, 2021.

[Gar19] Gartner says that iot is top 8 supply chain technology trends in 2019, 2019.

[Int21] Intelbras. *Datasheet Twibi Giga*, 2021.

[JKH20] Haojian Jin, Swarun Kumar, and Jason Hong. Providing architectural support for building privacy-sensitive smart home applications. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, pages 212–217, 2020.

[Kin03] Nicola King. Smart home–a definition. *Intertek Research and Testing Center*, pages 1–6, 2003.

[Mos21] Mosquitto. Eclipse mosquitto, January 2021.

[Nan21] Arduino Nano. *Arduino Nano*, 2021.

[NB13] Jakob Nielsen and Raluca Budiu. *Mobile usability*. MITP-Verlags GmbH & Co. KG, 2013.

[PKBT18] Matthias Pohl, Janick Kubela, Sascha Bosse, and Klaus Turowski. Performance evaluation of application layer protocols for the internet-of-things. In *2018 Sixth International Conference on Enterprise Systems (ES)*, pages 180–187. IEEE, 2018.

[PMY19] Sandy Suryo Prayogo, Yulisdin Mukhlis, and Bayu Kumoro Yakti. The use and performance of mqtt and coap as internet of things application protocol using nodemcu esp8266. In *2019 Fourth International Conference on Informatics and Computing (ICIC)*, pages 1–5. IEEE, 2019.

[PV19] Yanina Protskaya and Luca Veltri. Broker bridging mechanism for providing anonymity in mqtt. In *2019 10th International Conference on Networks of the Future (NoF)*, pages 110–113. IEEE, 2019.

[PWDC18] Pat Pannuto, Wenpeng Wang, Prabal Dutta, and Bradford Campbell. A modular and adaptive architecture for building applications with connected devices. In *2018 IEEE International Conference on Industrial Internet (ICII)*, pages 1–12. IEEE, 2018.

[RHPV17] Shahid Raza, Tómas Helgason, Panos Papadimitratos, and Thiemo Voigt. Securesense: End-to-end secure communication architecture for the cloud-connected internet of things. *Future Generation Computer Systems*, 77:40–51, 2017.

[SAR15] Muhammad Suryanegara, Muhamad Asvial, and Naufan Raharya. System engineering approach to the communications technology at unmanned aircraft system (uas). In *2015 IEEE International Symposium on Systems Engineering (ISSE)*, pages 475–480. IEEE, 2015.

[SBZB19] Farhan Siddiqui, Jake Beley, Sherali Zeadally, and Grant Braught. Secure and lightweight communication in heterogeneous iot environments. *Internet of Things*, page 100093, 2019.

[Tex22] Instruments Texas. *INA219 Data sheet*, 2022.

[TT06] Donna Tedesco and Tom Tullis. A comparison of methods for eliciting post-task subjective ratings in usability testing. *Usability Professionals Association (UPA)*, 2006:1–9, 2006.