Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

Implementing Internet of Things Protocols

Elias Berglin

MID SWEDEN UNIVERSITY
Department of Information Systems and Technology (IST)

Main field of study: Computer Engineering
Credits:
Semester, year: VT, 2022
Supervisor: Stefan Forsström
Examiner: Mikael Gidlund

Abstract

The abstract acts as a description of the reports contents. This allows for the possibility to have a quick review of the report and provides an overview of the whole report, i.e. contains everything from the objectives and methods to the results and conclusions. Examples: "The objective of this study has been to answer the question.... The study has been conducted with the aid of.... The study has shown that..." Do not mention anything that is not covered in the report. An abstract is written as one piece and the recommended length is 200-250 words. References to the report's text, sources or appendices are not allowed; the abstract should "stand on its own". Only use plain text, with no characters in italic or boldface, and no mathematical formulas. The abstract can be completed by the inclusion of keywords; this can ease the search for the report in the library databases.

Keywords: Human-computer-interaction, XML, Linux, Java.

Acknowledgements

Acknowledgements or Foreword (choose one of the heading alternatives) are not mandatory but can be applied if you as the writer wish to provide general information about your exam work or project work, educational program, institution, business, tutors and personal comments, i.e. thanks to any persons that may have helped you. Acknowledgements are to be placed on a separate page.

Table of Contents

Abbreviations

RTT	Round Trip Time
MQTT	Message Queue Telemetry Transport
CoAP	Constrained Application Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IoT	Internet of Things

1 Introduction

1.1 Background and problem motivation

IoT is a growing market in the world with MQTT and CoAP emerging as the dominant protocols for communication. Different IoT systems might need to communicate with one another event tho they might use different protocols. To do this a translator is needed to be able to translate CoAP to MQTT.

1.2 Overall aim

The aim for this project is to implement a system that translates responses from a CoAP server to a MQTT client running on a mobile application. The scenario is to monitor the CoAP server system usage and prepare a system for future integration with temperature sensors.

1.3 Concrete and verifiable goals

The goals for this project

- Combine a MQTT-client with a CoAP-client for translation.
- Implement a CoAP-server capable of serving CPU, memory usage and mock temperature sensors.
- Create a user application capable of displaying the information.
- Perform measurements on the system for evaluation.

1.4 Scope

This project will focus on implementing an end to end system from a CoAP-server to MQTT-client through a MQTT-broker. The only measurement that I will be conducting is the RTT from the client application to the CoAP-server from this the mean, min max and standard deviation will be calculated. From the total time for x amount of requests the request per second will be calculated.

1.5 Outline

Chapter 2 will go into theory about the technology used. Chapter 3 will motivate the technology choices as well as describe the system. Chapter 4 will go through the construction process for the system. Chapter 5 will present the result from the measurements. Chapter 6 will discuss the result as well as draw conclusion from this project.

1.6 Contributions

This project and report was done by me. I would like to thank a couple of people and companies for their open source packages that made this project possible. First I would like to thank plgd for their COaP package for Go. I would also like to thank the Eclipse foundation for their Go package for MQTT. Lastly I would like to thank Steve Hamblett for his MQTT package for Dart.

2 Theory

2.1 CoAP

CaAP is a communication protocol over UDP aimed for lower powered computers. It is often used by small microcontrollers that only have limited storage and RAM available. CoAP uses the client server model. The server accepts requests and then responds. The server also provides methods for clients to discover resources on the server. CoAP was designed to be easily translated to HTTP and uses similar request to HTTP, for example GET and POST. [1]

2.2 MQTT

MQTT is a communication protocol over TCP. It is lightweight and provides an open communication channel between devices. The MQTT system consists of one broker and many clients. A client can subscribe and publish data to topics. The broker then makes sure to distribute the data to all subscribers. The protocol was designed in the 1990s for use in oil pipeline monitoring. Today it is more commonly used in IoT applications. [2]

2.3 Flutter

Flutter is a framework for building multiplatform applications. Flutter is open source and developed by Google. Flutter applications are written in dart and the framework can compile the dart code to machine code for both x86 and ARM. It can also combine the code to JavaScript for web compatibility. [3] As of version 2.0 of Flutter the following platforms are supported: [4]

- Android
- IOS
- Web
- Windows (beta)
- Windows UWP (alpha)
- Linux (beta)
- MacOS (beta)

3 Methodology

3.1 System overview

The structure of the IoT system including the planned devices can be found in figure 1.

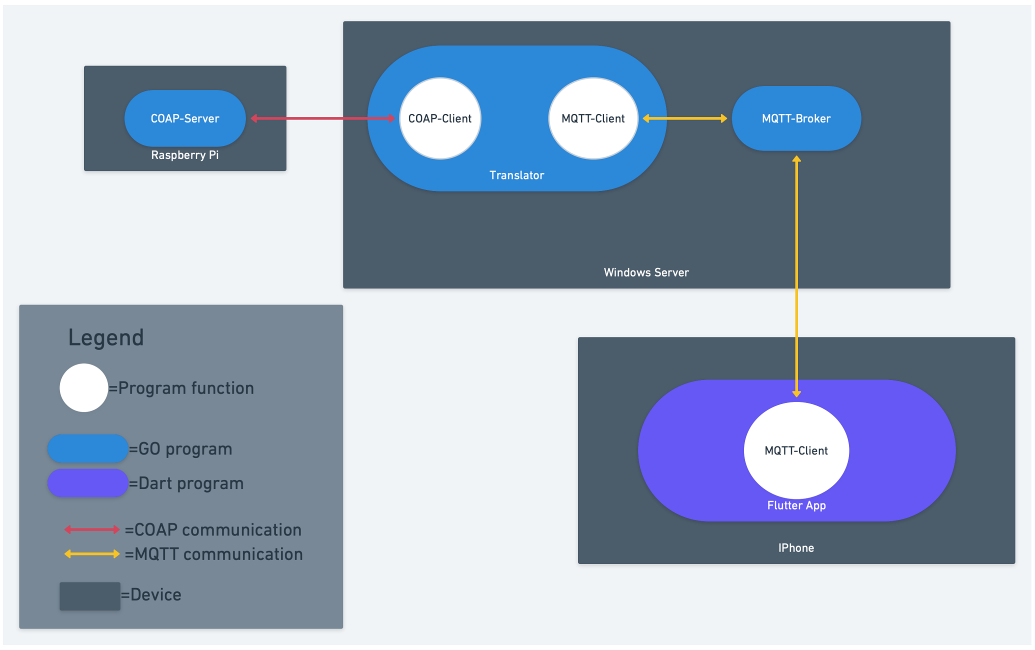

Figure 1: The planned communication scheme. See legend for explanation

3.2 Choice of technologies

Two components of the system was already developed as part of two previous projects. These are the CoAP-Client and the MQTT-broker. These where implemented in Go for its excellent concurrency. This does not affect the CoAP-client, but it is useful for the MQTT-broker for handling multiple connections. Another perk of Go is how easy it is to create the different connection types. Lastly Go has a built-in datatype for Bytes and that came in handy when reading and writing data from sockets.

Because of the Go implementation on the previous parts it was only natural to have the CoAP-server in Go. To create a server an external library was used [5]. The translator needs to combine both a CoAP-client and a MQTT-client. To be compatible with the created CoAP-Client the MQTT-client needs to be written in Go. For this I also used a Go library [6]. Lastly

Flutter will be used to create the user interface. Flutter was chosen for its capability to be compiled to multiple platforms. This allows for the app to be installed on multiple devices' whiteout needing to rewrite the application. To communicate with the MQTT-broker an external dart library was used [7].

3.3 System description

The primary purpose of the server is to collect system data more specifically CPU and RAM usage. A client will then be able to request this information. Additionally, the server can create mock temperature sensors. A sensor has a location, power status and a temperature. A client can request temperature, create a sensor, change power status and remove a sensor.

The translators have two purposes. It will collect data from the CoAP-server and publish using the MQTT-client. It will also react to published messages from other MQTT-clients and create CoAP-request accordingly. Ex. the Flutter app wants to create a sensor. The translator can then convert a publishing by the application to a CoAP POST-request.

The MQTT-broker is a standard MQTT-broker implemented using Golang. It will handle the subscription and publish messages, creating a communication channel between the translator and the Flutter application.

Lastly the Flutter application will be multi-platform application acting as the frontend of the IoT system. The app will have the following capabilities.

- Display CoAP-server system information.
- Display temperature sensor information.
- Create temperature sensors.
- Turn off specific temperature sensors.
- Delete a Temperature sensor.
- Run a benchmark.
- Display benchmark results.

The system requires one active CoAP-server, one active translator, one active MQTT broker and one or more Flutter applications. This implementation will put the CoAP-server on a Raspberry Pi running Raspbian and bundle the MQTT-broker and the Translator on a Server running Windows Server. The Flutter application will run on at least IOS but should be able to run on any operating system supported by Flutters build system.

3.4 Evaluation

To evaluate the system the RTT will be measured between the user Application and the CoAP server. This will be done using a benchmark function inside the Flutter application. Three benchmark runs will be done adding a new instance of the flutter application with each run to simulate different load on the system. The Flutter application will output the following metrics:

- Requests per second
- Average RTT
- Max RTT
- Min RTT

4 Implementation

4.1 CoAP-server

As mentioned in chapter 3.2 the CoAP-server is implemented in Go using an external library [5]. The library implements CoAP similarly to how Go normally handles HTTP. To start serving CoAP a router is created for routing request to different locations. A middleware was created to enable logging. The first rout added was the rout to get the mock temperature sensors. To create a rout an endpoint and a handler function needs to be provided to the router. In figure 2 we can see the code snipped to create the router, add the middleware and lastly creating the first route.

```go
r := mux.NewRouter()
r.Use(loggingMiddleware)
r.Handle("/temp", mux.HandlerFunc(handleTemp))
```

Figure 2: Go code snipped showing creating the CoAP router, adding a logging middleware and creating the handler for temperature

The temperature handler decodes the incoming CoAP message to figure out what type of request it was. It also checks if the message has a payload. All the request types to this endpoint requires a payload. Figure 3 shows the code snippet for the initial handling of a request to the temperature endpoint. If the body does not exist it ends the request there by responding with bad request.

```go
method := r.Code.String()
body := make([]byte, 128)
if r.Body == nil {
    w.SetResponse(codes.BadRequest, message.TextPlain, bytes.
       NewReader([]byte(string("No body included"))))
    return
}
n, _ := r.Body.Read(body)
body = body[:n]
```

Figure 3: Code showing initial handling of the temperature request

The handler can handle 4 different request types: GET, POST, PUT, DELETE.

GET responds with the current temperature of the processor if it exists otherwise it returns not found. Because the sensor is simulated, after the

GET request it either increase or decrease the temperature of the sensor by a random amount between 1-3°C. Figure 4 shows the structure for a temperature sensor.

```go
type TemperatureSensor struct {
    Status      bool
    Temperature int
    Location    string
}
```

Figure 4: Structure for a mock temperature sensor

POST creates a new temperature sensor. The request requires the payload to contain both the location of the sensor and the initial power status. When the request is received it creates a new instance of a temperature sensor. It assigns the values received and randomizes a temperature between -20-30°C. If the request is successful it responded with status code created.

PUT request are used to switch power status. The request requires the body to contain the new power status and the name of the sensor to be changed. If the sensor exists it changes its power status and responds with status Changed.

DELETE request removes a sensor if the payload includes the sensor name. If the request is successful it responds with Deleted.

For clients to be able to discover what temperature sensors is available the /all endpoint was created. This endpoint only accept GET request and uses Go's built in JSON library to convert the internal map of sensors to a JSON string to include in the response. The code implementation can be found in figure 5.

```go
r.Handle("/all", mux.HandlerFunc(func(w mux.ResponseWriter, r
    *mux.Message) {
    data, _ := json.Marshal(thermostats)
    w.SetResponse(codes.Content, message.TextPlain, bytes.
        NewReader(data))
}))
```

Figure 5: Creation of the /all endpoint

The last endpoint is /pi which enables clients to gather information about the CoAP-server's processor and memory utilization. The server collects this data through the Linux proc files. CPU information is gathered through the proc/stat file while the memory is gathered through the proc/meminfo file. I used a library to help width parsing these files[8]. A separate thread reads the proc files every second. For CPU usage it uses the last calculated value to derive the current usage in percentage. For the memory the proc/meminfo contains both free memory and total memory. From this the function derives the current usage in the format used/total.

4.2 Translator

The translator builds on a previous implemented CoAP client in Go. To translate from CoAP to MQTT a MQTT library was added to the project[6]. The translator will poll data from the CoAP and then publish it to the MQTT-broker. It will also subscribe to certain MQTT messages to be able to convert them to CoAP request. The response from these messages will then be published.

The first thing the Translator does is send a request to the CoAP-server to retrieve the available sensor. This is to keep an internal map of what sensor are online to prevent the Translator to pull unnecessary data. After it retrieves the information about the temperature sensors it starts the connection to the MQTT broker and creates some constant subscriptions. When subscribing the MQTT library requires callback functions that will fire when a message is received to that topic. The connection code can be found in figure 6

```go
populateSensors()
var broker = "localhost"
var port = 1883
opts := mqtt.NewClientOptions()
opts.AddBroker(fmt.Sprintf("tcp://%s:%d", broker, port))
opts.SetClientID("Coap Translator")

opts.SetDefaultPublishHandler(messagePubHandler)
opts.OnConnect = connectHandler
opts.OnConnectionLost = connectLostHandler
opts.SetPingTimeout(time.Second * 60)

client := mqtt.NewClient(opts)

if token := client.Connect(); token.Wait() && token.Error()
    != nil {
    panic(token.Error())
}
```

Figure 6: Connection code for the Go MQTT client

The constant subscription are: all, home/add, home/delete and home-

/change. The all topic is used for communication to the MQTT network the available temperature sensors. If an MQTT client publish a message with the structure GET:*id*. The Translator will get the information using the CoAP client and then forward the response to MQTT network via a publish message to the topic all/*id*. The function for parsing all the publish messages can be found in figure 7

```go
func allCallback(c mqtt.Client, m mqtt.Message) {
    message := string(m.Payload())
    fmt.Println(message)

    if strings.Contains(message, "GET") {
        split := strings.Split(message, ":")
        if len(split) != 2 {
            return
        }
        COAPmsg := createGet("all", "")
        payload := string(sendCreatedCoap(COAPmsg))
        fmt.Println(payload)
        tok := c.Publish("all/"+split[1], 0, false, payload)
        tok.Wait()
    }
}
```

Figure 7: Function for parsing the /all endpoint

The home/add subscription is used to create a sensor. The Translator listens for publish messages and then relays the information to the CoAP-server. Figure 8 shows a code for the callback function to the home/add subscribe.

```go
func addCallback(c mqtt.Client, m mqtt.Message) {
 message := string(m.Payload())
 msg := createPost("temp", message)
 payload := string(sendCreatedCoap(msg))
 fmt.Print(payload)
 populateSensors()
}
```

Figure 8: Function for parsing the home/add publish

The home/change and home/delete functions similarly to home/add. home/change is used to change the power status of a temperature sensor and the home/delete is used to delete a temperature sensor. For all of these function the Translator repopulates the internal map to be up-to-date on what sensors exist and their power status

The main loop of the Translator is to first pull the temperature sensors if they are online. It then publishes this temperature to home/*id*. After it has taken care of the temperature it moves on to the CPU and memory information. It creates a request to the CoAP-server and then splits the information in to two publishes pi/cpu and pi/mem. When it has published the data it sleeps for 10 seconds and then starts from the top. The code for the main loop can be found in figure 9

```go
for {
    for k, online := range sensors {
        if online {
            msg := createGet("temp", k)
            p := sendCreatedCoap(msg)
            tok = client.Publish("home/"+k, 0, true, p)
            tok.Wait()
        }
    }
    msg := createGet("pi", "")
    p := sendCreatedCoap(msg)
    //fmt.Println(string(p))
    tok = client.Publish("pi/cpu", 0, true, strings.Split(
        string(p), ":")[0])
    tok.Wait()
    tok = client.Publish("pi/mem", 0, true, strings.Split(
        string(p), ":")[1])
    tok.Wait()
    time.Sleep(time.Second * 10)
}
```

Figure 9: The main loop of the Translator

4.3 Flutter mobile application

5 Results

The results chapter is included when you have produced a systematic study, i.e. an evaluation of a program that you have developed, which is required for C - and D-level diploma work. In the results chapter objective results of the empirical study are presented. Keep in mind that possible comments in this chapter should only be used for clarification. Your own views and subjective (personal) comments belong in the chapter conclusion/discussion.

Strive to present the results, for example measurement-, calculations- and/or the simulation result, in a form that is as lucid and easily understandable as possible. The results are preferably presented in diagrams or tables. Accounts of interviews can be summarised, but may include concrete examples supporting your work.

Extensive results, for example complete summaries of survey results, large tables and long mathematical deductions, are placed in the appendices.

6 Conclusions / Discussion

The conclusion/discussion (choose a heading) is a separate chapter in which the results are analysed and critically assessed. At this point your own conclusions, your subjective view, and explanations of the results are presented.

If this chapter is extensive it can be divided up into more chapters or subchapters i.e. one analysis or discussion chapter with explanations of and critical assessment of the results, a concluding chapter where the most important results and well supported conclusions are discussed and to sum it up a chapter with suggestions for further research in the same area. In this chapter it is of vital importance that a connection back to the aim of the survey is made and thus the purpose is pointed out in a summary and analysis of the results.

In this chapter you should also include answers to the following questions: What is the project's news value and its most vital contribution to the research or technology development? Have the project's goals been achieved? Has the task been accomplished? What is the answer to the opening problem formula? Was the result as expected? Are the conclusions general, or do they only apply during certain conditions? Discuss the importance of the choice of method and model for the results. Have new questions arisen due to the result?

The last question invites the possibility to offer proposals to others relevant research, i.e. proposal points for measures and recommendations, points for continued research or development for those wishing to build upon your work. In technical reports on behalf of companies, the recommended solution to a problem is presented at this stage and it is possible to offer a consequence analysis of the solution from both a technical and layman perspective, for example regarding environment, economy and changed work procedures. The chapter then contains recommended measures and proposals for further development or research, and thus to function as a basis for decision-making for the employer or client.

6.1 Ethical and Societal Discussion

You will need to include a discussion on ethics, societal impact, and considerations.

6.2 Future Work

You should also explain potential future work based on your work.

References

[1] Zach Shelby, Klaus Hartke, and Carsten Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. June 2014. DOI: 10.17487/RFC7252. URL: `https://rfc-editor.org/rfc/rfc7252.txt`.

[2] Michael Yuan. *Getting to know MQTT*. URL: `https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/` (visited on 01/04/2022).

[3] Google. *Flutter*. URL: `https://flutter.dev` (visited on 01/03/2022).

[4] Google. *Multi-Platform*. URL: `https://flutter.dev/multi-platform` (visited on 01/03/2022).

[5] plgd. *go-coap*. Dec. 16, 2021. URL: `https://github.com/plgd-dev/go-coap` (visited on 01/02/2022).

[6] Eclipse Foundation. *Eclipse Paho MQTT Go client*. Dec. 28, 2021. URL: `https://github.com/eclipse/paho.mqtt.golang` (visited on 01/02/2022).

[7] Steve Hamblett. *mqtt_client*. Jan. 2, 2022. URL: `https://github.com/shamblett/mqtt_client` (visited on 01/02/2022).

[8] Yo-An Lin. *goprocinfo*. Jan. 30, 2021. URL: `https://github.com/c9s/goprocinfo` (visited on 01/04/2022).

A Source Code