Università degli Studi di Camerino SCUOLA DI

SCIENZE E TECNOLOGIE Corso di Laurea in

Informatica (Classe L-31)

PICOMP vs MQTT: benchmarking IoT protocols in a COVID-19 case study

Laureando

Francalancia Simone
Matricola N. 098538

Relatore

Prof. Mostarda Leonardo

Correlatore
Ing. Pagnotta Fabio

A.A. 2019/2020

INDEX

1 – Introduction:

The severe acute respiratory syndrome Coronavirus-2 (SARS-CoV-2) is the name given to the new coronavirus of 2019.

COVID-19 is the name given to the disease associated with the virus.

SARS-CoV-2 is a new coronavirus strain that has not previously been identified in humans which is deadly for specific groups of people such as older ones and for whom is not in good health conditions.

COVID-19 virus usually spreads with small droplets that can remain suspended mid-air thus making gathering and meeting places dangerous, especially indoor locals with a temperature around 17 °C and a relative humidity around 30%.

The purpose of this thesis work is to analyse indoor air quality with a WSAN network made up of some small IoT devices hooked up to some sensors for collecting environmental data in order to detect if the air in a determinate room is dangerous or not.

The above stated IoT WSAN network is going to be implemented twice with two protocols:

- The first protocol is called MQTT and it was developed by IBM in 1999 which is ideal for connecting small devices.

- The second protocol is called PICO-MP, it is being developed in Unicam by Pagnotta Fabio and it is an ad-hoc tool for WSANs macro-programming.

We are going to compare these two nets to determine which one is the most suitable for our case of study.

2 – Case of study:

Power and resource consumption always represented an issue within wireless sensor and actuator networks (WSANs) on which devices that powers it may only have little computing capabilities or may be running on batteries, thus, various factors influence the performance of a WSAN network, so we are going to benchmark these two protocols and see which one is the most suitable and resource efficient in a COVID-19 air quality case study where cold temperatures and low relative humidity conditions favour the survival and transmission of certain influenza virus and are associated with an increased occurrence of respiratory tract infections.

3 – MQTT:

MQTT (**Message Queue** Telemetry Transport) is an open OASIS standard messaging protocol for the Internet of Things (IoT). The protocol usually runs over TCP/IP and it is designed as an extremely lightweight **publish/subscribe** messaging transport that is ideal for connecting remote devices with a small code footprint (It can even run on 8-bit, 256KB ram controllers like the atmega 2560) and is also suitable for minimal network bandwidth, low power, high latency and high-cost connections situations, therefore it makes it really easy to establish a communication between multiple and different types of devices.

MQTT was created by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 to monitor oil pipelines but today It finds his place in a wide variety of industries, such as home automation, automotive, manufacturing, telecommunications, (still) oil and gas, communications over satellite links and in a wide range of, as mentioned above, small device scenarios.

MQTT is composed of the following elements:

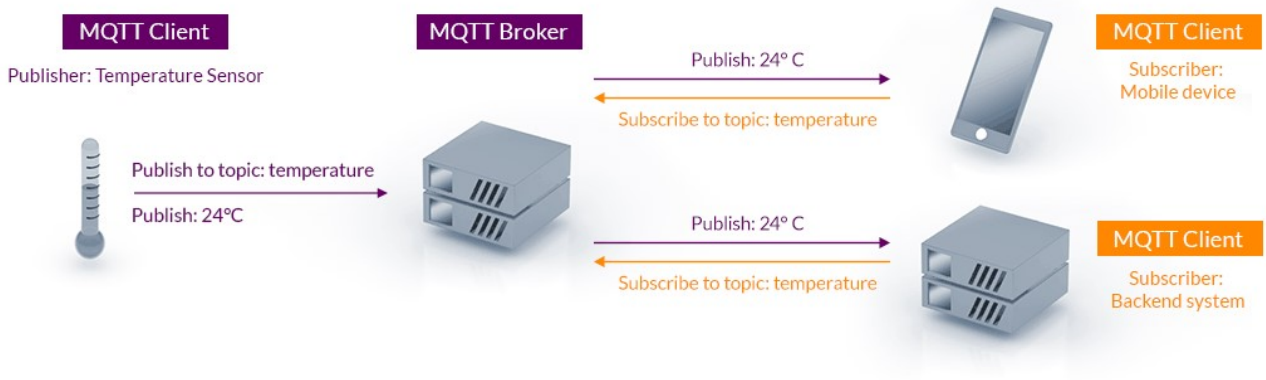
- **Clients:**
    - **Publisher:** node that can publish any kind of data to a specific topic.
    - **Subscriber:** the node that subscribes to topics to read published data.

  Both publishers and subscribers nodes are MQTT clients. They can be any device that is capable of running an MQTT library and has a network connection (for example it can be a micro controller or a raspberry, a pc or a server).

- **Broker:** the node that receives all messages from the sender clients and then routes the messages to the appropriate subscriber clients. The broker acts more or less like a post office and is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the messages the appropriate subscriber clients.

- **Topic:** usually an MQTT topic is a group of strings, called topic levels, separated by the topic level separators (the forward slash), following a hierarchical structure; they are used to specify where to publish messages (in the case of publishers) or to specify where to read incoming messages (in the case of subscribers),

3.1 - topic example



3.2 - mqtt full example

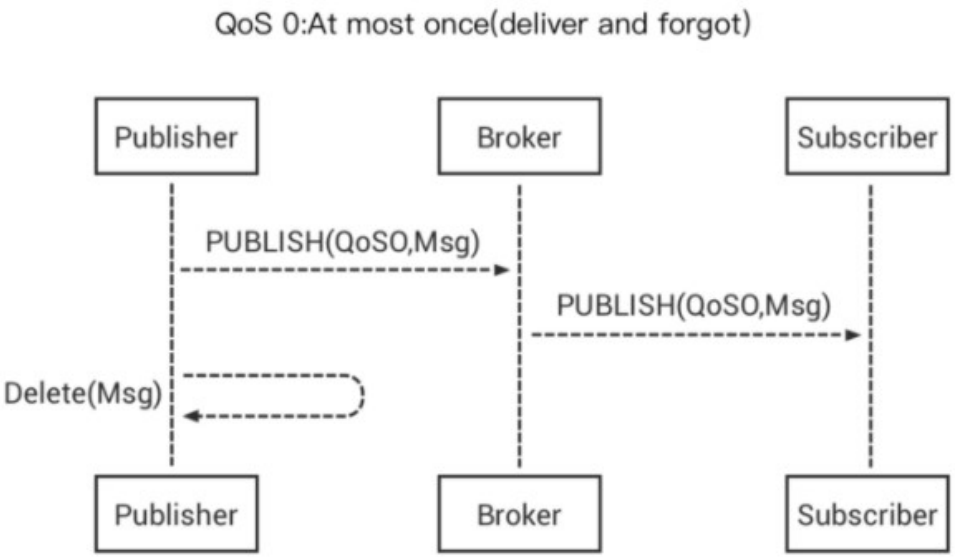
3.1 – MQTT QoS

The standard supports 3 levels of QoS (Quality of Service), bi-directional messaging to handle both device to cloud and cloud to device communication, deterministic message delivery, always/sometimes-connected device scenarios and scalability to support large numbers of IoT devices.

The QoS above mentioned is an “agreement” between the sender and the receiver that guarantees the delivery of a message. To better understand this concept, we need to split it in two parts: there is a message delivery from the publishing client to the broker and a message delivery from the broker to the subscribing client.
In the first part, the client that publishes the message to the broker defines the quality level of the message sent to the broker. In the second part, the QoS is chosen by the subscribing client, and the message from the broker to that subscriber is sent following the new defined rule.

These are the three available QoS levels:

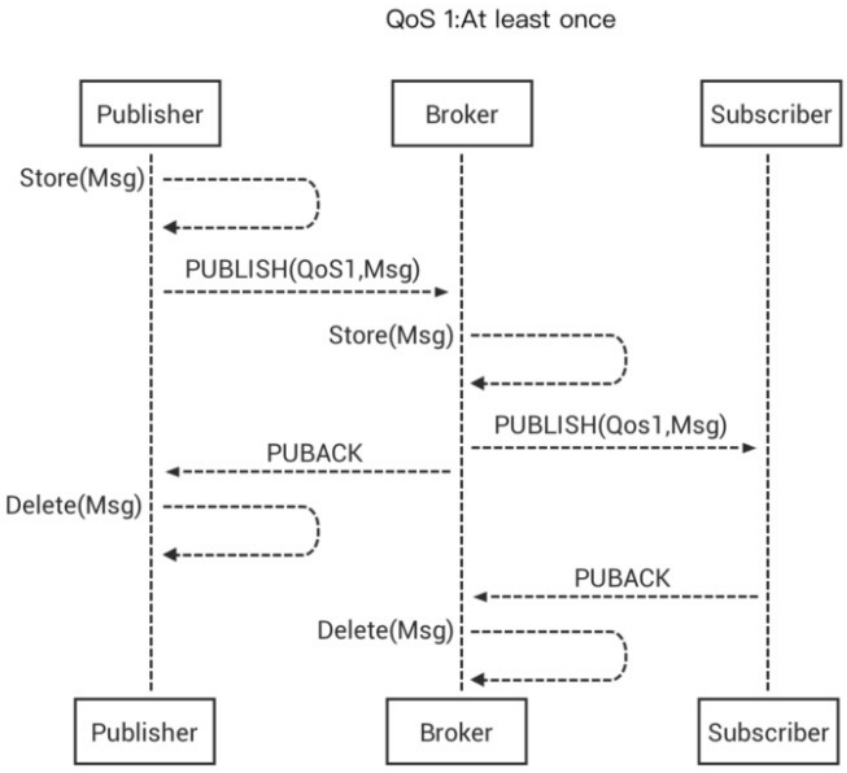
- **0 (at most once):**
  The publisher just sends the data to the broker, and there is no acknowledgement that the message was received (best effort delivery). This is simplest method of sending a message through the net, and should be utilized when the connection between the sender and the receiver is mostly or completely stable and if data loss can be handled without too many problems.



3.3 – QoS 0, at most once

- **1 (at least once):**

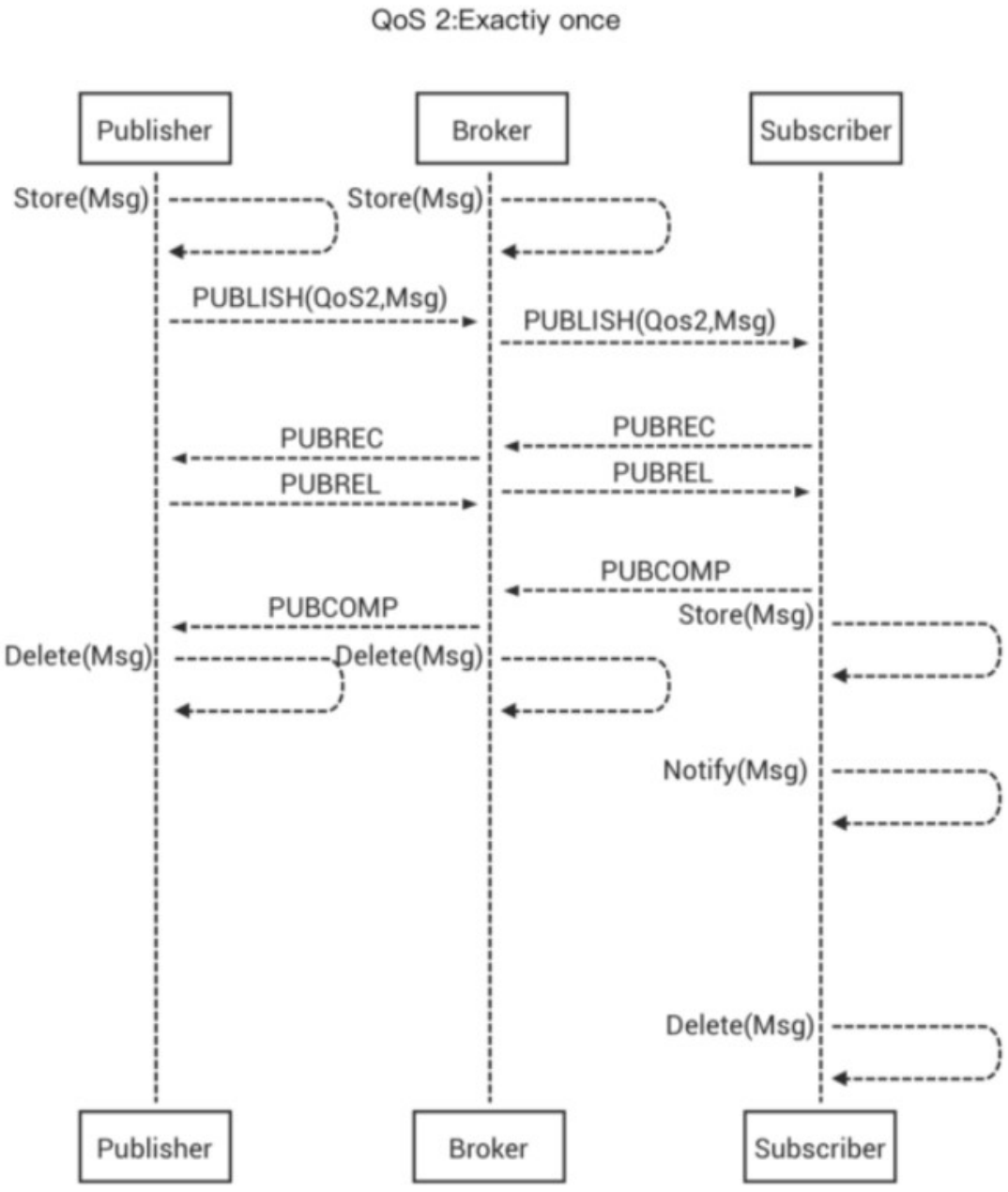
  With this method the broker sends back an acknowledgement (PUBACK) to the sender of the message. If the PUBACK gets lost in the transmission the sender won't realize the initial message was received by the broker, so It sends the same data again and again until an PUBACK is received. This means that sending is guaranteed, although the message may reach the broker more than once. The application using a QoS 1 must be capable of handling duplicate data.



3.4 – QoS 1, at least once

- **2 (exactly once):**

  This is the safest and slowest QoS level that MQTT can offer. It ensures that the data is received only once by the proper recipients by providing a four-part handshake between the devices.



3.5 – QoS 2, exactly once

3.2 - Publish/Subscribe model:

The publish/subscribe model is an architectural pattern (an architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context) which is intended to separate and decouple the client that sends the data (the publisher) from the client that receives that data (the subscriber).

A very important aspect of the pub/sub model is that only a part (subset) of the total messages is received by the subscribers to a certain topic. This filtering process is called Message Filtering and is performed by the broker node.

There are two common forms of message filtering:

- **Topic-based filtering:** In a topic-based system, all the subscribers will get all the messages published to the topic (or topics) on which they subscribe.

- **Content-based filtering:** In a content-based system, the subscribers to a topic will receive the message only if it matches the constraints defined by subscriber

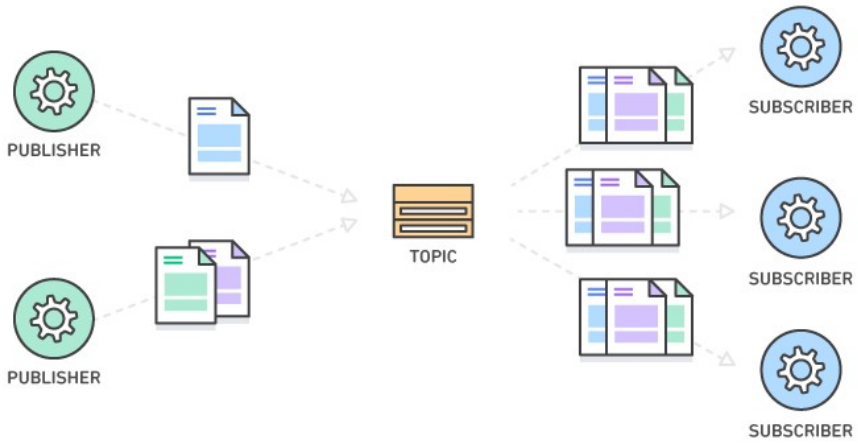
In a topic-based pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic.

But the messages are never directly sent from client A to client B (this provides an alternative to the classic client-server architecture) but instead gets categorized into classes, called topics.

The subscribers who want to retrieve data should subscribe to the interested topic/s without having any knowledge of the publisher (if any) who published the data in that class.

It’s clear that the communication between the sender and the receiver nodes that make up the network must be handled by another component, which is the broker.

The main advantage of this model is that it enables event-driven architectures and asynchronous parallel processing while improving performance, reliability and scalability to the system being developed.



3.6 – publish/subscribe model example

3.3 – Message queue:

A message queue is a form of service-to-service communication used in serverless and microservices architectures. It implements an asynchronous communication pattern between two or more processes/threads whereas the sending and receiving party do not need to interact with the message queue at the same time.

- A queue is a line of things waiting to be handled, starting at the beginning of the line and processing it in sequential order.

- A message is the data or the commands transported between the sender and the receiver clients; it's essentially a byte array with some headers at the top (the size of data that may be transmitted in a single message have implicit or explicit limits).

- A message queue is a queue of messages sent between applications. It includes a sequence of work objects that are waiting to be processed (The number of messages that may remain outstanding on the queue have implicit or explicit limits too).

Usually, messages are stored on the queue until they are processed by the consumer and finally deleted. Each message is processed only once, by a single consumer, for this reason, this messaging pattern is often called one-to-one, or point-to-point, communications;
so when a message needs to be processed by more than one consumer, message queues can be combined with a Pub/Sub messaging design pattern.

3.7 – message queue example

Message queues can be used to decouple heavyweight processing, to buffer work, and to smooth spiky workloads and therefore to provide communication and coordination for distributed modern-day applications without losing messages or requiring other services to be always available.

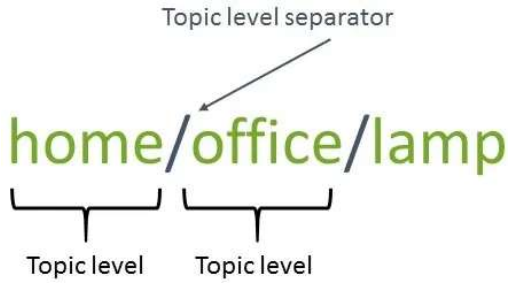
The basic architecture of a message queue is simple; there are client applications called producers that create messages and deliver them to the message queue. Another application, called a consumer, connects to the queue and gets the messages to be processed. Messages placed onto the queue are stored until the consumer retrieves them.

More specifically, in a typical message-queing implementation, an appropriate software (a queue manager or a broekr) is installed on a machine and a named message queue is defined (or a message queueing service is registered). Then an application registers a software routine that "listens" for messages placed onto the queue (where other software programs may already had transfered messages) and finally the receiving application processes the message in an appropriate manner.

3.4 - MQTT example:

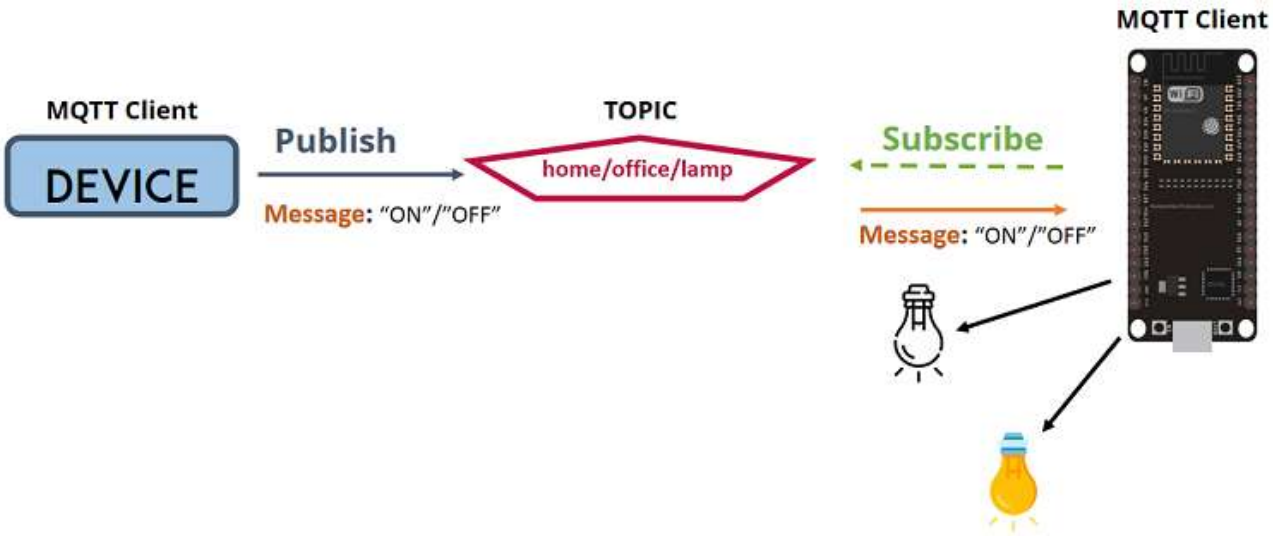
This example is going to show how to use the MQTT protocol to turn on or off a lamp with some board (Arduino, Esp etc...) or electronics capable of running an MQTT library that receives messages pushed by a publisher client to a user defined topic.

The first thing to do is to create a topic and a wrapping logic in the publisher device to push “on” or “off” messages to that topic.



3.8 – MQTT topic example

Once the topic is created, it is necessary to have another device capable of turning on or off the lamp (e.g., an Esp32 device connected to some relays) which is subscribed to the previously created topic.



3.9 – full MQTT example

Finally, the subscriber should implement some logic to perform actions (e.g., interact with relays) when the messages are received so that when the publisher publishes an “on” message to the “home/office/lamp” topic, the lamp is going to be turned on (if it was turned off) or off if it receives the “off” message.

4 – PICO-MP:

PICO-MP is the first publish/subscribe based, reliable and **fully decentralized** event-detection macro-programming middleware (a framework is a "software glue", which provides services to software applications beyond those available from the operating system) for wireless sensor and actuator networks (WSANs) applications.

A WSAN network is a group of sensors nodes that gather information about their environment and actuators, such as servos or motors, that interact with them. All elements communicate wirelessly and interaction can be autonomous or human-controlled.

Pico differs from other frameworks and middlewares for WSAN building because the events processed by the framework are expressed as a variation of a first-order logic formula and can be checked by nodes that composes the network (not only the root node) thus performing a distributed checking of the events, allowing edge computing to optimize and save a lot of resources (cpu/memory/bandwidth/battery capacity).

Global formulae in this language are checked in a distributed way by a network of brokers. Each broker uses the global formula to calculate a local sub-formula (called **projection**) which can be locally verified.

PICO-MP ensures that the distributed checking of projections produces the same results as the corresponding global formula returning the proper truth value only if the constraints defined by the developer while defining the predicate and the formulaes are met.

The high decoupling between information producers and consumers provided by the pub/sub paradigm implemented by other classic frameworks is a key factor in WSAN applications, where nodes can be added, removed, or can simply switch off their radio. Pub/sub systems can be categorized as topic-based and content/filter-based.

4.1 – PICO-MP formulae overview:

In PICOMP, a formula is often identified with two names: **subscription** and **projection**.

- **Subscriptions** are formulae coming from the subscriber node up to the root node. The packet that brings the subscription up is often called a subscription packet. The subscription formula is stored by the intermediate and the root node due to provide back the formula changes or also called notification.

- On the other hand, a **projection** represents the formula flooded by the root node down to the tree's leaf nodes. Projections are stored by brokers due to perform event checking. These brokers store all projections coming from the root, whereas the pubsmart node stores projection related to their channel.

4.2 – PICO-MP subscription overview:

A subscriber can subscribe to a formula to the upper node by sending a subscription. This subscription is a formula that can be used for event-detection or data filtering.

- **Event detection formula:**
    - $\forall x{:}T, \exists y{:}U \mid P(x) \wedge F(y)$

The formula above is an example of event detection formula and is divided into two parts:

- The former represents the definition of the variables (the part before the "|"). The set declared in the first part is user-defined and are related to the publication channel name. The channel name can be only an uppercase alphabetic character. Each declaration must be divided by a comma.

- A '|' character separates the two parts.

- The latter, instead, represents a set of predicates in a conjunction form. Each predicate cannot have more than one variable. Each variable used in the predicate must be declared in the first part

By subscribing an event detection formula, the framework returns only the global truth if the conditions specified in the formula subscribed are met.

- **Data filtering formula:**
    - $\exists x{:}T, \exists y{:}U | vP(x) \wedge F(y)$

The above stated formula is an example of data filtering formula.

This kind of formula allows the developer to retrieve actual data readings from the network and not only the truth value generated by the events.

In the example a character “v” is present in front of the P predicate, this means that the subscriber will receive data, which makes the formula truth, true locally. So the upper nodes will receive the truth value and the values that generated that truth value whether its a PICO_PREDICATE_TRUE or a PICO_PREDICATE_FALSE value; this depends of the character put in front of the subscribed formulae (needed to decide how data will pass through the father nodes) and how the developer decides to handle the proper **truth change** event.

Is it possible to use PICO in a data filtering way with other character as well:

- “**f”:** this character in front of a predicate state that the subscriber will receive data, which makes the formula truth, false locally

- **“c”:** the subscriber will receive data, which makes the formula truth locally changes

- **“a”:** the subscriber will receive data independently by the formula truth

It is possible for the developer to modify the pico/config/prodConfig.h and use the characters he wants.

It is important to notice that the pubsmart nodes MUST share the predicate implementation between the brokers, pubsmart and root nodes to ensure the correct behavior of the framework.

4.3 – PICO-MP nodes:

The Pico framework gives us the possibility to develop code for different types of nodes which mostly share the same events, making it easy for the developer to decide the overall behaviour of the WSAN system based on the “problem” he wants to face. These nodes are briefly described below:

4.3.1 - Publisher node:

the simplest and basic one. The node class is picoPubNetworkNode.The main functionality of the publisher is to send sensor data over the network. This node, similar to a MQTT publisher, is tailored for low capability sensor devices which can only collect and analyze data.

It has the following firm:

```
picoPubNetworkNode pubNode(5000,
                           nodeDisconnected,
                           childrenToRoot,
                           getTime,
                           PICO_PROTOCOL_QOS_FULL);
```

4.1 – PICO-MP publisher node declaration

The parameters required by the constructor are:

- **Keep alive timer:** Each node in PICOMP, has a different keep alive. This parameter represents the keep alive interval before the node is signaled to be out.

- **nodeDisconnected event:** The node disconnected event is triggered when the node didn’t contact the father node for a long time or the father node is disconnected form the network.

- **childrenToRoot event:** The event triggered when the current node wants to communicate with his father node, whether it's a root or a broker.

- **getTime event:** This method just returns the current time in milliseconds; necessary for Pico to work correctly.

- **Quality of Service (QoS):** The quality of the service lvel that pico provides. Developer should set this parameter based on how he is implementing pico. For example, he can avoid a QoS if he is using a reliable transport protocol like TCP to avoid unnecessary communication overhead, or he can choose PICO_PROTOCOL_QOS_FULL if he is implementing pico with UDP.

4.3.2 - Pubsmart node:

The pubsmart node is an advanced publisher. Publications can be filtered by projections sent by the father node. The formulae projected are related to publication channels sent before.

It has the following firm:

```
picoPubSmartNetworkNode pubSmartNode(5000,
                                     10000,
                                     nodeDisconnected,
                                     childrenToRoot,
                                     getTime,
                                     PICO_PROTOCOL_QOS_FULL);
```

4.2 – PICO-MP pubSmart node declaration

The parameters required by the constructor are the same as the pub node except for the second one which is:

- **Pubsmart Sync timer:** The pubsmart has a set of projections related to the channels published. These are provided by the upper node when the node executes the update or the sync method. The sync method is used to retrieve or delete formulae. This sync interval determines when the pubsmart triggers this method.

4.3.3 - Subscriber node:

The subscriber subscribes and unsubscribes formulae in the network. The developer does not specify predicates in the subscriber node. Formula predicates are defined in pubsmart/root and broker nodes.

For each subscribed formula, the node receives a truth notification when the formula truth changes. Moreover, subscribers can receive publications for data-filtering formulae.

It has the following firm:

```
picoSubNetworkNode(5000, 5000,
                   onNodeDisconnectEvent,
                   onGetTime,
                   onChildrenToRoot,
                   onDeletedSubscription,
                   PICO_PROTOCOL_QOS_FULL);
```

4.3 – PICO-MP subscriber node declaration

4.3.4 - Broker node:

The broker node communicates with children and the father nodes. Children's communications follow request-response or broadcast methods. The broker receives a request packet and forwards back the response to the same node. The next sent packet is sent to the last sender node. One request packet corresponds to one response packet; broadcast communication to children can also be sent.

PICOMP is not thread-safe. For this reason, packets are elaborated synchronously and only one packet can be processed at a time.

The developer may choose to enable, filter, or disabled these packets, depending on the protocol below.

```
picoBrokerNetworkNode(5000, 5000,
                      onChildrenToBroker,
                      onBrokerToChildrenPacket,
                      onTimeEvent,
                      PICO_PROTOCOL_QOS_FULL);
```

4.4 – PICO-MP broker node declaration

The different events that are required by the node are:

- **onChildrenToBroker event:** this is an equivalent method as the children to root event.
- **onBrokerToChildrenPacket event:** This event sends unreliable packets to children. This communication can be **broadcast** or a response to the children's packet.

4.3.5 - Root node:

Likewise the broker node, the root node, manages the underlying children and the other data structure. However, the root node is more straightforward than the broker one. The root is the upper node. The root node does not perform connection or disconnection activity. Furthermore, packets are sent and received only to children.

The root node has the following firm:

```
picoRootNetworkNode(onRootToChildren, onGetTime);
```

4.5 – PICO-MP root node declaration

- **onRootToChildren:** This event sends unreliable packets to children. This communication can be **broadcast** or a response to the children's packet.

4.4 - Publishing data:

The publisher node should send publications in the network to the father node. In a WSAN network, each node may have multiple sensors: temperature, humidity, luminosity. In PICOMP, we call these sensors types: channels. These channels are encoded as one character and varies between A and Z.

A publication is organised as a set of sub publication. Each contains a single user-defined channel and the data that goes along with it which can be represented with an ascii character that varies between ‘j’ and ‘z’.

Summing up, to send a publication, the developer should:

- Define a channel:

```
picoChannel channelT{'T'};
```

- Define a data name:

```
picoDataName dataNameX{'x'};
```

-
- Define the publication object:

```
picoPublication pub;
```

- Create the publication and add the desired data:

```
node.createPublication(pub, channelT);
node.addHalfFloat(pub, dataNameX, 10.0);
```

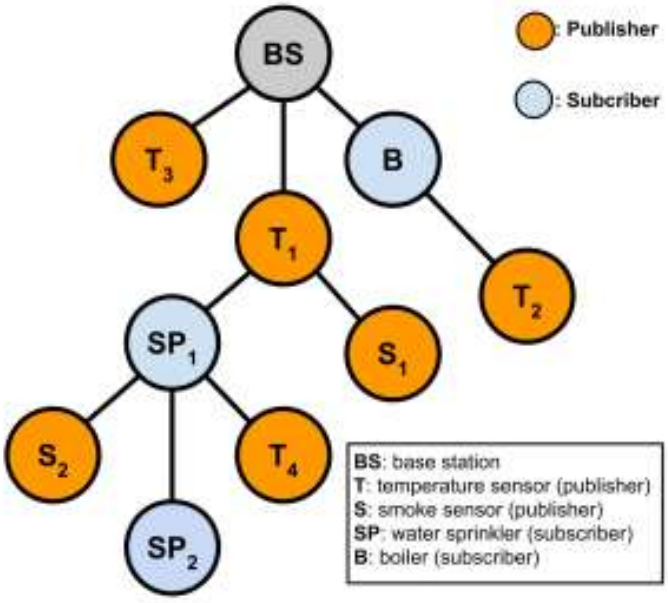
- And finally send the publication:

```
node.sendPub(pub);
```

4.5 – How PICO-MP works:

PICO-MP is a type-based pub/sub system. In a type-based subscription the declaration of a desired type is the main discriminating attribute by giving a coarse-grained structure on events (like in topic-based) on which fine-grained constraints can be expressed over attributes (like in content-based). Type-based pub/sub in this sense resembles the filtered topic model. Publishers periodically send information to the PICO-MP infrastructure in the form of typed events. Subscribers express their interests in the form of first order logic formulae which predicate over sets of events. Each set is composed of events of the same type. Notifications are sent to subscribers whenever a formula becomes true or false.

PICO-MP minimizes the size of the notifications since subscribers only get a true/false notification; subscribers with particular needs can request, together with the notification, the set of events that triggered the notification.



4.7 - example *of a PICO-MP deployment network*

PICO-MP brokers are organised in a tree overlay network.

Publishers and subscribers connect to exactly one broker. Brokers cooperate in order to manage subscriptions, events and distributed formula checking. Brokers ensure that all subscriptions flow up till the root node. The communications are reliable, and fault tolerance is guaranteed by using a timeout mechanism.

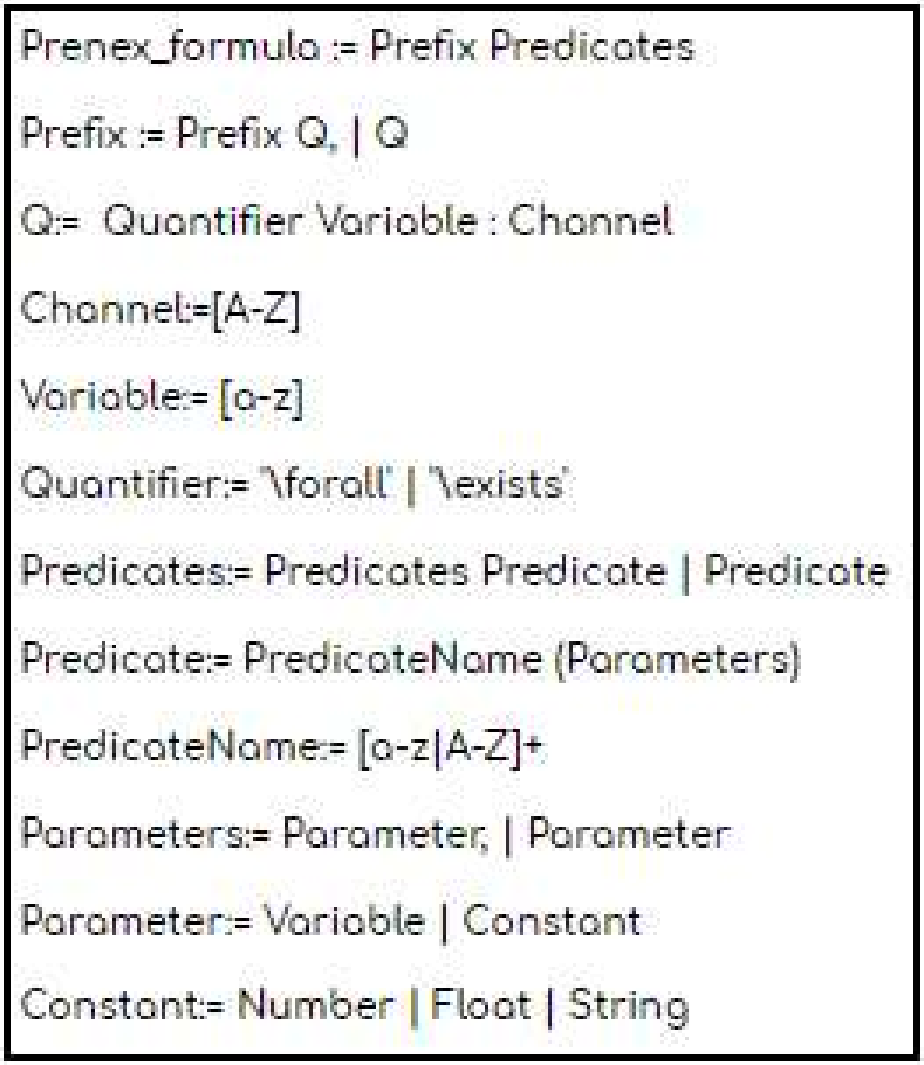
A broker can prevent a subscription to reach the root when an equivalent subscription was already sent. This optimisation process can reduce the number of subscriptions that flow into the notification service. The root forwards the global formulae down to the network of brokers in order to perform their truth check as close as possible to the event sources (i.e., the publishers).

More precisely, each broker receives the global formula and generates a local projection of it. This contains parts of the global formula whose truth can be locally verified at the broker. The projection generation process assures that if a projection truth does not change, then the truth of the originating formula does not change too.

When a projection switches its truth value the broker sends a message to its father, communicating which parts of the formula changed their truth and how.

Eventually, the root will receive enough information to state that a certain global formula changed its truth value, and will notify the subscribers.

4.6 – PICO-MP Subscription language:



4.8 – PICO-MP subscription language

The PICO-MP subscription language is defined by the grammar of Figure 4.8

A subscription is a first order logic formula in Prenex normal form (line 1 of the grammar of Figure 4.8) with no free variables.

This is composed of a prefix part that is followed by a matrix:

The prefix is a list of quantified variables (lines 3 – 4 of the grammar of Figure 4.8), while the matrix is a conjunction of predicates (line 11 of the grammar of Figure 4.8).

Constants and variable fields can be of the PICO-MP primitive types that are numeric (byte, short, int, long, double), boolean and string, while a quantified variable x ranges over the set St (lines 6 – 7 of the grammar of Figure 4.8).

4.7 – PICO-MP Distributed notification service:

```
1  publisherTable = {(t1, pub1), ..., (tn, pubn)}
2
3  brokerTable = {(t1, b1), ..., (tn, bn)}
4
5  subTable = {(f1, sub1), ..., (fn, subn)}
6
7  projTable = {(f1, f1∀, f1∃[]), ..., (fn, fn∀, fn∃[])}
```

4.9 – PICO-MP tables

4.7.1 - Event publication management:

Each broker b (note that the root is only considered a particular case of broker) implements the registration type set Tb and the subtree type set Ttree(b) by using a publisherTable and a brokerTable.

A broker uses these tables in order to obtain a projection from a global formula.

The publisherTable and brokerTable tables are kept updated by using the registration, unregistration, advertisement and unadvertisement PICO-MP messages:

- A publisher pub of the type t, registers to a broker b by using a registration(t,pub) message.

- When the publisher pub exits the system, it must send an unregistration(pub) message to its broker b.

- A broker b can send messages of the type advertisement(t,b) to its parent broker par(b). This uses the advertisements to keep its brokerTable updated.

- A broker b can send an unadvertisement(t,b) message to its parent par(b) when it cannot receive anymore events of the type t.

4.7.2 - Subscription management:

A broker b and a root r use the subTable (see the previous figure) in order to keep information about subscriptions that originated in their sub-tree. The subTable is kept updated by using the subscription and unsubscription PICO-MP messages:

- A subscriber sub can send a subscription(f,sub) message to a broker b. When b receives the message subscription(f,sub) its subTable is updated with the entry (f, sub). subscriptions are always propagated till the root.

- A subscriber sub which is no longer interested in a certain formula f can send an unsubscription(f,sub) message to its parent par(sub). When b receives this message it removes the entry (f, sub) from its subTable. Unsubscriptions are always propagated till the root

Notice: a broker does not forward a subscription formula f1 that is equivalent to a subscription f that was previously sent.

4.7.3 - Projection generation process:

A root always forwards down to the broker tree each new subscription f. When a broker b receives the formula f, it can perform the projection procedure The projection procedure forwards f to all children brokers that can generate at least a projection from f.

4.7.4 - Distributed event matching:

A broker b performs the distributed event matching to check the truth of projections. This check reduces the number of pubs that are forwarded towards the root and allows a distributed computation of each global formula f. This evaluation is performed on all publication events that can be locally observed at b. This forwards synchronisation predicates are true.

4.7.5 - Notification Delivery:

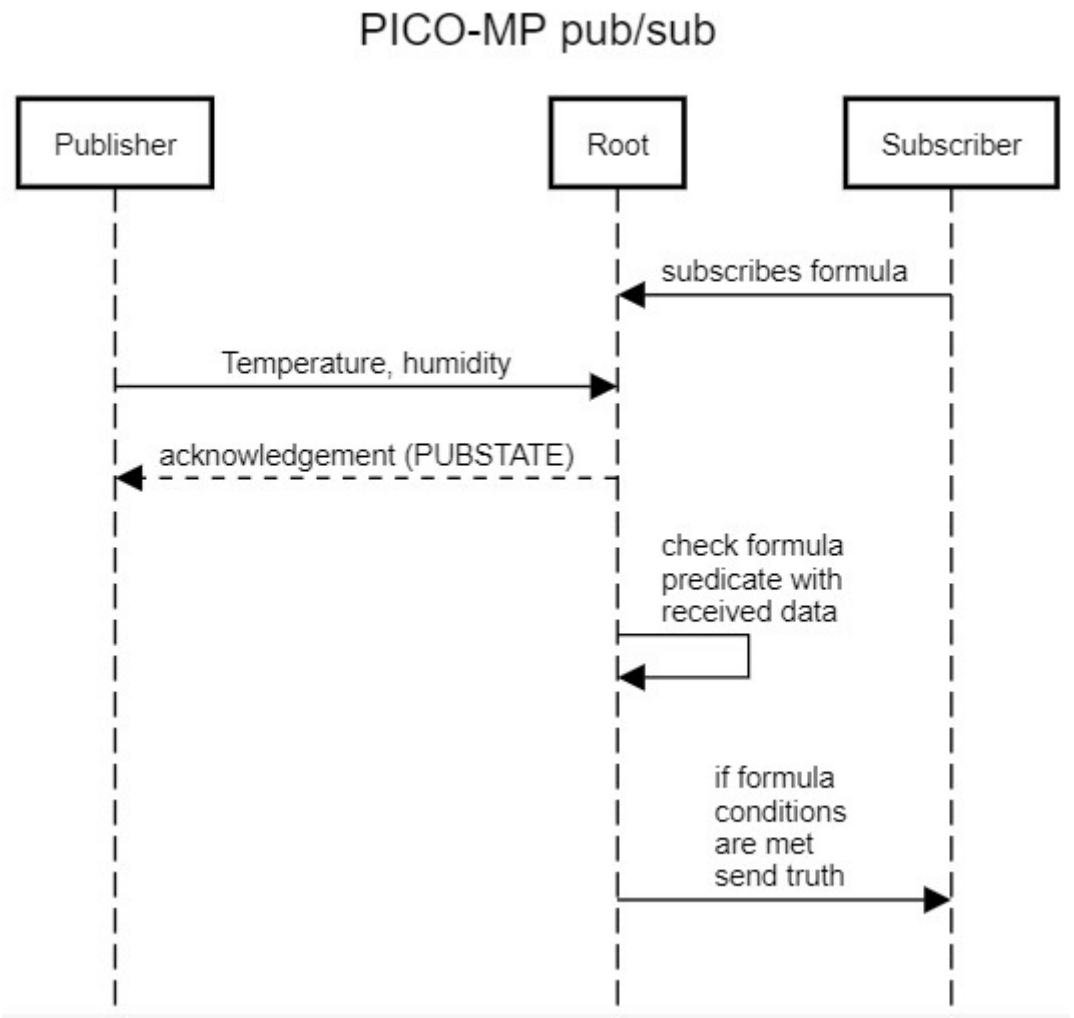
A root is a particular type of broker that has access to the entire state of the system. The root receives synchronisation messages from its children brokers and can calculate the truth of every global formula f. This is done by procedures that are similar to the broker ones. Whenever the root detects a change in the truth of a subscription formula f, it notifies its subscriber of it. To do this, each broker (starting with the root itself) uses its subTable to trace back the subscriber which submitted

the subscription. In this way, a notification(f) message is sent to the subscriber of f and all equivalent formulae. At this point, subscribers can react to notifications with a customised callback procedure, which gives the programmer the power to define any possible behaviour.

4.8 - How the nodes communicate:

- **PICO Pub/sub sequence diagram:**

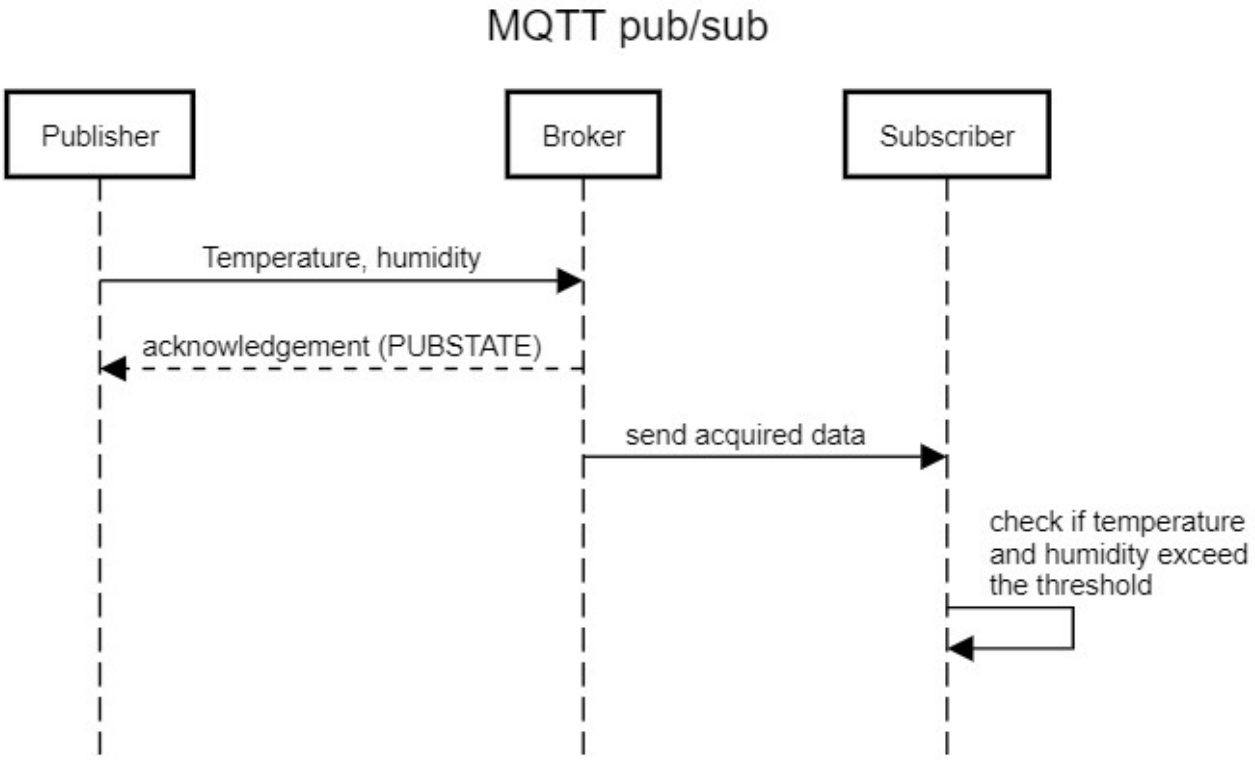
The below represented diagram briefly show how a PICO simple pub/sub model with one subscriber works:



4.10 – pico pub/sub sequence diagram

- **MQTT Pub/sub sequence diagram:**

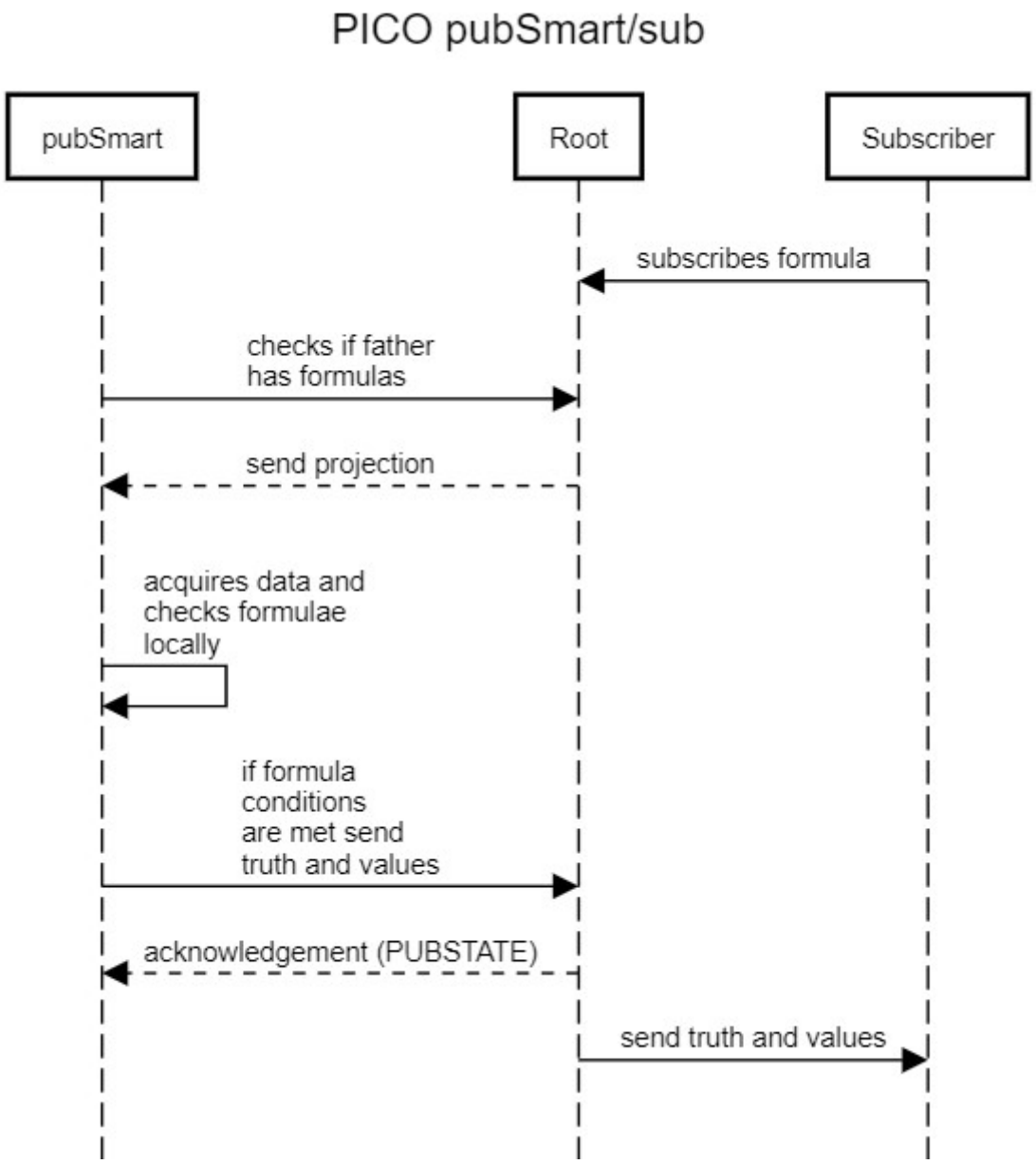
  The below represented diagram briefly show how a PICO simple pub/sub model with one subscriber works:



4.11 – mqtt pub/sub sequence diagram

- **PICO pubSmart/sub sequence diagram:**

  The below represented diagram briefly show how a PICO pubSmart/sub model with one subscriber works:



4.12 – pico pubSmart/sub sequence diagram

5 – Benchmarks between protocols

The purpose of this section is to show what has been actually done to achieve the results described by the thesis, explaining briefly how the PICO-MP and MQTT WSANs were set up (with photos), how the codes works and how to possibly expand the Pico project part in the future this project if anybody wants to and finally showing the collected data after the benchmarks and testings were done.

5.1 - Project notes:

The provided implementation of PICO-MP is based on UDP because the project was developed in a LAN network, so all the PICO-MP nodes were implemented using the PICO_PROTOCOL_QOS_FULL parameter in the constructor to let PICO provide reliability over the exchanged messages.

5.1.1 - Ideas for future expanding the project:

- Make the PICO and MQTT implementations independent of the connection used, whether it's wifi, bt or esp-now
- Crypt data transfer between the father nodes, the children node and viceversa
- Use a proper DBMS for the static storage of collected data
- Use a simple web server for data and statistics visualization
-

5.1.2 - Utilized formulae explanation:

The subscriber looked for temperature and humidity data which were respectively published in the picoPublications by the following channels:

- **"T" picoChannel:**
  contains the temperature publications encapsulated in the "w" picoDataName.
  - $\exists w:T$

- **"H" picoChannel:**
  contains the humidity publications encapsulated in the "x" picoDataName.
  - $\exists x:H$

- **"C" picoChannel:**
  the data provided by this channel is not utilized because no algorithm for the data was provided, but it correctly receives the data as the other channels, it contains the VoC (volatile organic compound) data in the "y" picoDataName.
  - $\exists y:C$

5.2 - Used materials:

To achieve this goal the following materials were used:

- 1 x Raspberry Pi 3 Model B

- 2 x ESP32 TTGO T-Beam modules
    - Note: there should have been 3 ESP32 modules, but one broke

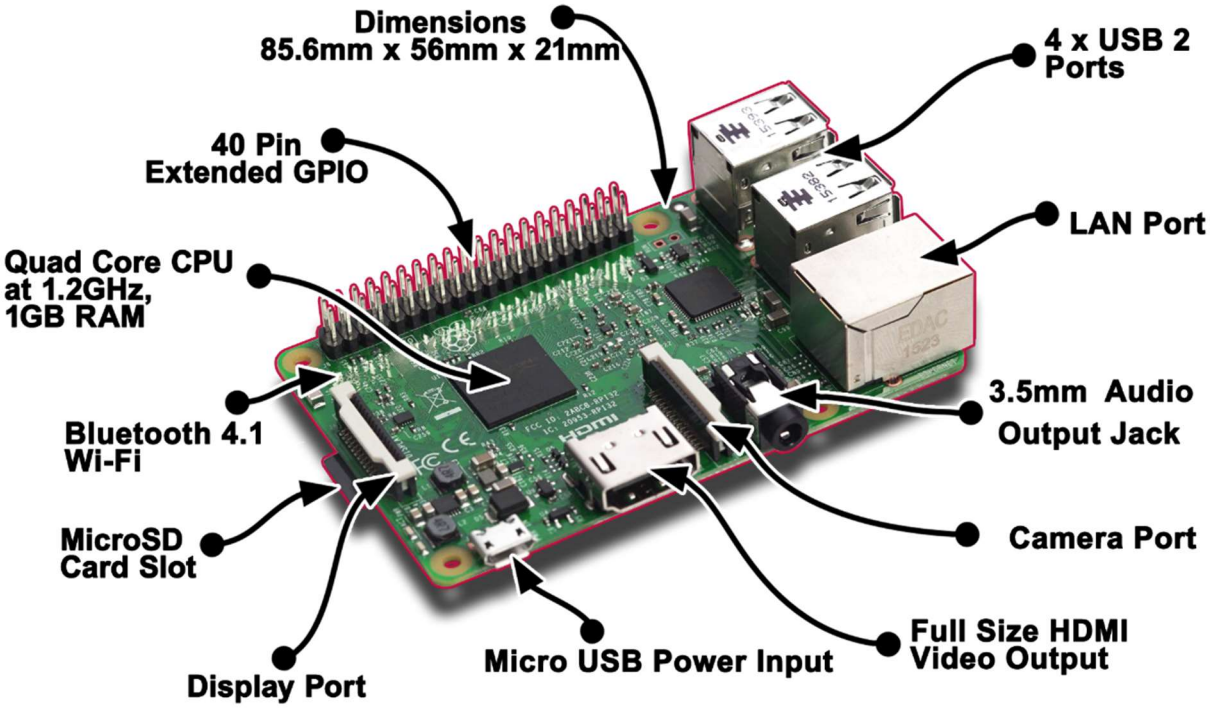
- 2 x BlueDot BME680 Bosch environmental sensors
    - Connected with the i2c bus

- 2 x Samsung 18650 protected batteries

- 1 x 18650 charger with display to read battery level measurements

5.2.1 - Raspberry Pi 3 Model B overview:

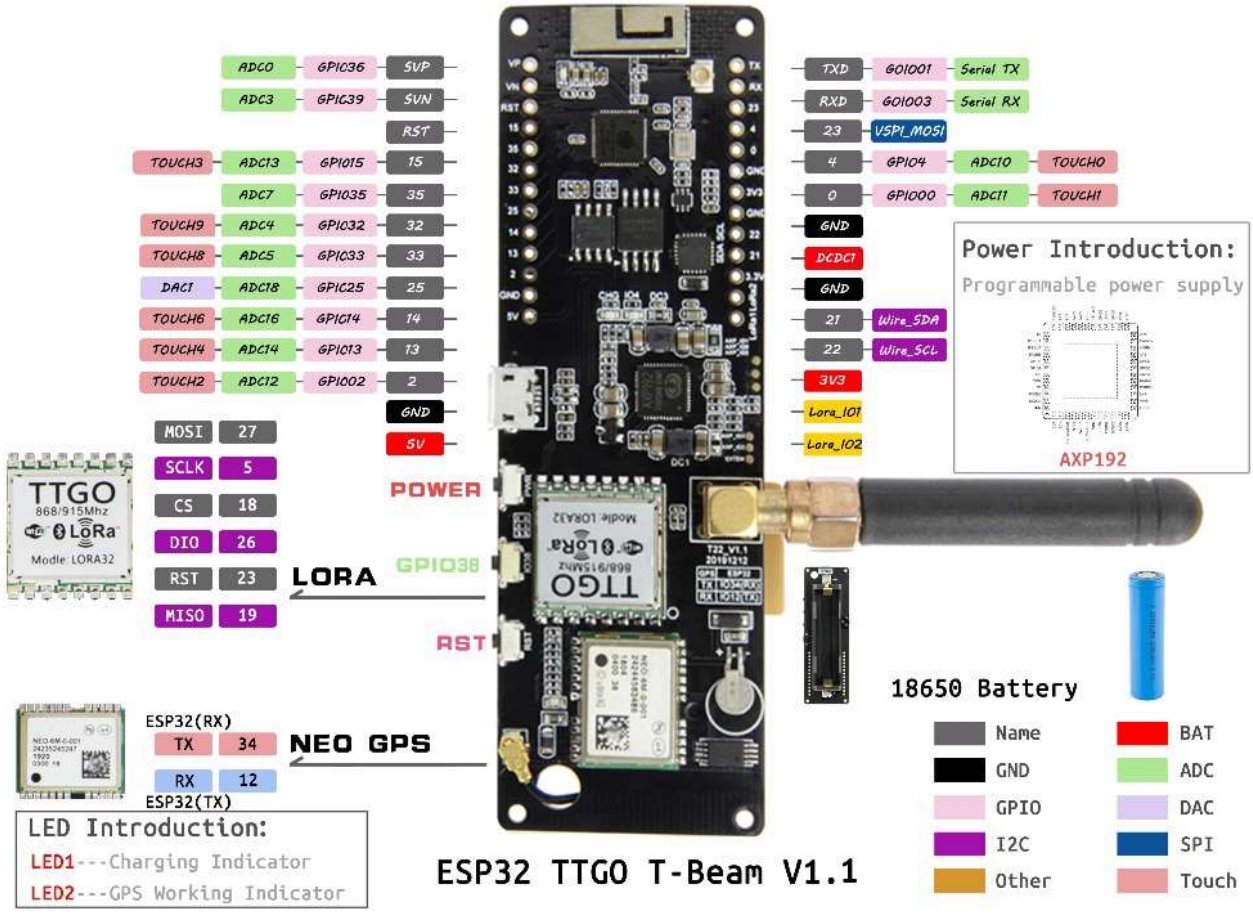
- 32Gb sd
- 4 arm cores
- 1Gb ram
- Raspbian Os



5.1 – Raspberry Pi 3 model B used in the project

5.2.2 - TTGO T-Beam overview:

- ESP32
- 8MB PSRAM
- 4MB Flash
- wifi, gps Bluetooth
- 18650 battery



5.2 – TTGO T-Beam used in the project

5.2.3 - Bluedot BME680 overview:

The BlueDot BME680 Environmental and Gas Sensor allows temperature, humidity, pressure, altitude and volatile organic compounds (VOCs) in the air measurements.

Down below is the sensor technical sheet.

Parameter	Value
Temperature Range	-40°C to 85°C
Absolute Accuracy (Temperature)	± 1.0°C (0...65°C) and ± 0.5°C (at 25°C)
Humidity Range	0% to 100%
Absolute Accuracy (Humidity)	± 3% (20...80%)
Pressure Range	300 hPa to 1100 hPa
Absolute Accuracy (Pressure)	± 0.6 hPa
Board Power Supply	2.6 V to 5.5 V
Sensor Standby Current	0.2 µA
Sensor Current @ 1Hz Humidity and Temperature	2.1 µA
Sensor Current @ 1Hz Pressure and Temperature	3.1 µA
Sensor Current @ 1Hz Humidity, Pressure and Temperature	3.7 µA
Sensor Current @ p/h/T/gas depending on operation mode	0.09 – 1.2 mA
Sensor Current on Sleep Mode	0.15 µA
Communication Modes	I2C, SPI
I2C Addresses	0x77 (default) and 0x76
Board Size	26 x 19 x 2 mm
Mounting Hole Size	2.5 mm

5.3 – BME680 specifications

BME680 via I2C connections:

The first step is to connect the board to the power supply.

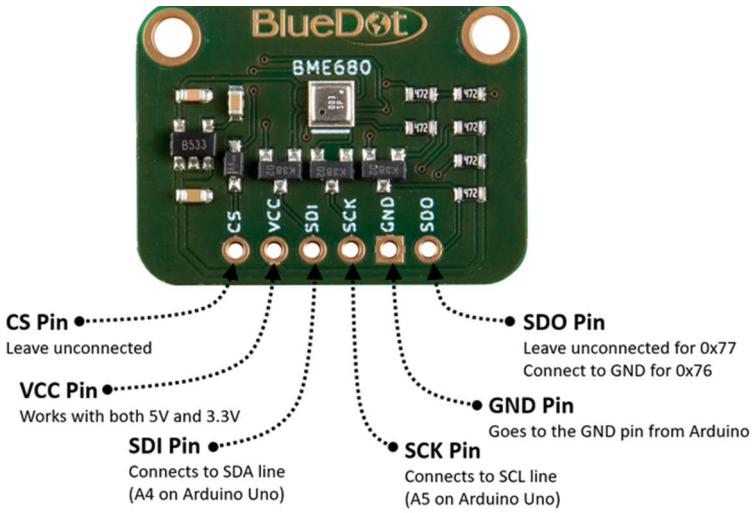
- **VCC Pin**. Connect the VCC pin from the board to either 5V or 3.3V output from ESP32 module
- **GND Pin**. Connect the GND pin from the board to the GND from the ESP32.

Once the power supply is connected, we need to allow it to use The I2C communication bus, which uses basically two wires.
The clock signal is generated by the esp and transferred to the sensor through the SCL line. The esp can send commands to the sensor using the SDA line. Just as well, all data from the sensor goes back to the esp through the SDA line. Because of that, the SDA line is bidirectional.

- **SDI Pin**. Connect the SDI pin from the board to the SDA line on the esp. This corresponds to the gpio 21 pin on the esp.
- **SCK Pin**. Connect the SCK pin from the board to the SCL line on your esp. This corresponds to the gpio 22 pin on the esp.
- **CS Pin**. Leave it unconnected.

Optional:

- **SDO Pin**. Here we have two options. Leave the SDO pin unconnected to use the default I2C address (0x77). Instead we can connect the SDO pin to GND in order to use the alternative I2C address (0x76).

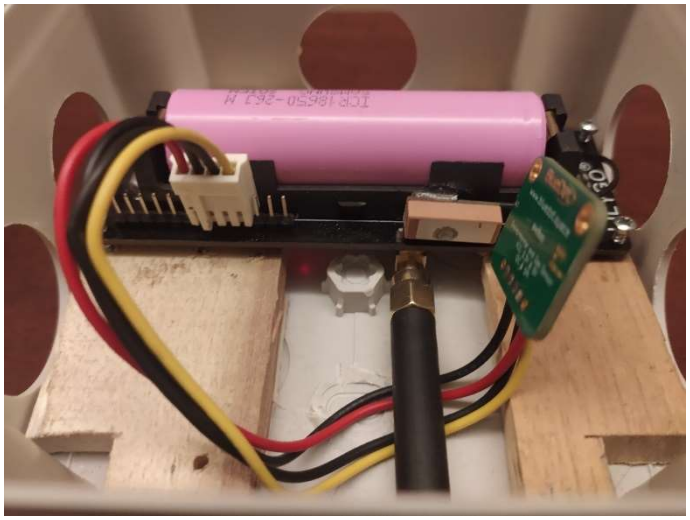


5.4 – Bluedot's BME680

5.3 – The project:

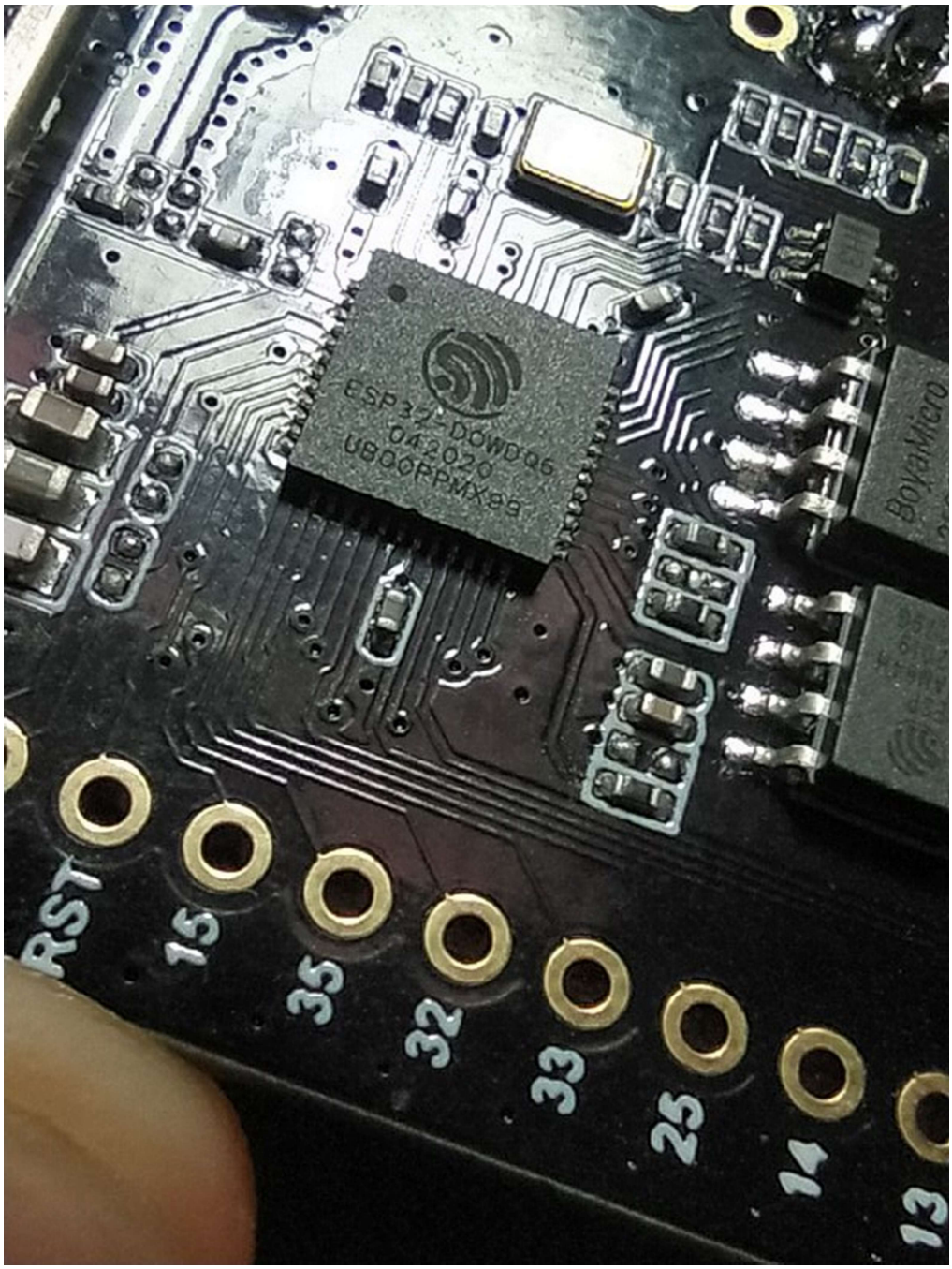
These are the two boxes that contains the esp32 used to run the PICO-MP pub, the PICO-MP pubsmart and MQTT publisher client.

They both contains 2 TTGO T-Beam boards powered on by a Samsung 18650 battery, fixed to the boxes with some wood and screws.



The BME sensor is plugged in with a proper connector to the esp, so that we can unplug it and replace it with anything we want by freeing the GPIO pins.

Macro photo of the involved ESP32-DoWDQ6 chip:



This is the Raspberry Pi 3 model B used for holding the both PICO-MP root node, the PICO-MP subscriber node, the MQTT broker node and the MQTT subscriber node:

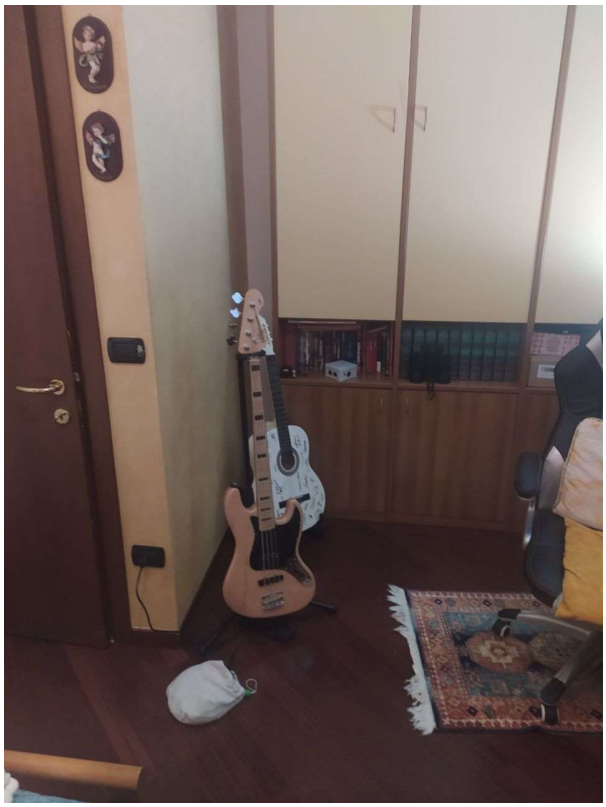
Charger used to measure battery levels and to charge the 18650 batteries:

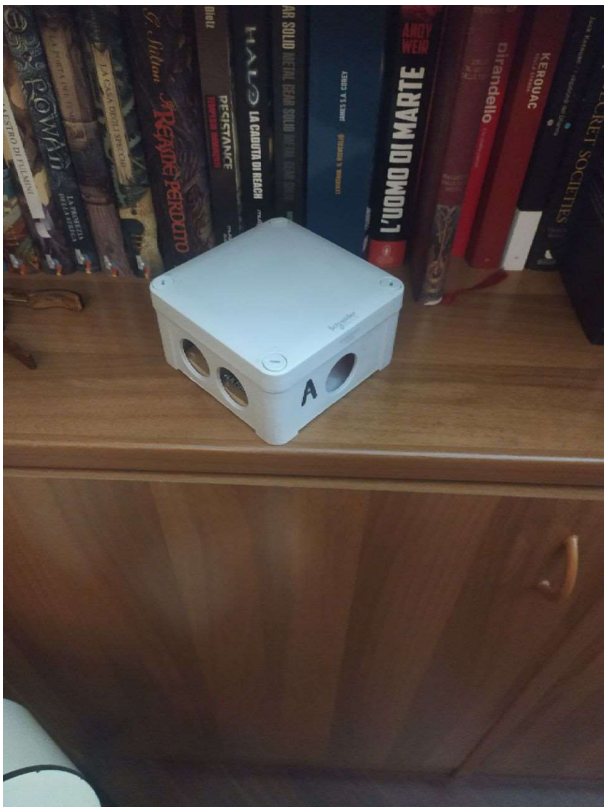
5.4 Important test notes:

These conditions are valid for all the executed tests:

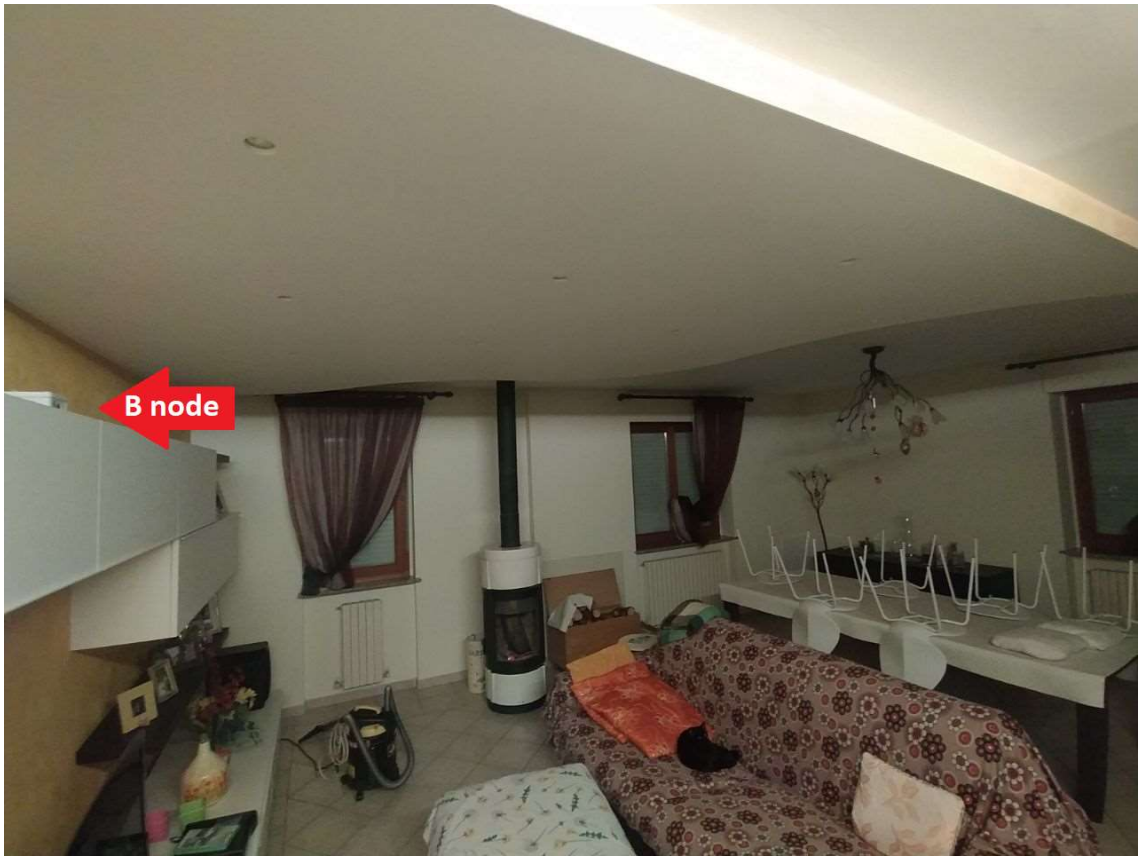
- The two publisher nodes are called A and B and their static addresses in my router are respectively 192.168.178.84 and 192.168.178.88
- The Broker and subscriber (for MQTT) are 2 different process in the same device
- The Root and the subscriber (for PICO) are 2 different process in the same device
- Position of node A (both Pico and MQTT): my bedroom (cold, humid)
- Position of node B (both Pico and MQTT): living room (hot, dry)

Node A testing in my bedroom, notice the position of the door to the left of the node, which was left mostly closed to favour a cold and humid ambient.





Node B testing in the living room, notice the stove (turned on almost every day) which favoured a hot and dry place, as opposite to the one in my room.



6 - EXECUTED TESTS AND COLLECTED DATA:

PicoMP test #1:

Network composed of:

- 2 standard publisher nodes
- 1 root node
- 1 subscriber node

Preconditions of the test:

- Battery level for node A: 100%
- Battery level for node B: 100%
- Deep sleep time: 180 seconds (3 mins)

Subscribed formula (1 predicate with 4 parameters, **event detection**):

- Ew:T, Ex:H | d(w,17.0,x.30.0)
    - Pico will return PICO_PREDICATE_TRUE if the temperature data published in the T channel will be less than 17.0 °C and if the relative humidity data published in the H channel will be less than 30.0%

Start | End time:

- Start time: 18:00
- End time: 00:00

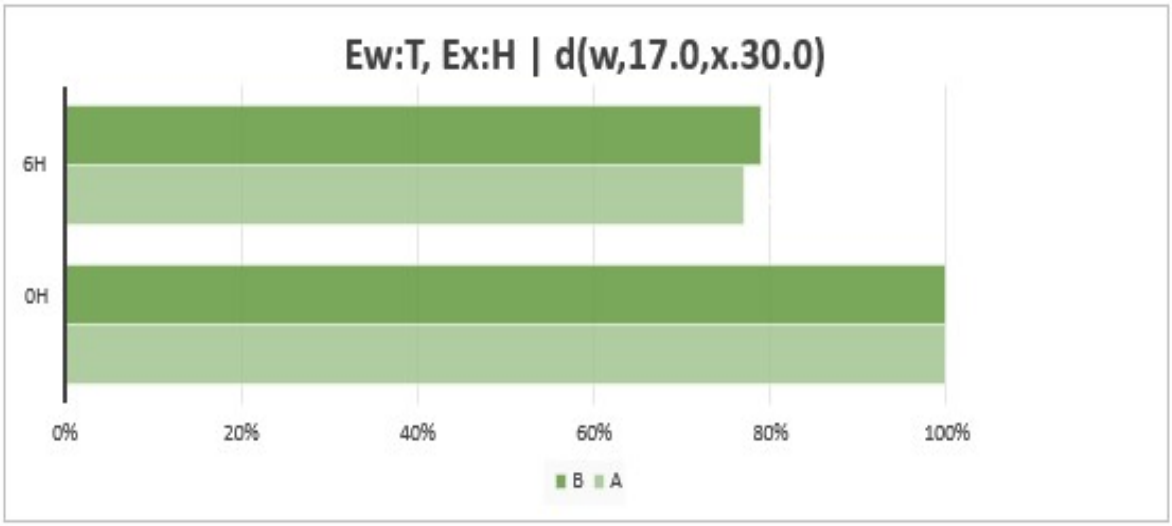
Remaining battery level after 6 hours of testing:

- Battery level for node A: 77%| used battery: 23%
- Battery level for node B: 79%| used battery: 21%

Remarks:

- The collected data is available in the “Logs” folder
    - Logs\logs_pico_sub\tests_sub\01-6_hrs_pub_1_predicate
    - Logs\logs_pico_root\tests_root\01-6_hrs_pub_1_predicate

Observed data:



	A	B
6H	77%	79%
0H	100%	100%

PicoMP test #2:

Network composed of:

- 2 standard publisher nodes
- 1 root node
- 1 subscriber node

Preconditions of the test:

- Battery level for node A: 100%
- Battery level for node B: 100%
- Deep sleep time: 180 seconds (3 mins)

Subscribed formula (2 predicates with 2 parameters, **event detection**):

- Ew:T, Ex:H | t(w,17.0) h(x,30.0)
    - Pico will return PICO_PREDICATE_TRUE the same way mentioned above but this time there are 2 separates predicates: one for each type of data.
    - To return PICO_PREDICATE_TRUE, both conditions needs to be true.

Start | End time:

- Start time: 17:23
- End time: 23:23

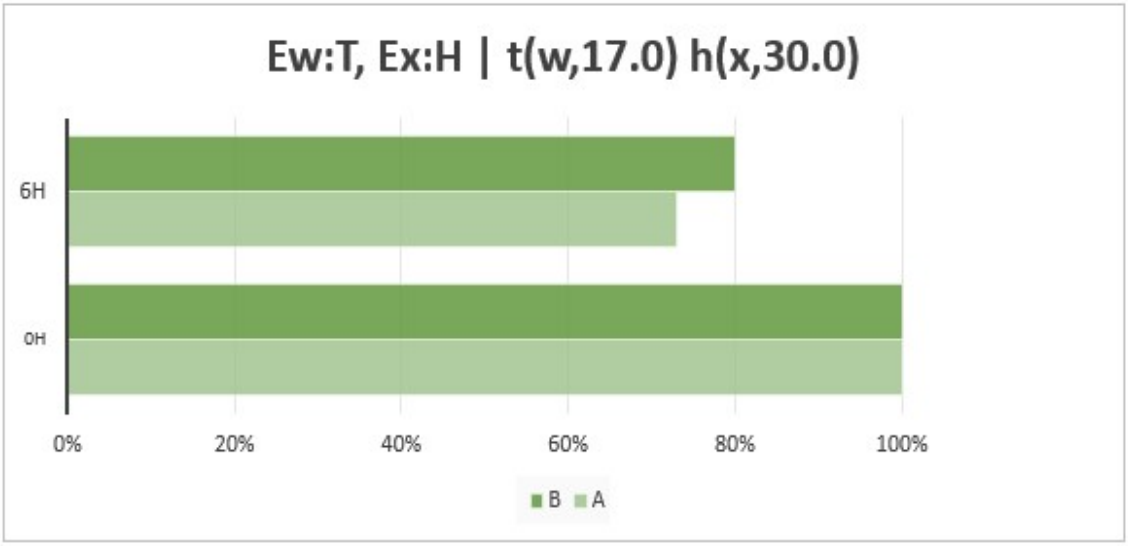
Remaining battery level after 6 hours of testing:

- Battery level for node A: 73%| used battery: 27%
- Battery level for node B: 80%| used battery: 20%

Remarks:

- The collected data is available in the “Logs” folder
    - Logs\logs_pico_sub\tests_sub\02-6_hrs_pub_2_predicates
    - Logs\logs_pico_root\tests_root\02-6_hrs_pub_2_predicates

Observed data:



	A	B
6H	73%	80%
0H	100%	100%

PicoMP test #3:

Network composed of:

- 2 pubSmart nodes
- 1 root node
- 1 subscriber node

Preconditions of the test:

- Battery level for node A: 100%
- Battery level for node B: 100%
- Deep sleep time: 180 seconds (3 mins)

Subscribed formula (1 predicate with 4 parameters, **data filtering**):

- Ew:T, Ex:H | **v**d(w,17.0,x.30.0)
    - Using the same formula as the #1 test but with a **"v" character in the front**, which means the subscriber will only get the measured data if the evaluated formula returns a PICO_PREDICATE_TRUE truth.

Start | End time:

- Start time: 03:37
- End time: 09:37

Remaining battery level after 6 hours of testing:

- Battery level for node A: 77% | used battery: 23%
- Battery level for node B: 80% | used battery: 20%

Remarks:

- The collected data is available in the “Logs” folder
    - Logs\logs_pico_sub\tests_sub\03-6_hrs_pubsmart_1_predicate_v
    - Logs\logs_pico_root\tests_root\03-6_hrs_pubSmart_1_predicate_v

- No actual data was received because the temperature and the relative humidity in my house **never passed the specified threshold**, but the reliability of the test was witnessed with adequate pre-tests before launching the official measurements

Observed data:

	A	B
6H	77%	80%
0H	100%	100%

PicoMP test #3.2 WiFi On/Off:

Network composed of:

- 2 pubSmart nodes
- 1 root node
- 1 subscriber node

Preconditions of the test:

- Battery level for node A: 100%
- Battery level for node B: 100%
- Deep sleep time: 180 seconds (3 mins)

Subscribed formula (1 predicate with 4 parameters, **data filtering**):

- Ew:T, Ex:H | **v**c(w,17.0,x.30.0)

Start | End time:

- Start time: 15:48
- End time: 21:48

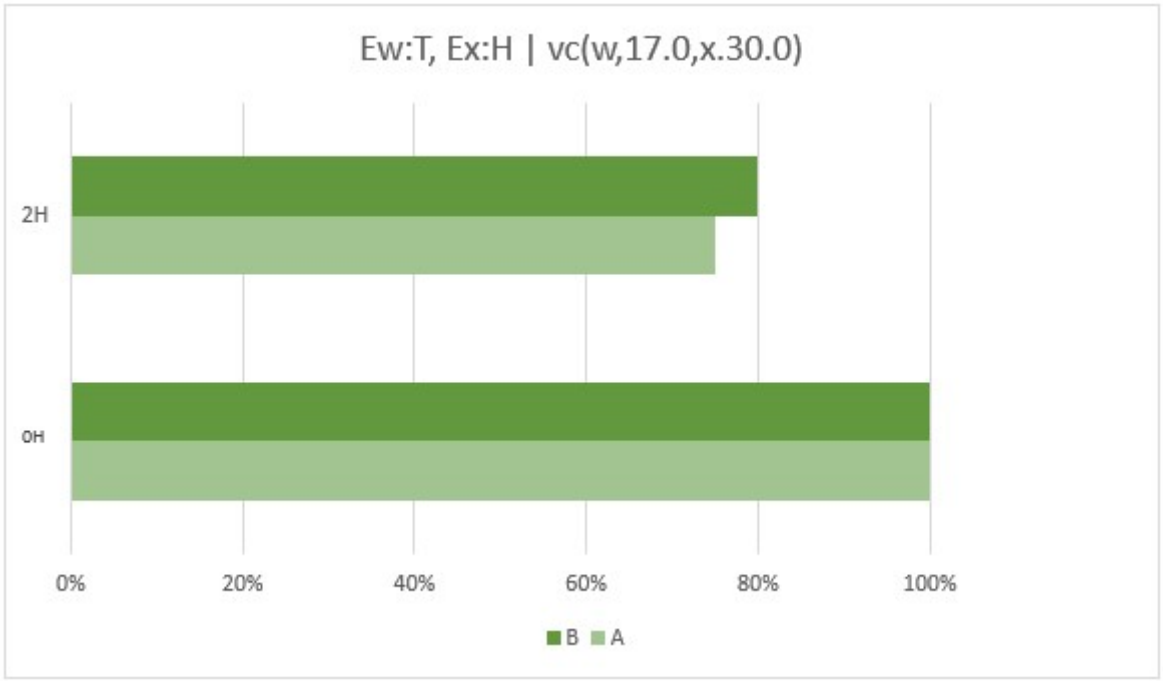
Remaining battery level after 6 hours of testing:

- Battery level for node A: 75% | used battery: 25%
- Battery level for node B: 80% | used battery: 20%

Remarks:

- The collected data is available in the “Logs” folder
    - Logs\logs_pico_sub\tests_root\06-6_hrs_pubSmart_1_predicate_v_noWiFi
    - Logs\logs_pico_root\tests_root\06-6_hrs_pubSmart_1_predicate_v_noWiFi
- The predicate character was switched to “c” but is actually the same as the previous one, no practical changes were made.
- **This test had the same preconditions as the test #3 but this time the ESP32 node was not connecting to the WiFi network if the evaluated formula returned a PICO_PREDICATE_FALSE truth.**

Observed data:



	A	B
6H	75%	80%
0H	100%	100%

PicoMP test #4

Network composed of:

- 2 pubSmart nodes
- 1 root node
- 1 subscriber node

Preconditions of the test:

- Battery level for node A: 95%
- Battery level for node B: 95%
- Deep sleep time: 180 seconds (3 mins)

Subscribed formula (1 predicate with 4 parameters, **data filtering**):

- Ew:T, Ex:H | **f**c(w,17.0,x.30.0)
    - Using the same formula as the #1 test but with a **"f" character** in the front, which means the subscriber will only get the measured data if the evaluated formula returns a PICO_PREDICATE_FALSE truth.
    - This means that every 3 minutes the pubSmart **will evaluate the formula false** (due to my in-house conditions) **and send the data, consuming more battery unlike the tests with a "v" character.**

Start | End time:

- Start time: 18:45
- End time: 00:45

Remaining battery level after 6 hours of testing:

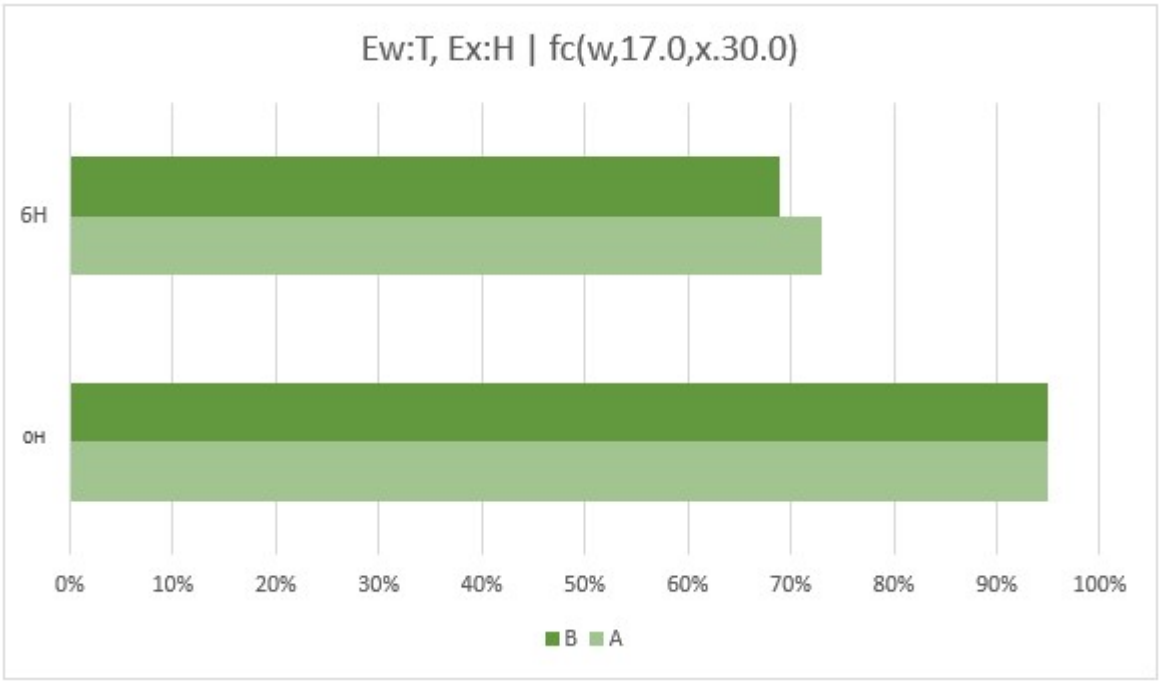
- Battery level for node A: 73% | used battery: 22%
- Battery level for node B: 69% | used battery: 26%

Remarks:

- The collected data is available in the "Logs" folder

- Logs\logs_pico_root\tests_root\05-6_hrs_pubSmart_1_predicate_f_OK
- Logs\logs_pico_sub\tests_root\05-6_hrs_pubSmart_1_predicate_f_OK

Observed data:



	A	B
6H	73%	69%
0H	95%	95%

MQTT test #1 (read remaks)

Network composed of:

- 2 publisher client nodes
- 1 broker node
- 1 subscriber client node

Preconditions of the test:

- Battery level for node A: 100%
- Battery level for node B: 100%
- Deep sleep time: 180 seconds (3 mins)
- **Used 2 ANDed boolean variable and a QoS 1 level to achieve the same behaviour as Pico**

Start | End time:

- Start time: 03:18
- End time: 10:18

Remaining battery level **after 7 hours** of testing:

- Battery level for node A:70% | used battery: 30%
- Battery level for node B:77% | used battery: 23%

Remarks:

- I consider this test unreliable because it ran 1 hour more than expected, check the next text for correct 6 hours measurements.
- The collected data is available in the “Logs” folder
    - Logs\logs_mqtt_sub\1

Observed data:

	A	B
7H	70%	77%
0H	100%	100%

MQTT test #2

Network composed of:

- 2 publisher client nodes
- 1 broker node
- 1 subscriber client node

Preconditions of the test:

- Battery level for node A: 70%
- Battery level for node B: 77%
- Deep sleep time: 180 seconds (3 mins)
- **Used 2 ANDed boolean variable and a QoS 1 level to achieve the same behaviour as Pico**

Start | End time:

- Start time: 11:30
- End time: 17:30

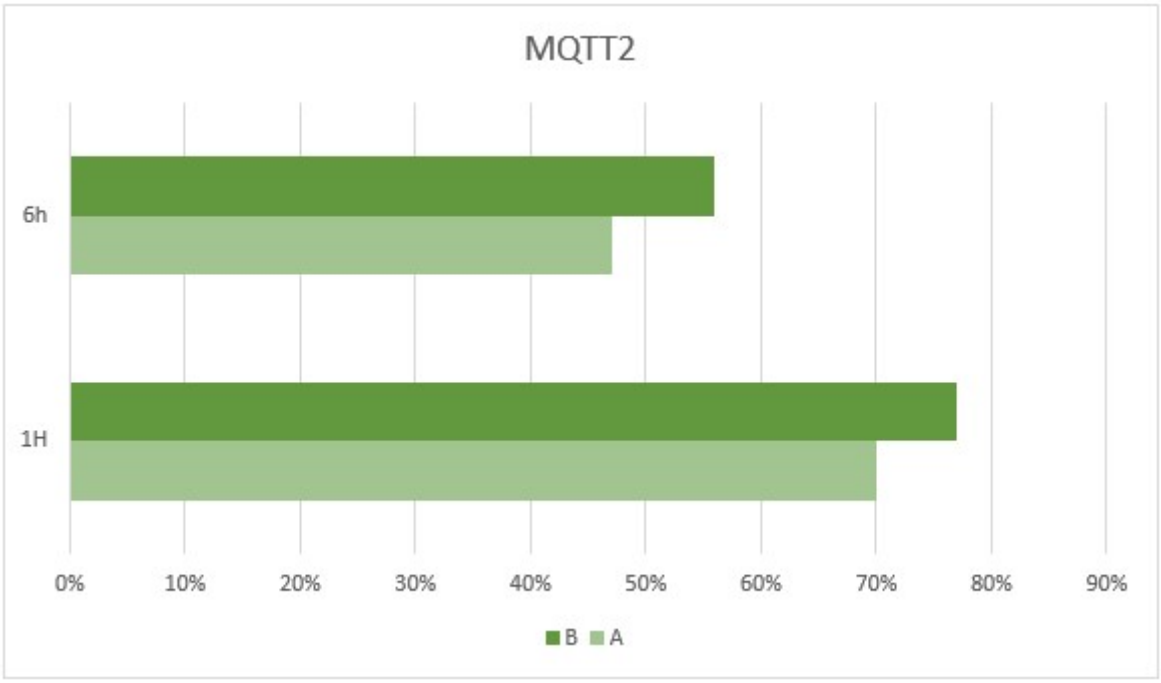
Remaining battery level after 6 hours of testing:

- Battery level for node A:47% | used battery: 23%
- Battery level for node B:56% | used battery: 21%

Remarks:

- I consider this test more reliable than the previous since it ran the correct amount of time and it is paragonable to Pico's benchmarks
- The behaviour of this MQTT publisher node was expected to be similar to a simple PicoMP publisher node. Indeed the battery usage identical.
- The collected data is available in the "Logs" folder
    - Logs\logs_mqtt_sub\2

Observed data:



	A	B
6H	47%	56%
0H	70%	77%

7 – **Conclusions:**

The executed test shown the results of the PICO-MP protocol from an energy efficiency point of view that stated how much battery level drained the pub and the pubSmart nodes against the MQTT publisher nodes in the same WSAN.

The executed tests shown coherent results by stating which was the most efficient one and the most inefficient one by highlighting the conditions that produced those results by implementing different types of formulaes in the same nodes.

If we compare the energetic performance data of the PICO-MP publisher nodes with the energetic performance data obtained by the MQTT publisher nodes we can state that the battery level after 6 hours of testing were pretty much the same, thus showing that a simple PICO-MP publisher node is equivalent to a MQTT publisher node because the action performed by both the nodes are very similar.

Instead, if we compare the energetic performance data of the PICO-MP pubSmart node with an MQTT publisher node we can state that the pubSmart has the same/worse performance compared to the MQTT publisher one because the pubSmart node performs more actions than a simple MQTT publisher node.

Take as example the test #4: the pubSmart node was **constantly** performing edge computing, analysing the collected data fom the sensor and sending the data through the network, while the MQTT pub was simply publishing the collected data.

But this showed better performance under the network throughput point of view (not valid for the test #4 obviously) for all the other pubSmart nodes test: draining more battery but avoiding unnecessary network flooding unlike the MQTT publisher did.

To be precise, the PICO-MP test #3.2 is not 100% fair because the node simply just did not connect to the WiFi network and the relative modules (WiFi, Bluetooth and radio) were still powered on but left idle, thus still consuming battery.

Turning completely off the communication modules on the ESP32 board while publishing data is not required is definitely going to be the next step of the project, to achieve an extremely energy, network and overall resource efficient WSAN.

8 - Sitography:

- COVID-19
    - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7504028/
    - https://link.springer.com/article/10.1007/s13762-020-02991-8#Sec10
    - https://www.elektronikpraxis.vogel.de/sensorfusion-und-kuenstliche-intelligenz-befeuern-die-sensortechnik-a-960939/
    - https://it.wikipedia.org/wiki/Composti_organici_volatili
- 18650
    - https://www.youtube.com/watch?v=1rg3ZWxBNUE
    - https://commonsensehome.com/18650-battery/
- Platformio
    - https://github.com/platformio/platformio-docs/issues/124
    - https://docs.platformio.org/en/latest/core/userguide/index.html#piocore-userguide
    - https://docs.platformio.org/en/latest/faq.html#platformio-udev-rules
    - https://docs.platformio.org/en/latest/frameworks/arduino.html#framework-arduino
- TTGO T-Beam
    - https://docs.platformio.org/en/latest/boards/espressif32/ttgo-t-beam.html
    - https://github.com/platformio/platform-espressif32/tree/master
    - https://docs.platformio.org/en/latest/platforms/espressif32.html
    - https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-upsources/#:~:text=The%20ESP32%20goes%20into%20deep,which%20GPIO%20touch%20was%20detected.
- Arduino
    - https://forum.arduino.cc/index.php?topic=428015.0

- ESP32
    - https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F
    - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
    - https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
    - https://www.instructables.com/ESP-NOW-WiFi-With-3x-More-Range/
    - https://www.espressif.com/en/products/software/esp-now/overview
    - https://www.espressif.com/en/support/documents/technical-documents
    - https://github.com/LetsControltheController
    - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/?utm_source=platformio.org&utm_medium=docs
    - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html
    - https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/
    - https://randomnerdtutorials.com/get-change-esp32-esp8266-mac-address-arduino/
    - https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/
    - https://github.com/espressif/esp-idf/tree/master/examples

- C++
    - https://stackoverflow.com/questions/1675351/typeadef-struct-vs-struct-definitions
    - https://embeddedartistry.com/blog/2017/07/26/stdstring-vs-c-strings/
    - https://embeddedartistry.com/blog/2017/07/26/stdstring-vs-c-strings/
    - https://www.splinter.com.au/sending-a-udp-broadcast-packet-in-c-objective
    - https://stackoverflow.com/questions/6729366/what-is-the-difference-between-af-inet-and-pf-inet-in-socket-programming#:~:text=AF_INET%20is%20the%20address%20family,case%20an%20Internet%20Protocol%20address).&text=the%20safest%20option.-,Meaning%2C%20AF_INET%20refers%20to%20addresses%20from%20the%20internet%2C%20IP%20addresses,protocol%2C%20usually%20sockets%2Fports.
- Udp client/server
    - https://www.bogotobogo.com/cplusplus/sockets_server_client.php
    - https://www.raspberrypi.org/forums/viewtopic.php?t=25569
    - http://www.iotsharing.com/2017/06/how-to-use-udpip-with-arduino-esp32.html
    - https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/examples/WiFiUDPClient/WiFiUDPClient.ino
    - https://stackoverflow.com/questions/26204553/need-file-descriptor-for-socket-in-udp
    - https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-recvfrom
    - https://www.abc.se/~m6695/udp.html

- BME680
    - https://randomnerdtutorials.com/esp32-bme680-sensor-arduino/
    - https://www.bluedot.space/products/bme680/
    - https://github.com/G6EJD/BME680-Example
    - https://forums.pimoroni.com/t/bme680-observed-gas-ohms-readings/6608/17
- Publish/subscribe model
    - https://www.ably.io/topic/pub-sub
    - https://cloud.google.com/pubsub/docs/overview
    - https://docs.microsoft.com/it-it/azure/architecture/patterns/publisher-subscriber
    - https://aws.amazon.com/it/pub-sub-messaging/
- Message queue
    - https://aws.amazon.com/it/message-queue/
- Event driven architectures
    - https://aws.amazon.com/it/event-driven-architecture/
- MQTT
    - https://diyiot.com/introduction-into-mqtt/
    - https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/
    - https://www.hivemq.com/mqtt-essentials/
    - https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/
    - https://diyiot.com/microcontroller-to-raspberry-pi-wifi-mqtt-communication/
    - https://raw.githubusercontent.com/RuiSantosdotme/Random-Nerd-Tutorials/master/Projects/ESP32/ESP32_MQTT/ESP32_BME280.ino

9 - Ringraziamenti:

Ringrazio uomino, compagno di studi, di consigli e di sclerate che mi ha sopportato e dato man forte durante questo viaggio. Senza di lui non ce l'avrei proprio fatta, grande uomì.

Ringrazio Fabio, che mi ha offerto questo progetto di tesi, senza di lui non mi sarei laureato, scusa se non ho presentato un lavoro completo.

Ringrazio la mia ragazza che mi ha aiutato durante la tesi mentre il mio stato mentale stava lentamente degradando e che non mi ha ucciso non ostante tutto. Volevo piangere mentre si è offerta di farmi i grafici.

Ringrazio mio fratello, mia cugina mia madre mio padre, tutta la mia famiglia e tutti i miei amici più stretti che in tutti questi anni mi hanno sostenuto, ascoltato ed aiutato non ostante le difficoltà che sembravano non finire mai.

Ringrazio tutte le nuove persone conosciute durante questo percorso e che hanno contribuito a renderlo piacevole.

Ma soprattutto ringrazio me stesso per averci messo tutto ciò che avevo.

Inoltre vorrei sfruttare i ringraziamenti per dire a Momo, Besjan e Ponzi di non lasciare gli studi e di non sprecare tutto questo tempo.
Siete arrivati alla fine, spero che questo incoraggiamento vi sproni ad andare avanti.

Simone