UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

University degree in
Computer Engineering for the Internet of Things

Course of
Network Aspects of Internet of Things

Project
"Tic Tac Toe using MQTT and CoAP"

Professor
Iera Antonio

Teaching assistant
Genovese Giacomo

Candidate
Fabio Capparelli 214490

INDICE

INTRODUZIONE

Il seguente elaborato ha come principale scopo la descrizione di un piccolo progetto che si basa sull'implementazione, studio e utilizzo di due protocolli di comunicazioni centrali nel mondo dell'Internet of Things, quali MQTT (Message Queue Telemetry Transport) e CoAP (Costrained Application Protocol) e in particolare, viene effettuato un confronto tra i due in termini di prestazione (risposta temporale).
Il progetto ha come soggetto l'implementazione di un piccolo e semplice gioco (Tic Tac Toe) che tutti conosciamo e al quale tutti abbiamo giocato, conosciuto anche come tris e grazie all'utilizzo di due board (ESP32) che comunicano tra loro, appunto tramite MQTT e CoAP, consentiamo partite multiplayer tra due giocatori. Per l'implementazione grafica sono utilizzati due display led touch. Caratteristica fondamentale, che spiegheremo meglio in seguito è quella della possibilità di scegliere il protocollo di comunicazione da utilizzare, ovviamente entrambe le board dovranno concordarsi sull'utilizzo del medesimo protocollo.

HARDWARE ADOPERATI

- ***2x SeeKool ESP WROOM 32***

MCU

- ESP32-D0WD-V3 embedded, Xtensa® dual-core 32-bit LX6 microprocessor, up to 240 MHz
- 448 KB ROM for booting and core functions
- 520 KB SRAM for data and instructions
- 16 KB SRAM in RTC

Wi-Fi

- 802.11b/g/n
- Bit rate: 802.11n up to 150 Mbps
- A-MPDU and A-MSDU aggregation
- 0.4 μs guard interval support
- Center frequency range of operating channel: 2412 ~ 2484 MHz

Bluetooth®

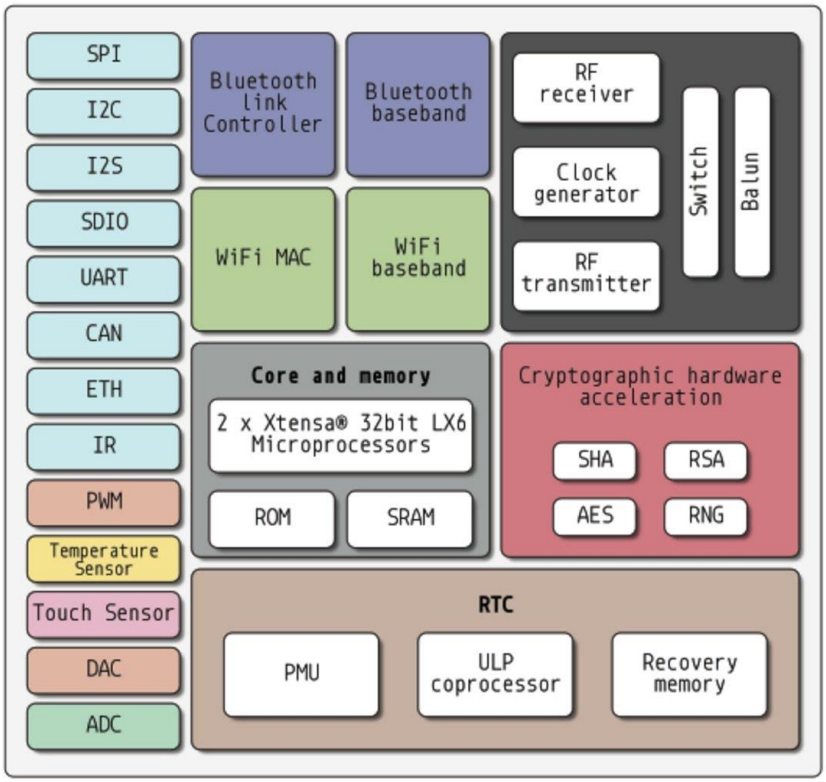
- Bluetooth V4.2 BR/EDR and Bluetooth LE specification
- Class-1, class-2 and class-3 transmitter
- AFH
- CVSD and SBC

Hardware

- Interfaces: SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
- 40 MHz crystal oscillator
- 4 MB SPI flash
- Operating voltage/Power supply: 3.0 ~ 3.6 V
- Operating temperature range: –40 ~ 85 °C
- Dimensions: See Table 1

Certification

- Bluetooth certification: BQB
- RF certification: FCC/CE-RED/SRRC
- Green certification: REACH/RoHS



Questa board, della famiglia esp wroom, è molto utile al fine di questo progetto, perché integra un'antenna wi-fi che consente la comunicazione senza fili con standard 802.11b/g/n a 2.4 GHz.
Dotato inoltre di connessione bluetooth (non utilizzata nel progetto) ed è dual core fino a 240MHz.
Utilissima e diffusissima in progetti e soluzioni che richiedono connettività e capacità elaborative superiori alle comunissime Arduino board con MCU "ATmega328p" che non integrano antenne wi-fi o bluetooth e sono dotate di un unico core. Altre differenze stanno nell'integrazione delle periferiche, ESP implementa la periferica I2S e ah dei pin capacitivi sensibili al tocco, e inoltre implementa non solo la conversione ADC, ma anche la conversione DAC.

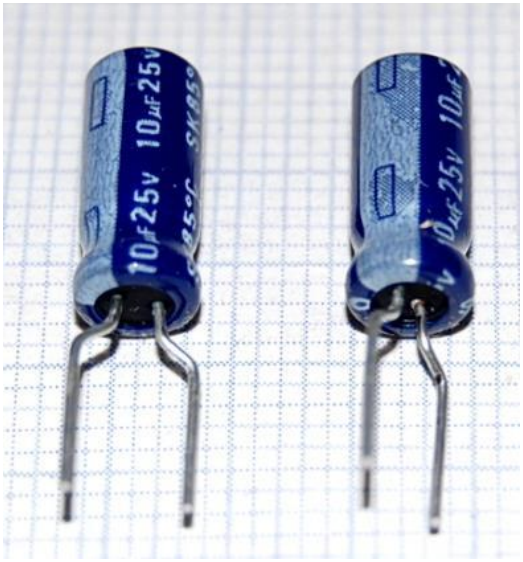
- ***2x HiLetgo 2.8" ILI9341 TFT LCD Screen Display Touch Panel SPI Serial 240*320***

Display touch screen utile per board che implementano la comunicazione SPI (Serial peripheral Iterface), grazie alla quale e all'utilizzo di 3 pin (MISO, MOSI e SCK), rispettivamente:

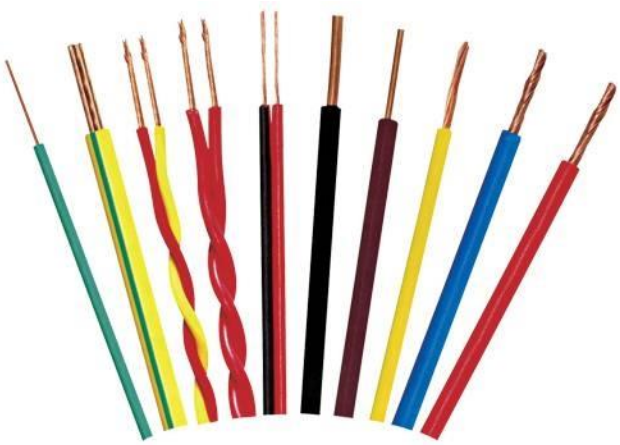
MISO = master in slave out; MOSI = master out slave in; SCK = serial clock.

Il display funge da slave, cioè riceverà dati (comandi) dall'esp32 che sarà il master, il rate di invio dei bit è guidato dal segnale di clock condiviso.
Il display utilizzato è capacitivo, cioè sensibile al tocco, esso ha installato al di sotto del display un pannello touch, il display è led a 2.8'' e 240x320 pixel. Dotato inoltre, di pennino e possibilità di lettura della scheda sd.

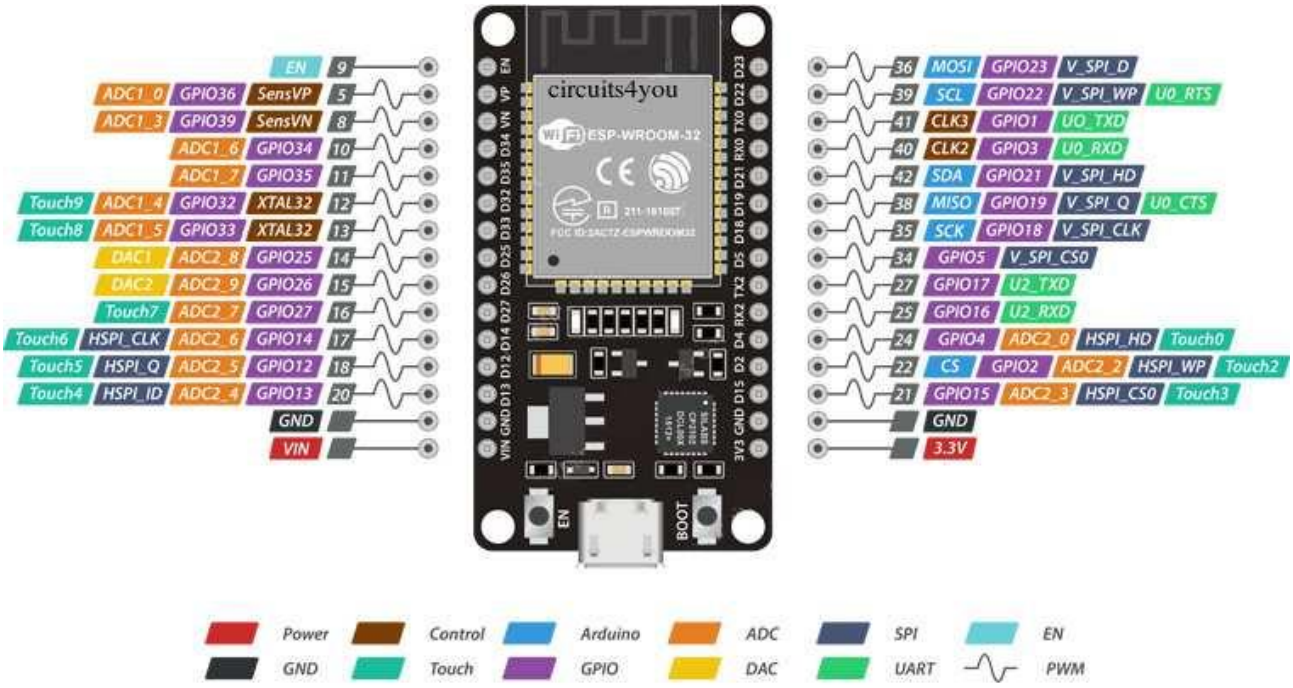
- ***2x condensatori da 10uF***

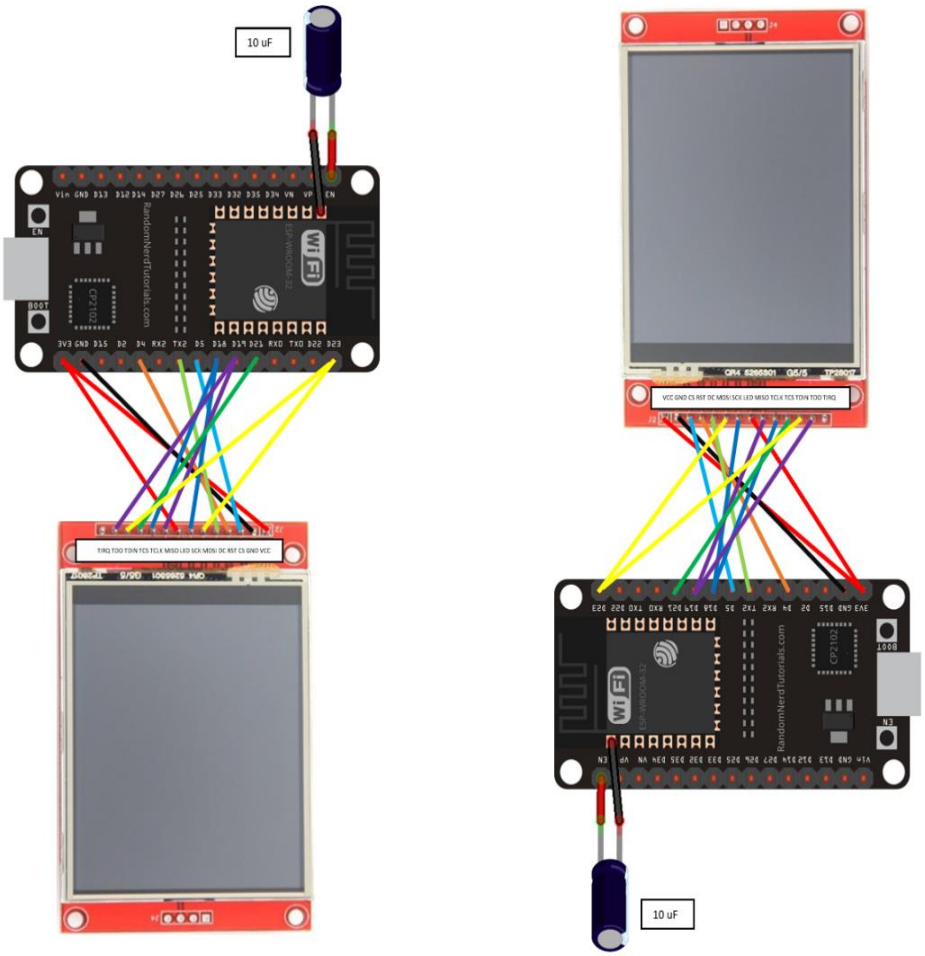


- ***N x cavi elettrici***



SCHEMA CIRCUITALE





vcc Condensatore → pin EN

gnd Condensatore → pin GND

vcc Display → pin 3.3V

gnd Display → pin GND

cs Display → pin GPIO5

rst Display → pin GPIO4

dc Display → pin GPIO17 / TX2

mosi Display → pin GPIO23

sck Display → pin GPIO18

led Display → pin 3.3 V

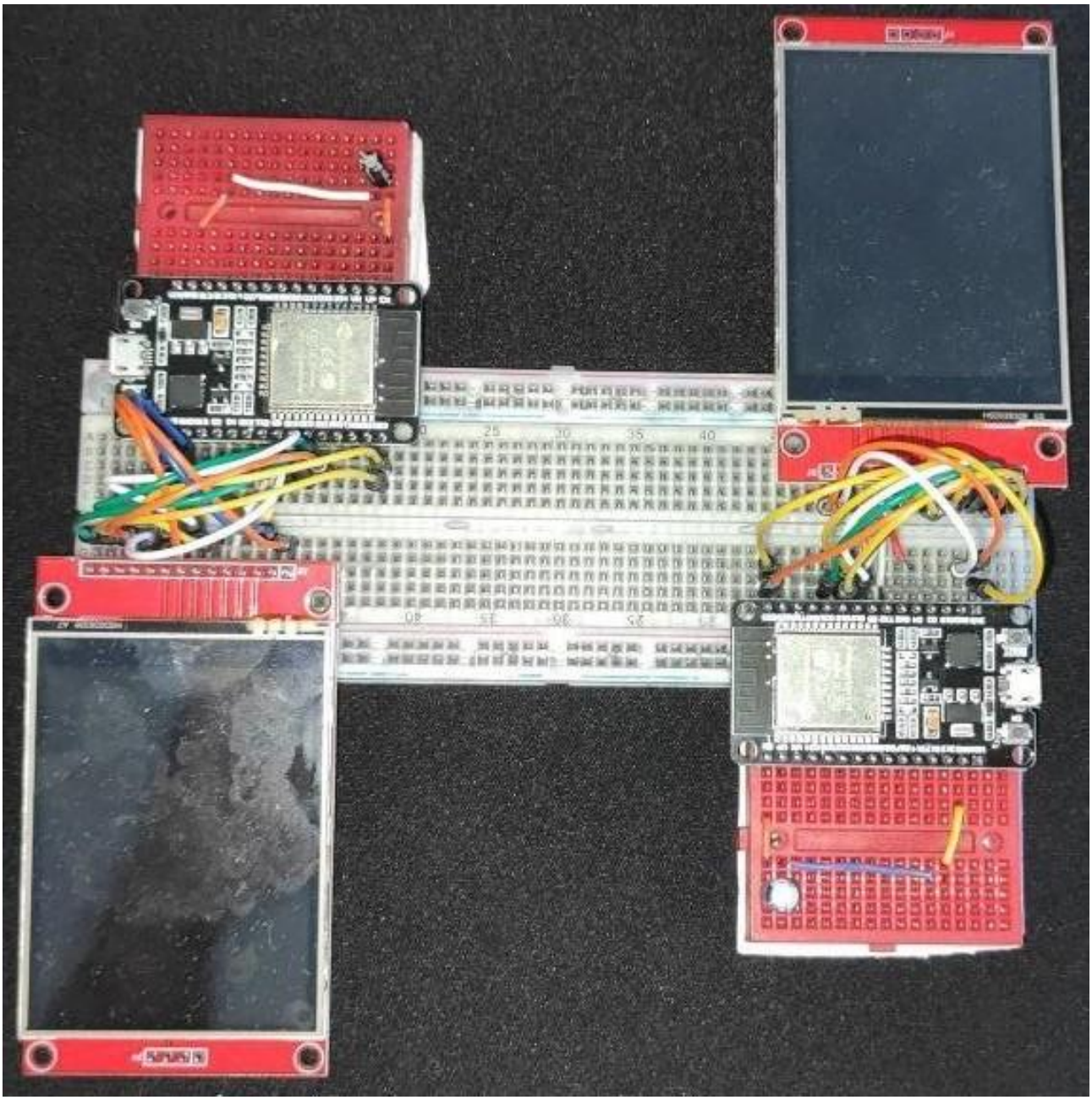
miso Display → pin GPIO19

t_clk Display → pin GPIO18

t_cs Display → pin GPIO21

t_din Display → pin GPIO23

t_do Display → pin GPIO19

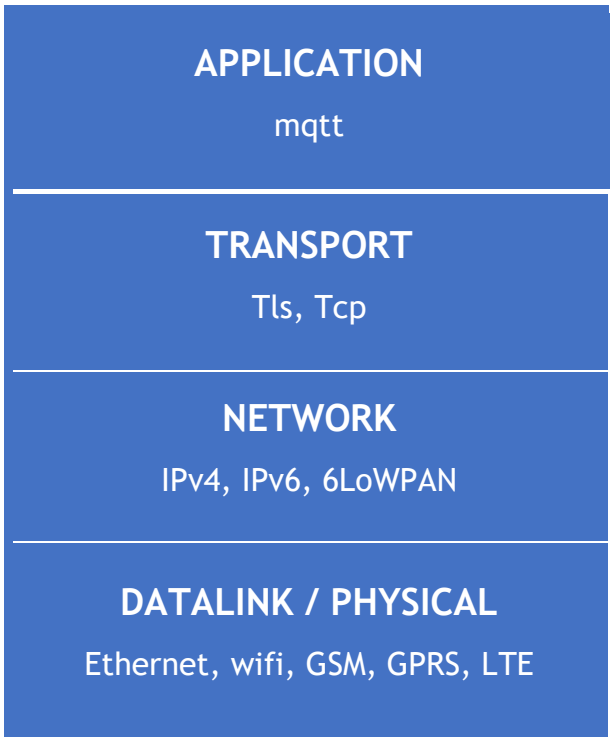


PROTOCOLLI USATI

MQTT – Message queue telemetry transport

MQTT è un protocollo basato sullo scambio di messaggi leggeri proposto inizialmente da IBM nel 1999 e successivamente standardizzato ne 2014 da Oasis (v 3.1.1). E' disponibile anche una versione di MQTT per le Sensor Network riconosciuto con l'acronimo MQTT-SN. Adatto per applicativi e comunicazione di tipo M2M (machine-to-machine), per le WSN (wireless sensor network) e negli ultimi anni utilizzato in progetti e applicativi IoT, ovvero scenari in cui sensori e attuatori comunicano i loro dati/azioni attraverso questo protocollo.

- ***MQTT stack***



A sinistra è mostrato lo stack MQTT, esso si posiziona sullo stack TCP/IP, cioè è responsabile dello scambio di messaggi a livello applicativo e orientato alla connessione grazie all'utilizzo di TCP e garantisce un'adeguata sicurezza grazie all'utilizzo di TLS (transport layer security), cioè sicurezza di livello trasporto. Inoltre, può implementare a livello rete diversi protocolli tra cui IPv4, IPv6 e 6LoWPAN, quest'ultimo se implementato produce una compressione dell'header TCP. A livello datalink e fisico è possibile utilizzare standard wi-fi (radio), ethernet, LTE ecc.

Il modello MQTT è basato su una comunicazione asincrona di messaggi o eventi e sull'accodamento degli stessi, gli attori principali sono un client e un server (broker), i quali seguono una politica di publish/subscribe, ovvero di pubblicazione e sottoscrizione.

- *Gli attori principali sono*:
  - Client → un programma o dispositivo che stabilisce una connessione al server (broker), esso può:
    1. Pubblicare messaggi ai quali gli altri client potrebbero essere interessati;
    2. Sottoscriversi a un topic per ricevere messaggi da altri client.
  - Server → il broker, ovvero l'entità principale che accetterà le connessioni dei client e sarà responsabile dell'archiviazione e ricezione dei messaggi pubblicati dai client, o che invierà i messaggi sottoscritti dai client.
  - Messaggio → i dati inviati nel payload del pacchetto.
  - Topic → un'etichetta associata al messaggio e utilizzata per le pubblicazioni o sottoscrizioni. ES: (topic, message) → (temperature, 27 °C).
  - Topic filter → un'espressione contenuta nel topic, cioè una stringa parziale. ES: topic = temperature; topic_filter = temp.
  - MQTT control packet → il pacchetto di informazioni inviato sulla rete, MQTT definisce 14 tipi di pacchetto, uno dei quali, il PUBLISH è utilizzato per inviare i messaggi.

- ***MQTT control packet***

Mqtt utilizza pacchetti di controllo per la comunicazione, e come detto sopra essi si distinguono in 14 tipi. Caratteristica principale di MQTT è la leggerezza del pacchetto, poiché esso utilizza un header MQTT di soli 2 byte.

FIXED HEADER (sempre presente) Size 2 byte	VARIABLE HEADER (opzionale) Size variabile	PAYLOAD (opzionale) Size variabile

Fixed header →

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

Variable header → (packet identifier)

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

Nome	Valore	Direzione	Descrizione	Packet id	Fixed header flags Bit 0 - 1 - 2 - 3	Payload
Reserved	0					
CONNECT	1	Client → Server	Client request to connect	NO	0 0 0 0	Richiesto
CONNACK	2	Server → Client	Connection ack	NO	0 0 0 0	Null
PUBLISH	3	Client → Server Server → Client	Publish message	SI (QoS>0)	DUP QoS QoS RETAIN	Opzionale
PUBACK	4	Client → Server Server → Client	Publish ack	SI	0 0 0 0	Null
PUBREC	5	Client → Server Server → Client	Publish received	SI	0 0 0 0	Null
PUBREL	6	Client → Server Server → Client	Publish release	SI	0 0 1 0	Null
PUBCOMP	7	Client → Server Server → Client	Publish complete	SI	0 0 0 0	Null
SUBSCRIBE	8	Client → Server	Client subscription	SI	0 0 1 0	Richiesto
SUBACK	9	Server → Client	Subscribe ack	SI	0 0 0 0	Richiesto
UNSUBSCRIBE	10	Client → Server	Unsubscribe request	SI	0 0 1 0	Richiesto
UNSUBACK	11	Server → Client	Unsubscribe ack	SI	0 0 0 0	Null
PINGREQ	12	Client → Server	Ping request	NO	0 0 0 0	Null
PINGRESP	13	Server → Client	Ping response	NO	0 0 0 0	Null
DISCONNECT	14	Client → Server	Client disconnection	NO	0 0 0 0	Null
Reserved	15					

DUP[1] = Duplicate delivery of a PUBLISH Control Packet
QoS[2] = PUBLISH Quality of Service
RETAIN[3] = PUBLISH Retain flag

- ***MQTT quality of service flag (QoS)***

Questo campo indica il livello di sicurezza d'invio di un messaggio, riassunti di seguito:

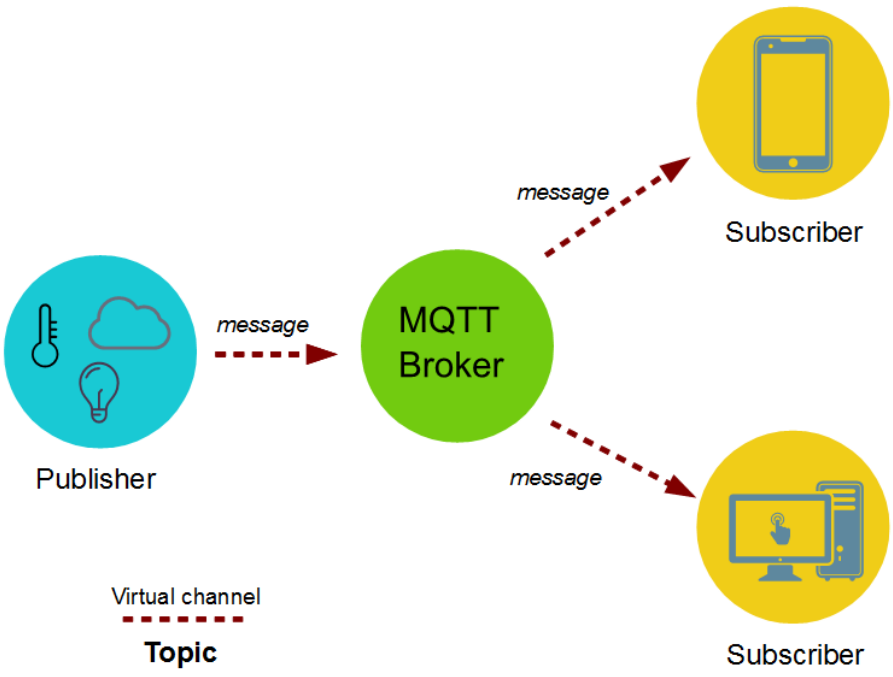
QoS value	Bit 2	bit 1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved – must not be used

- ***MQTT duplicate flag (DUP)***

se DUP è settato a 0 indica il primo tentativo di invio del PUBLISH control packet tra Client e server, se è settato a 1 vuol dire che si sta facendo un altro tentativo di invio. Esso però è utilizzato quando QoS è settato a 1 o a 2, è sempre 0 quando QoS è 0.

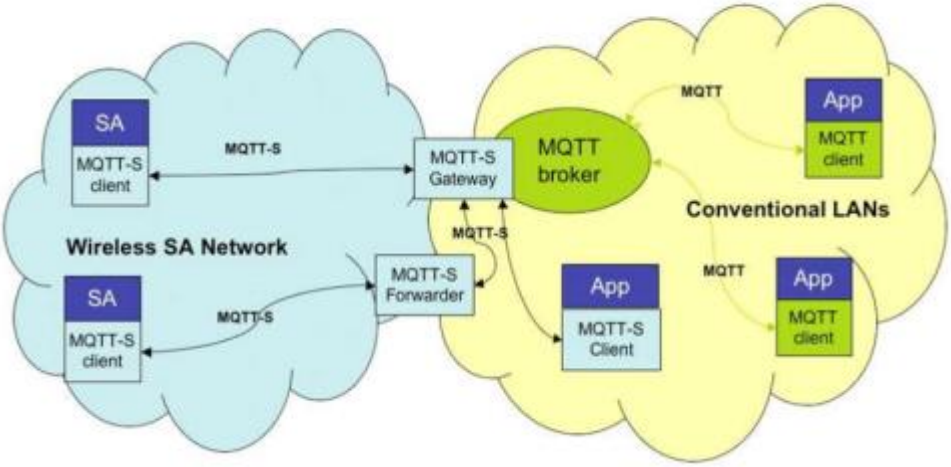
- ***MQTT retain flag (RETAIN)***

Se settato a 1 il PUBLISH contro packet deve essere archiviato in maniera persistente dal server per invii successivi a client sottoscritti dormienti. Se è 0 il messaggio è inviato solo ai client attivi.



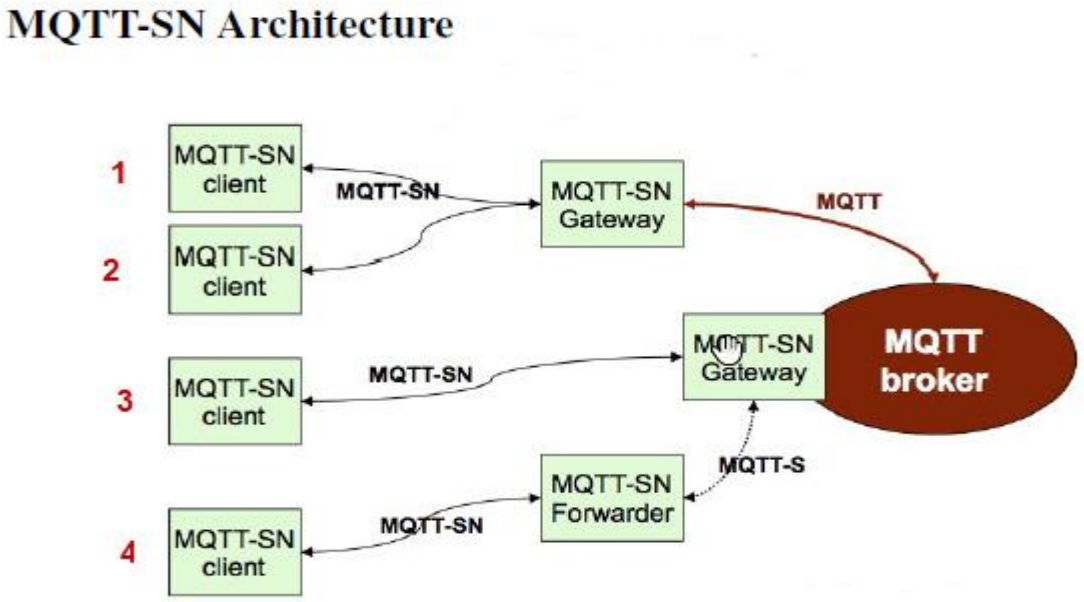
- ***MQTT-SN per Wireless Sensor Network***

Le wireless sensor network solitamente non implementano uno stack TCP/IP, ma un proprio stack con ZigBee, Bluetooth ecc. Per garantire maggiore leggerezza ed efficienza a dispositivi limitati. Di conseguenza non è possibile un'implementazione diretta ai broker (server) MQTT, bisogna quindi, consentire la connessione delle WSN a reti TCP/IP, le quali comunicheranno con il broker MQTT. La soluzione che si è trovata è quella di introdurre gateway TCP/IP intermedi, ai quali le WSN si connetteranno con MQTT-SN e avranno a disposizione le funzionalità di reti TCP/IP e la con nessione al broker tramite MQTT.



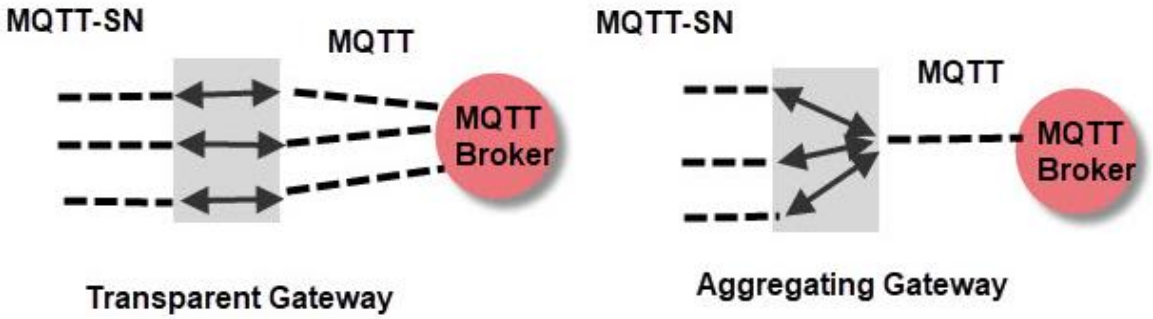
Le novità introdotte sono le seguenti:

1. Utilizzo di un ID per i topic e non più una stringa;
2. I topic ID predefiniti non richiedono registrazione;
3. Procedure di ricerca dei broker dinamica, senza aver bisogno di configurare il broker staticamente;
4. Riduzione della size dei payload;
5. Utilizzo di UDP e non più TCP, quindi la connessione al broker è "virtuale";
6. Risveglio e ricezione di messaggi bufferizzati per i client dormienti.



MQTT-SN gateway sono di due tipi:

- Trasparent = dove ogni connessione MQTT-SN corrisponde ad un'unica connessione MQTT;
- Aggregation = dove connessioni MQTT-SN multiple corrispondono ad un'unica connessione MQTT.



CoAP - Constrained Application Protocol

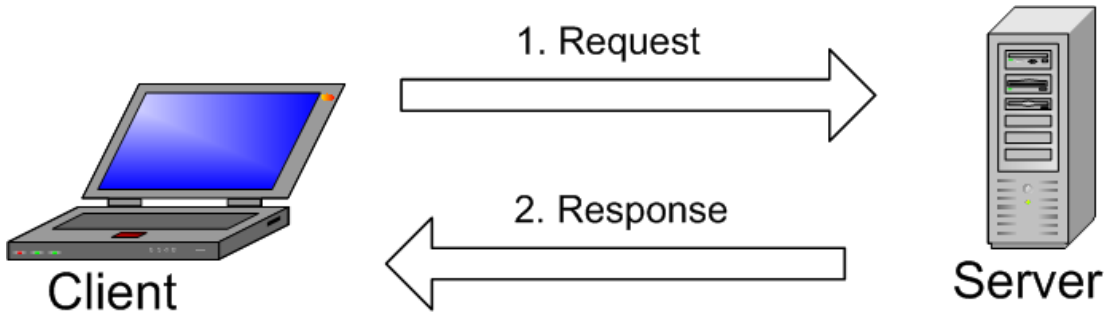
CoAP è un protocollo basato sullo scambio di messaggi da parte di device limitati in termini di consumo di energia, capacità di comunicazione o reti poco affidabili ecc. Protocollo di livello applicativo definito in RFC 7252, esso garantisce la comunicazione a dispositivi attraverso internet utilizzando protocolli e stack più leggeri rispetto al TCP/IP e anche utile nei casi di connettività mobile. CoAP è incentrato sul fornire servizi e progettato per tradurre e alleggerire HTTP, essendo basato su un'architettura RESTful e un modello di richiesta/risposta come HTTP. CoRE (Constrained RESTful Environments Working Group) è un IETF group che ha fornito la maggiore standardizzazione di questo protocollo.

- ***CoAP stack***

APPLICATION CoAP
TRANSPORT UDP, dtls
RETE IPv6, 6LoWPAN, RPL
DATALINK / PHYSICAL Ethernet, wifi, GSM, GPRS, LTE

A livello trasporto CoAP utilizza UDP, molto più leggero di TCP, e DTLS (Datagram Transport Layer Securuty) per proteggere la privacy dei dati scambiati con CoAP e per evitare intercettazione, manomissioni o falsificazioni dei messaggi CoAP.
A livello rete implementa protocolli come IPv6, 6LoWPAN per maggiore leggerezza o anche RPL routing protocol per reti LLN.

Basato su una comunicazione asincrona o sincrona richiesta/risposta e distinzione tra metodi CoAP e transactions CoAP.



- ***CoAP message***

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	32	VER		Type		Token Length				Request/Response Code								Message ID															
8	64	Token (0 - 8 bytes)																															
12	96																																
16	128	Options (If Available)																															
20	160	1	1	1	1	1	1	1	1	Payload (If Available)																							

  - HEADER CoAP → i primi 4 bytes solo per l'header CoAP.
    - VER (2 bits) : versione del protocollo CoAP utilizzata.
    - Type (2 bits) : descrive il tipo di datagramma inviato rispettivamente per richiesta e risposta.

      Richiesta:

      0: Confirmable (CON) = il messaggio di richiesta richiede l'ack.

      1: Non-Confirmable (NON) = la richiesta non richiede ack.

      Risposta:

      2: Acknowledgement (ACK) = l'ack corrispondente al CON.

      3: Reset (RST) = questo messaggio indica che il CON è ricevuto ma qualcosa è mancante.
    - Token Length (4 bits): indica la lunghezza del campo token, che è variabile e occupa $2^4 = 32\ bits = 8\ bytes$.
    - Request/Response Code (8 bits) = i primi 3 bit sono per la classe, (cioè la tipologia di risposta o richiesta) gli ultimo 5 sono per il codice.

0	1	2	3	4	5	6	7
Class			Code				

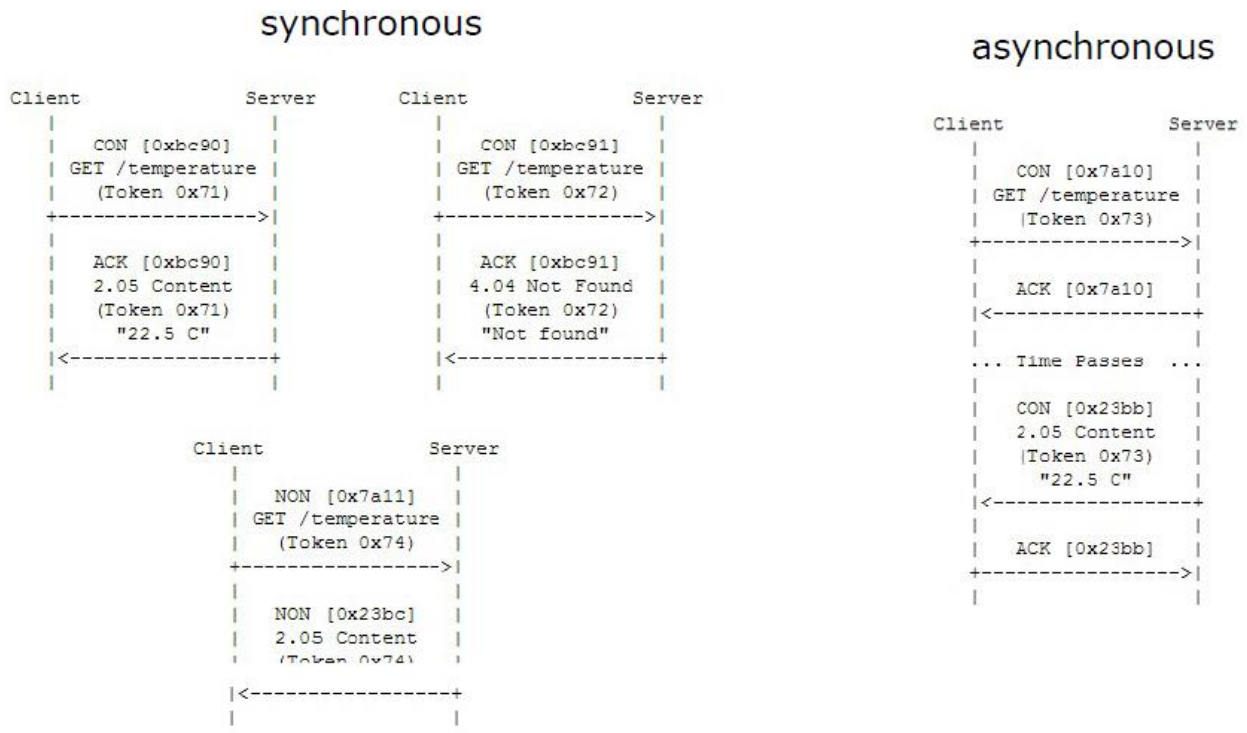
- Method: 0.XX
  0. EMPTY
  1. GET
  2. POST
  3. PUT
  4. DELETE
  5. FETCH
  6. PATCH
  7. iPATCH
- Success: 2.XX
  1. Created
  2. Deleted
  3. Valid
  4. Changed
  5. Content
  31. Continue
- Client Error: 4.XX
  0. Bad Request
  1. Unauthorized
  2. Bad Option
  3. Forbidden
  4. Not Found
  5. Method Not Allowed
  6. Not Acceptable
  8. Request Entity Incomplete
  9. Conflict
  12. Precondition Failed
  13. Request Entity Too Large
  15. Unsupported Content-Format
- Server Error: 5.XX
  0. Internal Server Error
  1. Not Implemented
  2. Bad Gateway
  3. Service Unavailable
  4. Gateway Timeout
  5. Proxying Not Supported
- Signaling Codes: 7.XX
  0. Unassigned
  1. CSM
  2. Ping
  3. Pong
  4. Release
  5. Abort

        - Message ID (16 bits) : usato per rilevare duplicati e per accoppiare messaggi di tipo ack/reset to Confirmable/non-Confirmable, cioè per accoppiare richiesta e risposta. Il messaggio di risposta avrà lo stesso message ID del messaggio di richiesta.
    - TOKEN → variabile da 0 a 8 bytes è un campo opzionale la cui size è indicata nell'header dal campo token length, generata dal client. È utilizzato come un identificatore locale per il client per forni contenuti extra in transazioni concorrenti.
    - OPTIONS : campo opzionale e variabile in dimensione, utilizzato per l'opzione di Observe.
    - PAYLOAD : campo opzionale e di dimensione variabile, conterrà i dati relativi al messaggio di richiesta o risposta.

- ***CoAP URI e principali METODI***

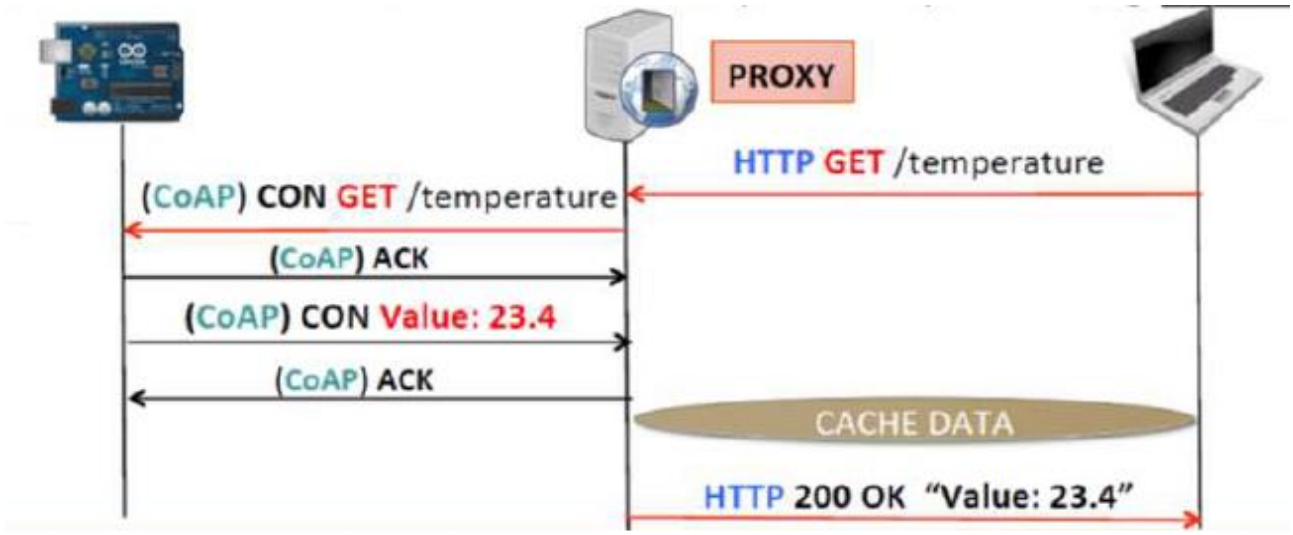
URI è l'acronimo di (Uniforme Resource Identifier) esso è utilizzato per identificare univocamente ogni risorsa posseduta dal server e soggetta alle richieste del client. Una risorsa può essere vista come una coppia <chiave, valore> = <URI, value>. Esempio banale è : <temperature, 27°C>.

Metodo	Descrizione
GET	Usato per richiedere informazioni sul valore di una risorsa
POST	Usato per creare una nuova risorsa (nuovo URI) e il suo corrispondente valore
PUT	Usato per modificare/aggiornare il valore di una risorsa
DELETE	Usato per eliminare una risorsa identificata da un URI
OBSERVE	Usato per sottoscriversi e ricevere aggiornamenti costanti sul valore di una certa risorsa (simile alla sottoscrizione di MQTT).



- ***CoAP proxy e caching***

CoAP implementa, anche se in maniera limitata, alcune funzionalità di HTTP, grazie alle quali è possibile implementare anche meccanismi di caching. Il trucco sta nel fatto che il client, nel richiedere il valore di una risorsa (metodo GET) lo fa attraverso un proxy intermedio e attraverso una richiesta http. Il proxy inoltrerà tale richiesta al server che può essere un dispositivo IoT (Arduino o ESP32) attraverso una richiesta CoAP. A questo punto la risposta da parte del server avviene sempre tramite CoAP e al proxy il quale, utilizzando http contatterà nuovamente il client e gli inoltrerà la risposta che a questo punto, con http, potrà essere memorizzata nella cache.



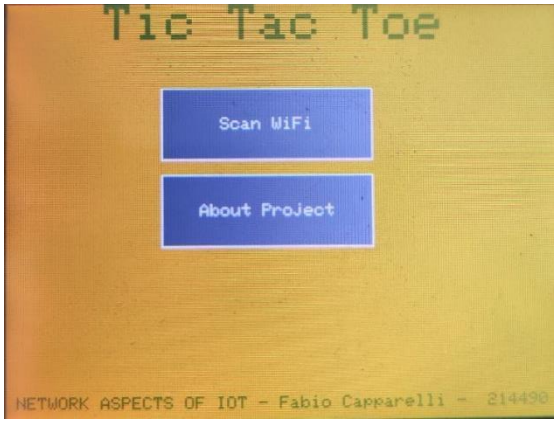
IMPLEMETAZIONE PROGETTUALE

Overview e descrizione delle fasi

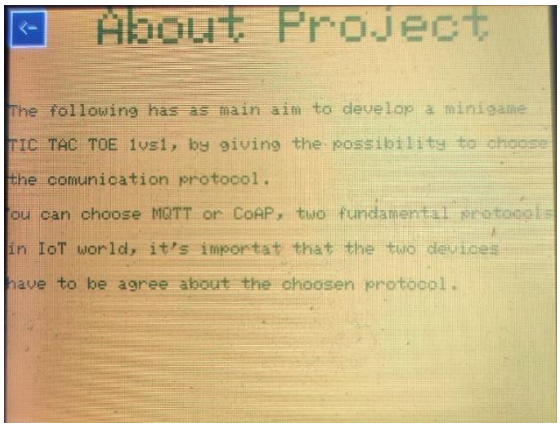
Di seguito è riportata una descrizione completa del progetto e sono presentate le diverse fasi o passi del prototipo. Ovvero il codice è suddiviso in step, in base allo screen (display) in cui ci si trova.
Le principali fasi sono 3:

1. Connessione alla rete locale tramite WI-FI;
2. Scelta del protocolla da utilizzare;
3. Entrata in modalità gioco, scelta del segno 'O' o 'X' e poi gioco.

- ***Fase 0 – Screen di default***

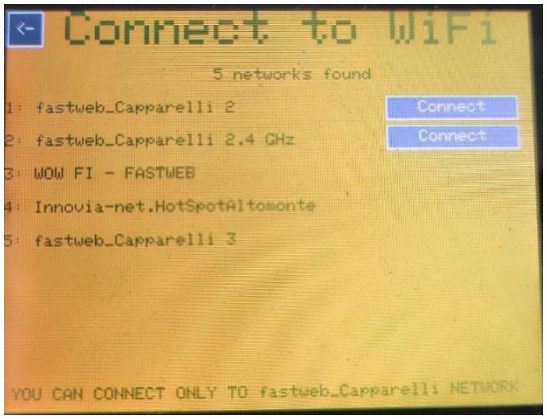


Questa è la pagina di default, quella che appare quando l'esp32 è alimentato, qui abbiamo una duplice scelta, faro lo scan delle reti wifi disponibili tramite il pulsante "Scan WiFi", oppure tramite "About Project" è possibile leggere una breve intro riguardo il progetto, mostrata di seguito.



"*the following has as main aim to develop a minigame, TIC TAC TOE 1vs1, by giving the possibility to choose the communication protocol.*
You can choose MQTT or CoAP, two fundamental protocols in IoT world, it's important that the two devices have to be agree about the chosen protocol."

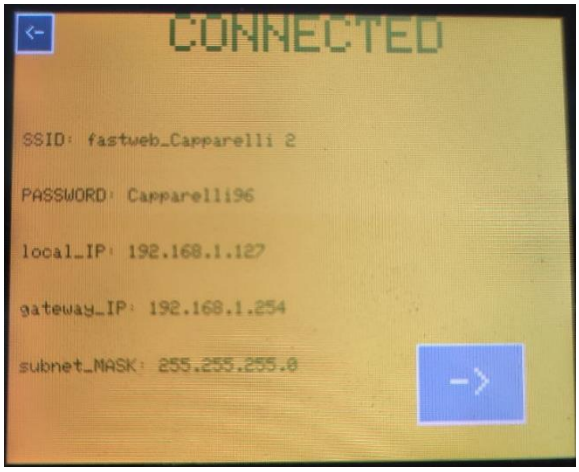
- ***Fase 1 – Connessione al WI-FI***



Dopo la scansione delle reti disponibili è possibile connettersi però solo a due delle cinque reti trovate, cioè:

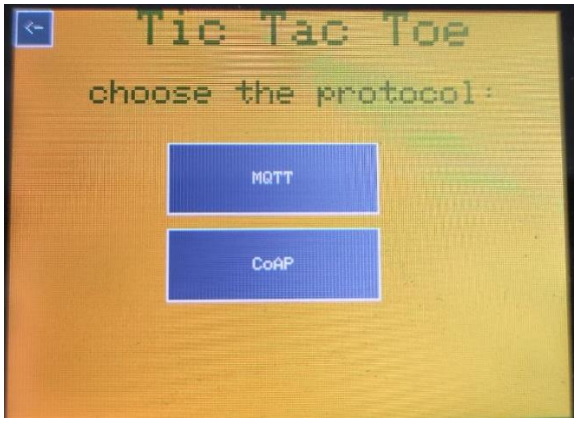
“fastweb_Capparelli 2”
“fastweb_Capparelli 2.4GHz”

Questo perché solo queste due reti condividono il medesimo range di indirizzi di rete locale. “fastweb_Capparelli 3” espone una propria rete locale poiché è un piccolo router che estende la rete principale (fastweb_Capparelli 2.4GHz).



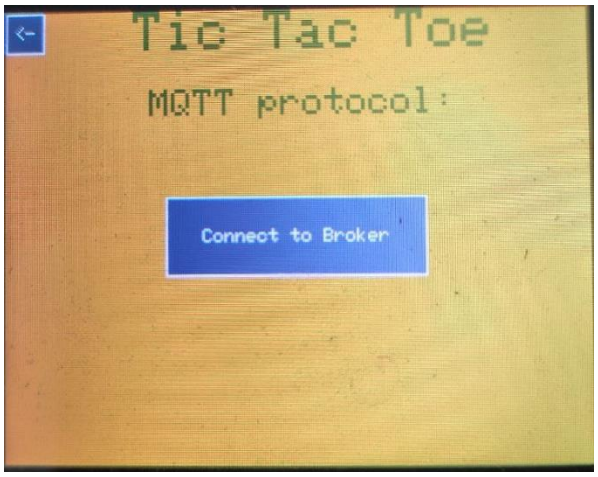
- *Player 1 = 192.168.1.127*
- *Player 2 = 192.168.1.101*

- ***Fase 2 – Scelta protocollo***

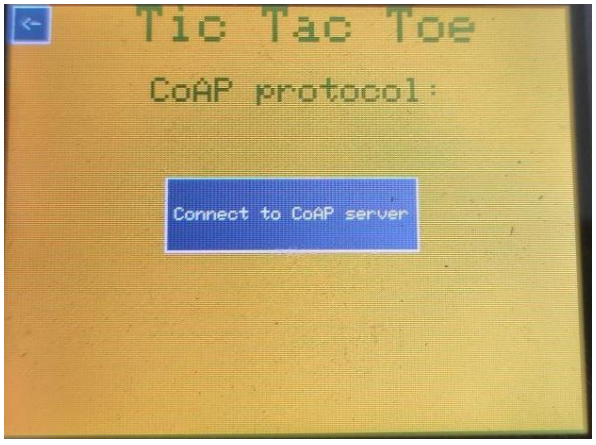


Qui è possibile scegliere il protocollo di comunicazione dei due player, è necessario che entrambi siano d'accordo sul medesimo protocollo.

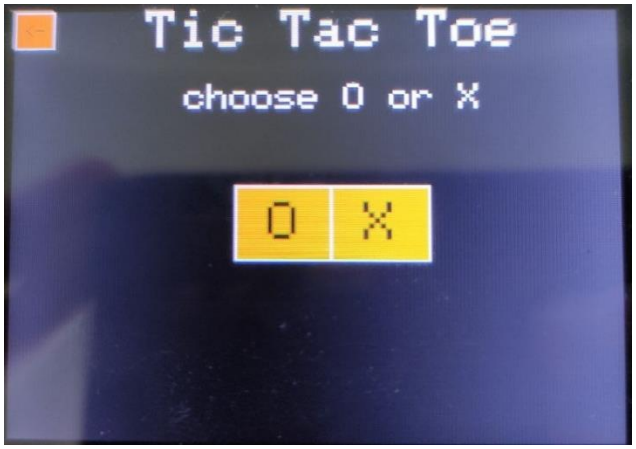
Per MQTT bisogna connettersi al broker centrale (server), implementato attraverso Mosquitto col mio PC, e indirizzo ip 192.168.1.146.



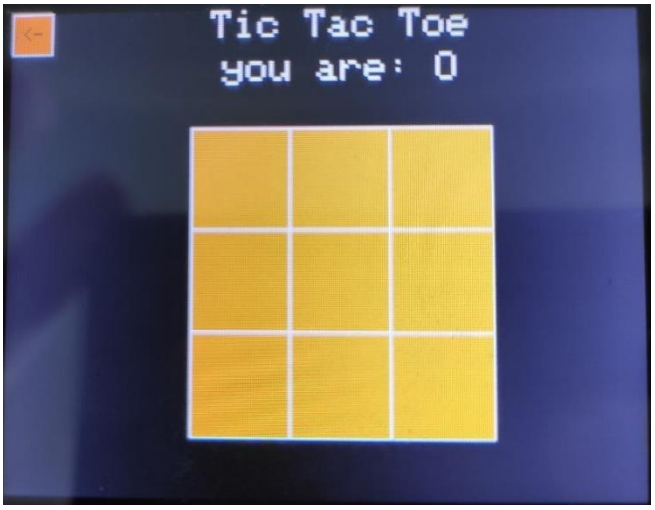
Per CoAP è necessario settare se stessi come server, qui non è necessaria alcuna entità centrale predisposta per lo scambio di messaggi, ma lo scambio avviene in maniera diretta.



- ***Fase 3 - Gioco***



Adesso si entra in modalità gioco, i due player dopo aver cliccato play e aver scelto il medesimo protocollo si troveranno questa schermata, il primo che sceglierà sarà il primo cominciare il gioco e ovviamente produrrà il cambio di schermata sul display del player che non ha scelto.



In conclusione, sono mostrati i differenti screen in caso di vincita, perdita o pareggio.

Parte software

Le componenti software utilizzate sono:

- Arduino IDE = utilizzato per scrivere il codice per gli ESP32.
- Mosquitto = utilizzato per avviare il broker MQTT sul localhost del mio PC.
- WireShark = utilizzato per catturare il traffico di rete WiFi tra gli ESP 32 e il broker MQTT.

Di seguito verranno illustrate nel dettaglio le scelte di programmazione effettuate, le parti di codice ovviamente più interessanti e inerenti alla comunicazione coi protocolli, nonché le scelte strategiche effettuate per programmare l'intero gioco e l'uso dei protocolli MQTT e CoAP. Si procederà, come fatto precedentemente per le schermate, a step, nel seguente modo:

1. Setup() e Loop().
2. Utilizzo del display tft touch.
3. Connessione alla rete WiFi locale.
4. Comunicazione con MQTT.
5. Comunicazione con CoAP.
6. Metodi principali utili al gioco.

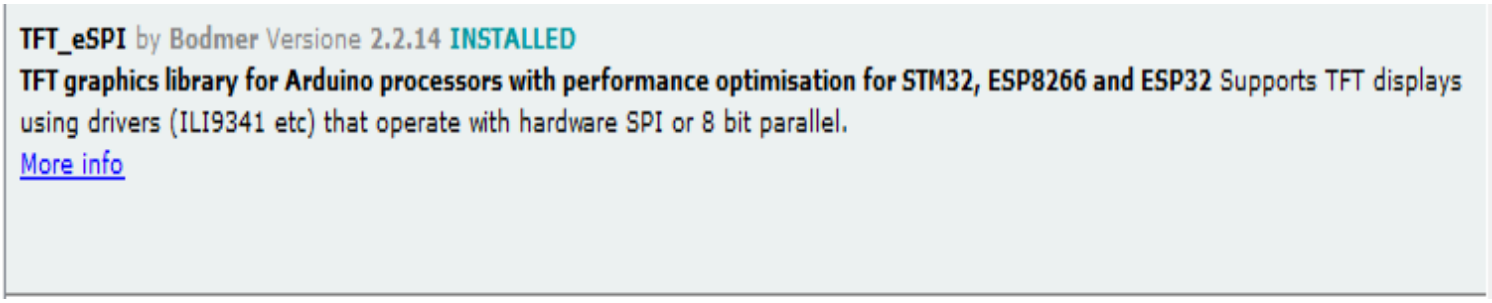
Setup() e Loop()

```
void setup() {
  caso = 0;

  tft.init();
  tft.setRotation(3); //horizontal
  touch_calibrate();

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  emptyMatrix();

  defaultScreen();
}

void loop() {
  uint16_t x, y;
  if (tft.getTouch(&x, &y))
       eventScreen(caso, x, y);
  eventScreen(caso, -1, -1);
}
```

Nel setup() vengono inizializzati rispettivamente il display, il wifi, la matrice 3x3 utilizzata per archiviare le mosse dei giocatori e viene lanciato lo screen di default.

Nel loop() viene lanciato sempre il metodo eventScreen(), che è utilizzato per riprodurre e rilevare gli eventi nelle varie schermate, l'evento corrispondente alla schermata in cui ci troviamo è denotato dalla variabile caso, inizializzata a 0 nel setup(). Se avviene un tocco, passiamo il punto toccato, altrimenti -1 -1, questo perché rimaniamo sempre attivi durante la comunicazione coi protocolli.

- ***Display tft***



Questa è la libreria utilizzata per la comunicazione tramite SPI tra il display tft e l'esp32, nel setup() notiamo l'inizializzazione dello stesso.

Mostriamo i due metodi principali utilizzati per comporre gli oggetti nelle varie schermate, che essenzialmente sono due: bottoni e label di testo. E anche il metodo della schermata principale, nella trattazione ometteremo il codice di tutte le schermate per brevità e per

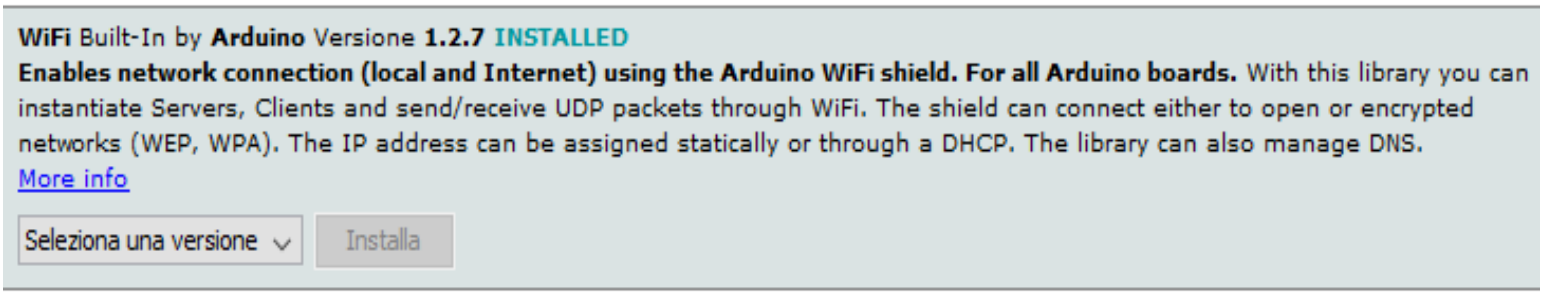
ridurre ridondanza nei metodi, anche perché questo non è argomento del corso.

```
void setText(String text, int x, int y, int s,  int color){
  tft.setTextColor(color);
  tft.setTextSize(s);
  tft.setTextDatum(MC_DATUM);
  tft.drawString(text, x, y);
}


void setBtn(String label, int x, int y, int w, int h, int s, int fillColor, int labelColor, int cornerColor){
  tft.drawRect(x-1, y-1, w+2, h+2, cornerColor);
  tft.fillRect(x, y, w, h, fillColor);
  tft.setTextColor(labelColor);
  tft.setTextSize(s);
  tft.setTextDatum(MC_DATUM);
  tft.drawString(label, x + (w/2), y + (h / 2));
}
```

Questi due metodi sono utilizzati, il primo per settare delle label di testo e il secondo per settare i pulsanti, sono di facile intuizione.

```
void defaultScreen(){
  tft.fillScreen(TFT_ORANGE);

  setText("Tic Tac Toe", 155, 0, 3, TFT_OLIVE);
  setBtn("Scan WiFi", 90, 50, 120, 40, 1, TFT_DARKGREY, TFT_WHITE, TFT_WHITE);
  setBtn("About Project", 90, 100, 120, 40, 1, TFT_DARKGREY, TFT_WHITE, TFT_WHITE);

  setText(" NETWORK ASPECTS OF IOT - Fabio Capparelli -  214490", 0, 230, 1, TFT_OLIVE);
}
```

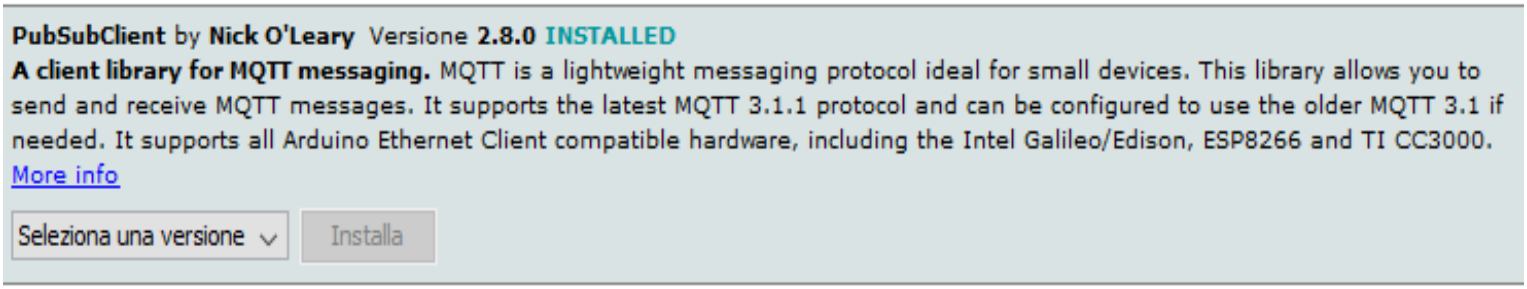
- ***Connessione al WiFi***



Questa è la libreria utilizzata per la connessione alla rete locale Wireless. Di seguito riportiamo il caso 1 del metodo eventScreen, che è utilizzato per la connessione alla rete desiderata.

```
      const char PASSWORD[] = "Capparelli96"; //same password for all networks

      uint8_t nNetworks = WiFi.scanNetworks();
      if (nNetworks == 0)
      {
        setText("Scanning . . .", 170, 40, 2, TFT_ORANGE);
        setText("no networks found", 170, 40, 1, TFT_OLIVE);
      }
      else
      {
        setText("Scanning . . .", 170, 40, 2, TFT_ORANGE);
        setText(" " + String(nNetworks) + " networks found", 170, 40, 1, TFT_OLIVE);
        setText(" YOU CAN CONNECT ONLY TO fastweb_Capparelli NETWORK", 0, 230, 1, TFT_OLIVE);

        uint8_t c=0;
        for (int i = 0; i < nNetworks; ++i)
        {
          String ssid = WiFi.SSID(i);
          if(ssid.startsWith("fastweb_Capparelli 2"))
          {
            ssidNet[c] = ssid;
            setText(String(c+1) + ": " + ssid, 0, (c+3)*20, 1, TFT_OLIVE);
            setBtn("Connect", 230, (c+3)*18, 80, 13, 1, TFT_DARKGREY, TFT_WHITE, TFT_WHITE);
            c++;
          }
        }
        nNet = c;

        for (int i = 0; i < nNetworks; ++i)
        {
          String ssid = WiFi.SSID(i);
          if(!ssid.startsWith("fastweb_Capparelli 2"))
          {
            setText(String(c+1) + ": " + ssid, 0, (c+3)*20, 1, TFT_OLIVE);
            c++;
          }
        }

      }

case 1: //WIFI SCREEN
   if( (x<=20 && x>=0) && (y<=20 && y>=0) ){
       defaultScreen();
       caso = 0;
   }
   for (int i = 0; i<nNet; i++){
       if( (x<=310 && x>=230) && (y<=((i+3)*18)+13 && y>=(i+3)*18) ){

        if(WiFi.status() == WL_CONNECTED) WiFi.disconnect();
        char ssid[ssidNet[i].length()+1];
        ssidNet[i].toCharArray(ssid, ssidNet[i].length()+1);
        WiFi.begin(ssid, PASSWORD);
        while (WiFi.status() != WL_CONNECTED){
          delay(500);
          connectingScreen(String(ssid), String(PASSWORD));
        }
        connectedScreen(String(ssid), String(PASSWORD), WiFi.localIP().toString(), WiFi.gatewayIP().toString(), WiFi.subnetMask().toString());
        caso = 2;
     }
   }
   break;
```

- ***Comunicazione Mqtt***



PubSubClient è la libreria utilizzata per implementare negli esp32 i client. Di seguito mostreremo i codici utilizzati nella comunicazione mqtt e spiegheremo nel dettaglio la strategia utilizzata per implementare il gioco.

```
const char mqttBROKER[] = "192.168.1.146"; //ip of my PC
const char mqttCLIENT[] = "player1";

//for mqtt
WiFiClient espClient;
PubSubClient client(espClient);
```

```
//for MQTT
void callback(char topic[], byte payload[], unsigned int length) {
  msgIn = "";
  for (int i = 0; i < length; i++)
    msgIn = msgIn + ((char)payload[i]);
  subscribeMessage(topic);
 }


void publishMessage(char topic[], char msg[]){ client.publish(topic, msg); }

void reconnectMqtt(char topic[]) {
  while (!client.connected()) {
    if (client.connect(mqttCLIENT, "","","", 1, false,"")) { //QoS = 1 retain = false
        client.subscribe(topic, 1);  //QoS = 1
    }else
      delay(5000);
  }
}

void subscribeMessage(char topic[]){ client.subscribe(topic, 1); }
```

- Il metodo callback() è utilizzato per raccogliere char per char del messaggio ricevuto dalla sottoscrizione a un topic.
- Il metodo publicMessage() è utilizzato per pubblicare un messaggio sotto il dato topic.

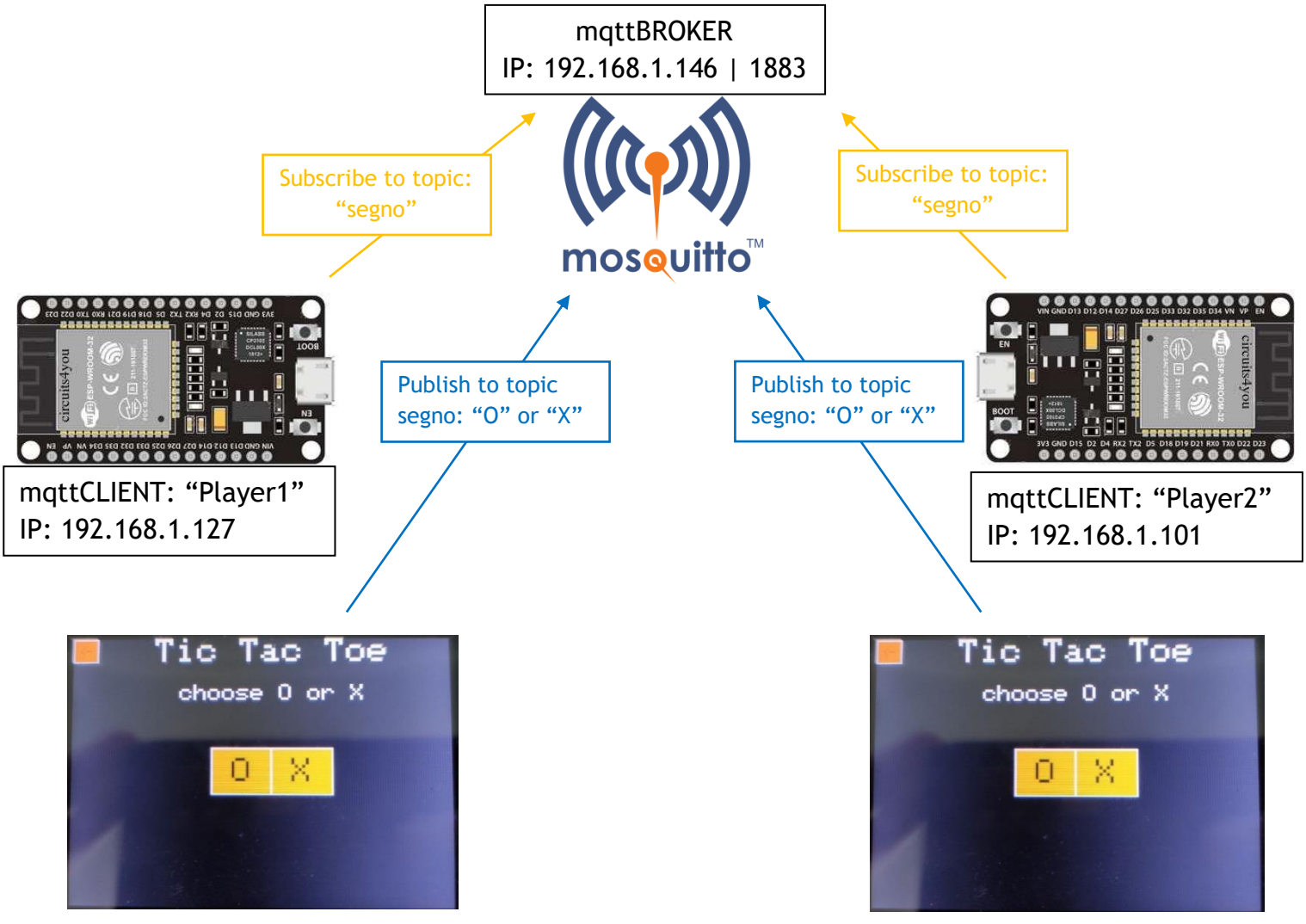
- Il metodo subscribeMessage() è utilizzato per sottoscriversi ad un topic, con QoS pari a 1, che significa che At Least Once Delivery e retain pari a false, cioè il messaggio non è archiviato per i client dormienti, d'altronde il progetto ha come scopo la comunicazione sempre attiva dei due player.
- Il metodo reconnectMqtt() è utilizzato per riconnettere il client e sottoscriversi qual ora esso non è più connesso al broker.

STRATEGIA

La strategia di comunicativa è suddivisa in due step:

1. Scambio di messaggi per la scelta del segno.
2. Scambio di messaggi per le mosse del gioco.

1 → di seguito mostriamo un'immagine riepilogativa di come funziona lo scambio di messaggi per la scelta del segno.



Entrambi i client si sottoscrivono al topic segno, ed entrambi hanno la possibilità di scegliere, ovviamente chi sceglie prima causa la scelta automatica ed opposta in segno dell'altro client. Il primo che sceglie pubblica sotto il topic segno e l'altro, sottoscritto al topic, riceverà la scelta avversaria e setterà il proprio segno all'opposto. Chi sceglie il segno per primo sarà il primo ad iniziare il gioco, settando turn = true.

```
case 1: //MQTT
  if( (x<=20 && x>=0) && (y<=20 && y>=0) ){
    protocolScreen();
    caso = 3;
    segno = '-';
    client.disconnect();
    }
  if (!client.connected()) reconnectMqtt("segno");
  client.loop();

  if(segno == '-' && msgIn == "O") {
    segno = 'X';
    msgIn = "";
    caso = 8;
    turn = false; //opponent turn
    gameScreen();
    client.unsubscribe("segno");
    client.disconnect();
  }
  if(segno == '-' && msgIn == "X") {
    segno = 'O';
    msgIn = "";
    caso = 8;
    turn = false; //opponent turn
    gameScreen();
    client.unsubscribe("segno");
    client.disconnect();
  }
  if(segno == '-' && (x<=170 && x>=120) && (y<=140 && y>=100)){
    publishMessage("segno", "O");
    msgIn="";
    segno = 'O';
    caso = 8;
    turn = true; //my turn
    gameScreen();
    client.unsubscribe("segno");
    client.disconnect();
  }
  if(segno == '-' && (x<=220 && x>=170) && (y<=140 && y>=100)){
    publishMessage("segno", "X");
    msgIn="";
    segno = 'X';
    caso = 8;
    turn = true; //my turn
    gameScreen();
    client.unsubscribe("segno");
    client.disconnect();
  }
  break;
```

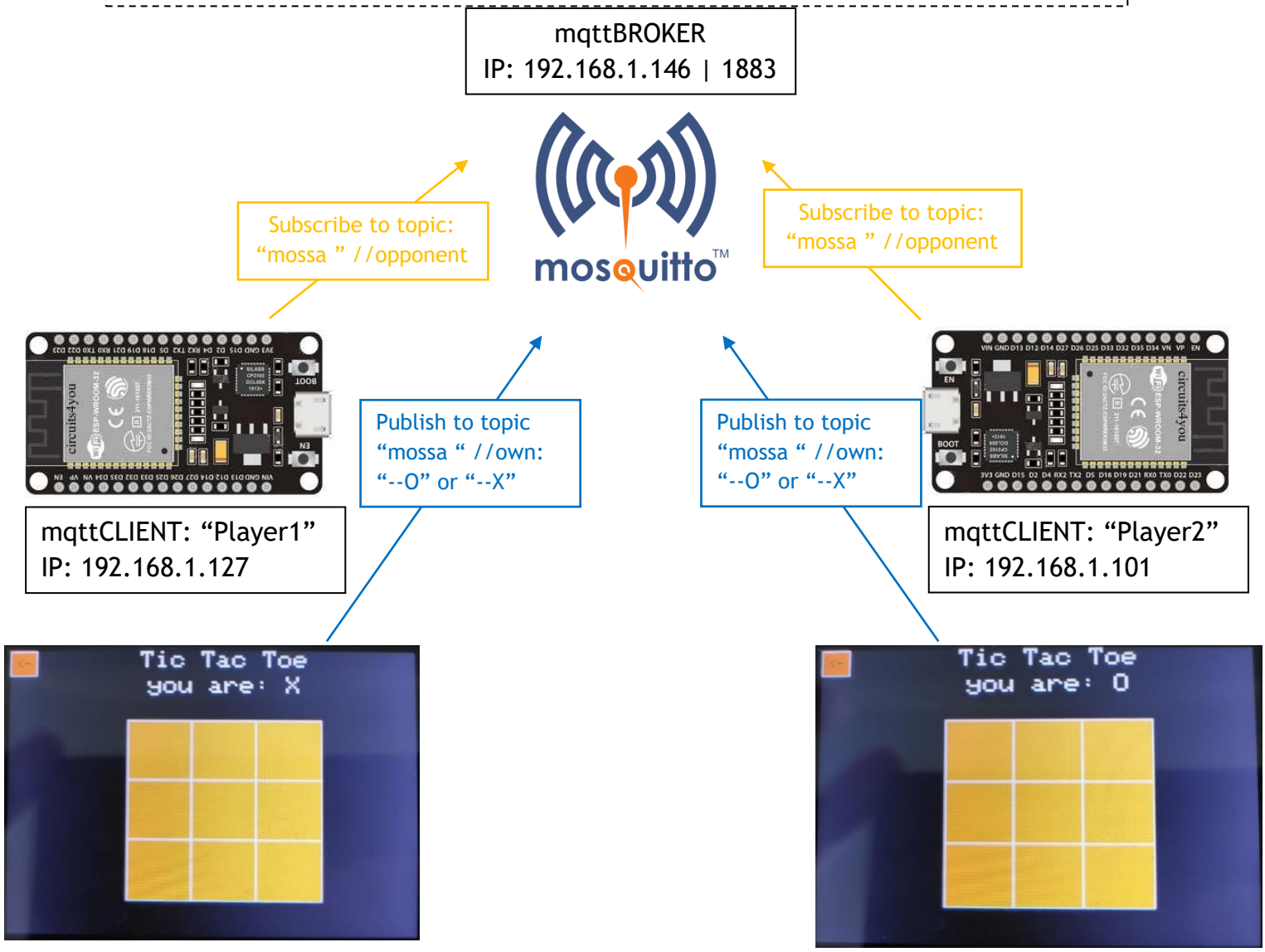
Mi connetto al broker e sottoscrivo il topic.

If utilizzati se ricevo la scelta avversaria

If utilizzati se scelgo io e di conseguenza pubblico il mio segno

2 → ora è mostrato come funziona la comunicazione MQTT per le mosse. Il meccanismo è molto semplice, ho due topic rispettivamente uno per pubblicare le mie mosse e un altro per sottoscrivermi alle mosse avversarie. I topic sono così composti:

```
case 1: //MQTT_____________________________________________________________

  char tpc1[] = "mossa ";  //Own Topic
  tpc1[5] = segno;

  char tpc2[] = "mossa ";  //Opponent Topic
  tpc2[5] = (segno=='O')?'X':'O';

  if (!client.connected()) reconnectMqtt(tpc2); //mi sottoscrivo al topic avversario
  client.loop();
```



Il messaggio scambiato e pubblicato è del formato “ - -SEGNO” cioè “11X” oppure “01O”, ovvero i primi due caratteri indicano la posizione della matrice su cui posizionare la propria mossa, il secondo è il segno del player. Il formato del topic è il seguente “mossaX” e “mossaO”.

```
 opponentMoves(false);

 if(turn)
   for(int i=0; i<3; ++i)
     for(int j=0; j<3; ++j)
       if( ((matrix[i][j] != 'X') && (matrix[i][j] != 'O')) && ( x<=((j+1)*50)+95 && x>=(j*50)+95 && y<=((i+1)*50)+60 && y>=(i*50)+60 ) ){
           String msgS = String(i) + String(j) + String(segno);
           char msg[msgS.length()+1];
           msgS.toCharArray(msg, msgS.length()+1);
           publishMessage(tpc1, msg);
           matrix[i][j] = segno;
           setBtn(String(segno), (j*50)+95, (i*50)+60, 50, 50, 3, TFT_ORANGE, TFT_BLACK, TFT_WHITE);
           turn = false;
           if(win()) {
              winScreen("You Won");
              caso = 9;
           }
           if(parity()){
              winScreen("No one won");
              caso = 9;
           }
       }

 break;

void opponentMoves(boolean isCoap){
   //OPPONENT MOVES
   uint8_t r,c;  //row column sign, y=row , x=column
   char s;
   if(!turn)
     if(isCoap) int getid = coap.get(IPAddress(192, 168, 1, 101), 5680, "mossa", COAP_CON);

     if(msgIn.length() == 3){
       r = uint8_t(msgIn[0])-48;
       c = uint8_t(msgIn[1])-48;
       s = char(msgIn[2]);

      if(matrix[r][c] != 'X' && (matrix[r][c] != 'O') && (matrix[r][c] == '-')){
          matrix[r][c] = s;
          setBtn(String(s), (c*50)+95, (r*50)+60, 50, 50, 3, TFT_ORANGE, TFT_BLACK, TFT_WHITE);
          turn = true;
          if(win()) {
             winScreen("You Lost");
             caso = 9;
          }
          if(parity()){
             winScreen("No one won");
             caso = 9;
          }
       }
       msgIn="";
       mossaOut="";
   }
}
```

*Wi-Fi

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

ip.addr == 192.168.1.127

No.	Time	Source	Destination	Protocol	Length	Info
366	57.679048	192.168.1.146	192.168.1.127	TCP	58	1883 → 53665 [SYN, ACK] Seq=0 Ack=1 Win=64620 Len=0 MSS=1460
367	57.683911	192.168.1.127	192.168.1.146	TCP	60	53665 → 1883 [ACK] Seq=1 Ack=1 Win=5744 Len=0
368	57.686081	192.168.1.127	192.168.1.146	MQTT	75	Connect Command
369	57.686205	192.168.1.146	192.168.1.127	MQTT	58	Connect Ack
370	57.691033	192.168.1.127	192.168.1.146	MQTT	66	Subscribe Request (id=2) [segno]
371	57.691172	192.168.1.146	192.168.1.127	MQTT	59	Subscribe Ack (id=2)
372	57.943901	192.168.1.127	192.168.1.146	TCP	60	53665 → 1883 [ACK] Seq=34 Ack=10 Win=5735 Len=0
373	58.152155	192.168.1.127	192.168.1.146	MQTT	64	Publish Message [segno]
374	58.152389	192.168.1.146	192.168.1.127	MQTT	64	Publish Message [segno]
375	58.169919	192.168.1.127	192.168.1.146	MQTT	65	Unsubscribe Request (id=3)
376	58.170034	192.168.1.146	192.168.1.127	MQTT	58	Unsubscribe Ack (id=3)
377	58.171107	192.168.1.127	192.168.1.146	MQTT	60	Disconnect Req
378	58.171143	192.168.1.146	192.168.1.127	TCP	54	1883 → 53665 [ACK] Seq=24 Ack=58 Win=64564 Len=0
379	58.171241	192.168.1.146	192.168.1.127	TCP	54	1883 → 53665 [FIN, ACK] Seq=24 Ack=58 Win=64564 Len=0
380	58.172516	192.168.1.127	192.168.1.146	TCP	60	52031 → 1883 [SYN] Seq=0 Win=5744 Len=0 MSS=1436
381	58.172567	192.168.1.146	192.168.1.127	TCP	58	1883 → 52031 [SYN, ACK] Seq=0 Ack=1 Win=64620 Len=0 MSS=1460
382	58.175161	192.168.1.127	192.168.1.146	TCP	60	53665 → 1883 [RST, ACK] Seq=58 Ack=24 Win=5744 Len=0
383	58.175957	192.168.1.127	192.168.1.146	TCP	60	53665 → 1883 [RST, ACK] Seq=58 Ack=24 Win=5744 Len=0
384	58.176391	192.168.1.127	192.168.1.146	TCP	60	53665 → 1883 [RST, ACK] Seq=58 Ack=25 Win=5744 Len=0
385	58.177990	192.168.1.127	192.168.1.146	TCP	60	52031 → 1883 [ACK] Seq=1 Ack=1 Win=5744 Len=0
386	58.178678	192.168.1.127	192.168.1.146	MQTT	75	Connect Command
387	58.178800	192.168.1.146	192.168.1.127	MQTT	58	Connect Ack
388	58.185881	192.168.1.127	192.168.1.146	MQTT	67	Subscribe Request (id=2) [mossaX]
389	58.186003	192.168.1.146	192.168.1.127	MQTT	59	Subscribe Ack (id=2)
391	58.395450	192.168.1.127	192.168.1.146	TCP	60	52031 → 1883 [ACK] Seq=35 Ack=10 Win=5735 Len=0

```
> Frame 4846: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{3DC4E9
> Ethernet II, Src: IntelCor_af:68:ce (e4:42:a6:af:68:ce), Dst: Espressi_fd:64:00 (24:62:ab:fd:64:00)
> Internet Protocol Version 4, Src: 192.168.1.146, Dst: 192.168.1.101
> Transmission Control Protocol, Src Port: 1883, Dst Port: 51385, Seq: 28, Ack: 70, Len: 13
> MQ Telemetry Transport Protocol, Publish Message
```

```
0000  24 62 ab fd 64 00 e4 42  a6 af 68 ce 08 00 45 00   $b··d··B ··h···E·
0010  00 35 76 95 40 00 80 06  ff e5 c0 a8 01 92 c0 a8   ·5v·@··· ········
0020  01 65 07 5b c8 b9 60 90  99 89 00 00 19 b4 50 18   ·e·[··`· ······P·
0030  fc 27 4f f0 00 00 30 0b  00 06 6d 6f 73 73 61 58   ·'O···0· ··mossaX
0040  31 30 58                                          10X
```

```
> Frame 4781: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_
> Ethernet II, Src: Espressi_fd:64:00 (24:62:ab:fd:64:00), Dst: IntelCor_af:68:ce (e4:42:a6:af:68
> Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.146
> Transmission Control Protocol, Src Port: 1883, Dst Port: 51385, Seq: 35, Ack: 10, Len: 13
> MQ Telemetry Transport Protocol, Publish Message
```

```
0000  e4 42 a6 af 68 ce 24 62  ab fd 64 00 08 00 45 00   ·B··h·$b ··d···E·
0010  00 35 00 11 00 00 ff 06  37 6a c0 a8 01 65 c0 a8   ·5····· 7j···e··
0020  01 92 c8 b9 07 5b 00 00  19 91 60 90 99 77 50 18   ·····[·· ··`··wP·
0030  16 67 3e ee 00 00 30 0b  00 06 6d 6f 73 73 61 4f   ·g>···0· ··mossaO
0040  31 31 4f                                          110
```

```
> Frame 3369: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface \Device\NPF_{3DC4E9
> Ethernet II, Src: Espressi_fd:64:00 (24:62:ab:fd:64:00), Dst: IntelCor_af:68:ce (e4:42:a6:af:68:ce)
> Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.146
> Transmission Control Protocol, Src Port: 51005, Dst Port: 1883, Seq: 34, Ack: 10, Len: 10
> MQ Telemetry Transport Protocol, Publish Message
```

```
0000  e4 42 a6 af 68 ce 24 62  ab fd 64 00 08 00 45 00   ·B··h·$b ··d···E·
0010  00 32 00 06 00 00 ff 06  37 78 c0 a8 01 65 c0 a8   ·2····· 7x···e··
0020  01 92 c7 3d 07 5b 00 00  19 8f 42 5c 0f 7b 50 18   ···=·[·· ··B\·{P·
0030  16 67 60 db 00 00 30 08  00 05 73 65 67 6e 6f 58   ·g`···0· ··segnoX
```

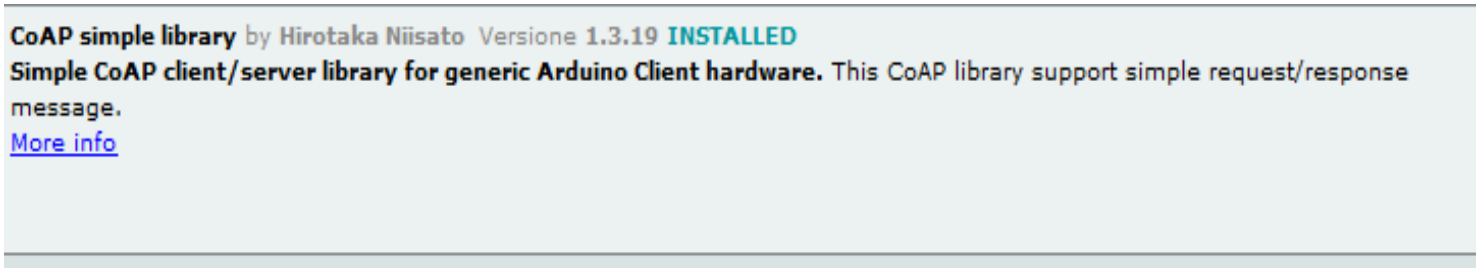
Precedentemente è stata mostrato qualche screen di cattura su WireShark, e abbiamo analizzato in particolare la pubblicazione di un messaggio e in giallo è evidenziato il payload. Possiamo anche notare la connessione al broker TCP attraverso l'handshaking a tre vie e anche la disconnessione avvenuta la scelta del segno.
Di seguito mostriamo anche, attraverso l'uso di Mosquitto e la sottoscrizione ai topi "segno" "mossaX" "mossaO" i messaggi pubblicati e sottoscritti.

```
PS C:\programmi\mosquitto> ./mosquitto_sub -h 192.168.1.146 -p 1883 -t mossaO
120
010
200
```

```
PS C:\programmi\mosquitto> ./mosquitto_sub -h 192.168.1.146 -p 1883 -t mossaX
21X
22X
```

```
PS C:\programmi\mosquitto> ./mosquitto_sub -h 192.168.1.146 -p 1883 -t segno
X
```

- ***Comunicazione CoAP***



CoAP simple library: è stata la libreria utilizzata per implementare la comunicazione CoAP dei due dispositivi, questa libreria è costruita su una comunicazione UDP per questo, da come vedremo da codice, bisogna importare la libreria WiFiUDP e inizializzare un client UDP.

```
//for coap
WiFiUDP udp;
Coap coap(udp);
```

```
//FOR CoAP
//when server receives the request for sign from player 2
void callback_segno(CoapPacket &packet, IPAddress ip, int port) {
  char p[packet.payloadlen + 1];
  memcpy(p, packet.payload, packet.payloadlen);
  p[packet.payloadlen] = NULL;

  String message(p);
  if(message.equals("X")) segno = 'X';

  else if(message.equals("O")) segno = 'O';

  else{
    if(segno == 'X') coap.sendResponse(ip, port, packet.messageid, "X");
    else if(segno == 'O') coap.sendResponse(ip, port, packet.messageid, "O");
    else coap.sendResponse(ip, port, packet.messageid, "-");
  }
}

void callback_mossa(CoapPacket &packet, IPAddress ip, int port) {
  char p[packet.payloadlen + 1];
  memcpy(p, packet.payload, packet.payloadlen);
  p[packet.payloadlen] = NULL;

  String message(p);

  if(String(message[2]).equals(String(segno))){
      mossaOut = message;

      if(win()) {
         winScreen("You Won");
         caso = 9;
         }
      if(parity()){
         winScreen("No one won");
         caso = 9;
      }
      turn = false;

  }else {
    char msg[mossaOut.length()+1];
    mossaOut.toCharArray(msg, mossaOut.length()+1);
    coap.sendResponse(ip, port, packet.messageid, msg);
  }
}
```

```
//when client receives the response
void callback_response(CoapPacket &packet, IPAddress ip, int port){
  char p[packet.payloadlen + 1];
  memcpy(p, packet.payload, packet.payloadlen);
  p[packet.payloadlen] = NULL;

  msgIn = String(p);
  //Serial.println(" 1: " + msgIn);
}
```

- Il metodo callback_segno() è utilizzato per settare gli eventi da fare quando avviene una get() o put() per la scelta del segno. Se message non è vuoto, quindi sto facendo una put(), setto il segno, se faccio una get() quindi il payload è vuoto restituisco il segno corrente.
- Il metodo callback_mossa() è utilizzato similmente al segno, se faccio una get() chiedo l'ultima mossa effettuata e il payload è vuoto, se faccio una put() inserisco la mia mossa nel payload e controllo se ho vinto.
- Il metodo callback_response() è utilizzato per ricevere la risposta del server da parte del client e salvare il payload nella variabile volatile msgIn.

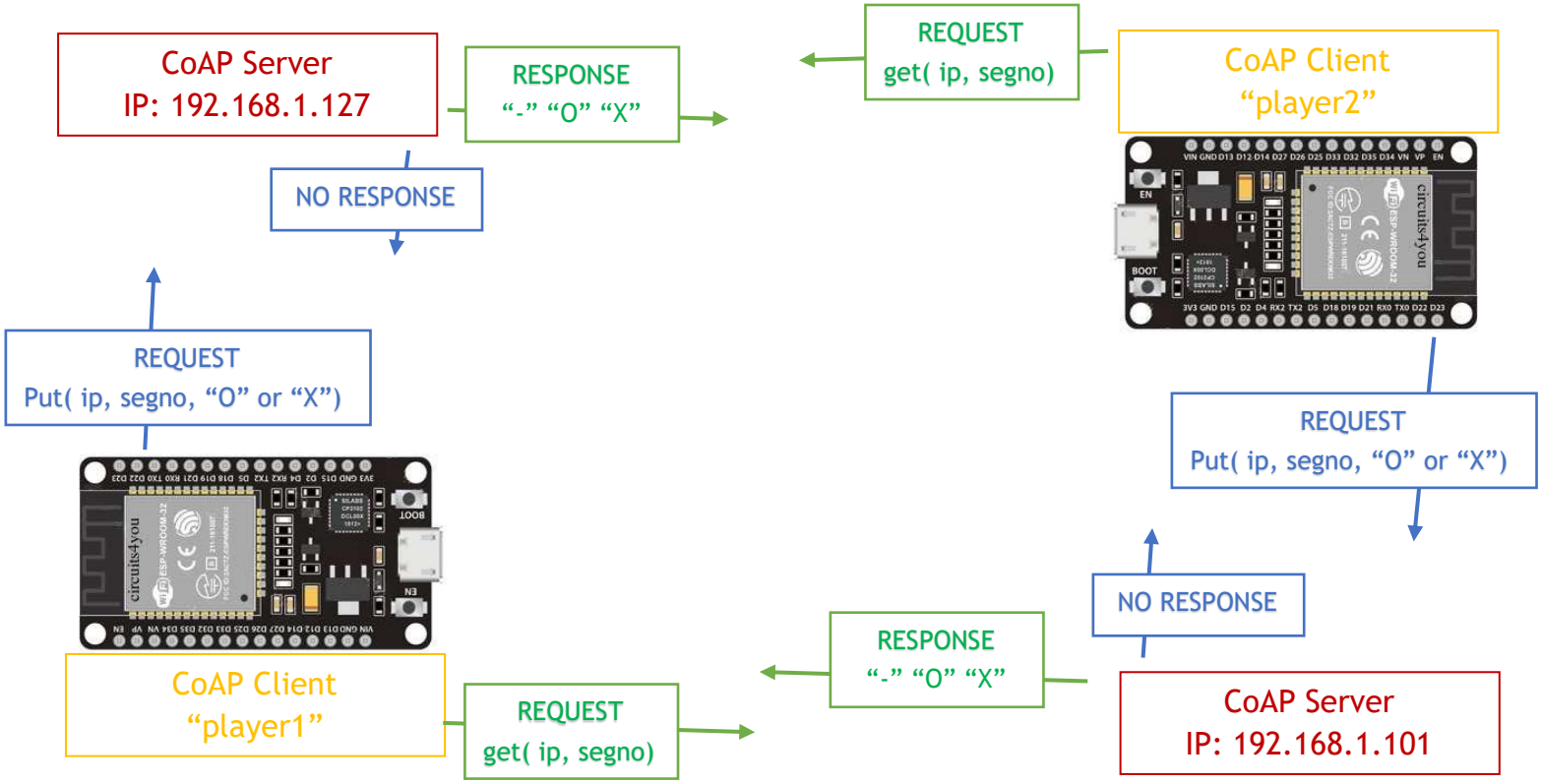
STRATEGIA

La strategia è molto semplice e similare per entrambi i metodo, ovvero per la scelta del segno e per l'invio della mossa. CoAP è basata su una architettura Client-Server, senza broker centrale a differenza di MQTT, quindi entrambi i dispositivi sono in comunicazione diretta e entrambi dovranno implementare interfacce server e client. La strategia è la seguente:

- Il client del dispositivo (sia Player1 che Player2) se decide di settare il segno o la mossa farà una put() sul proprio server, il client avversario è in continuo ascolto degli eventi sul server opposto, per cui periodicamente invierà get() al server avversario per capire se il segno è stato settato o se una mossa è avvenuta. In figura di seguito questo meccanismo è illustrato meglio.
- Le get() tentano di emulare una sorta di "Observe", che purtroppo questa libreria non mette a disposizione.

```
case 5: //CoAP SCREEN
 if( (x<=20 && x>=0) && (y<=20 && y>=0) ){
     protocolScreen();
     caso = 3;
 }
  if( (x<=230 && x>=90) && (y<=140 && y>=100) ){
     coap.server(callback_segno, "segno");
     coap.server(callback_mossa, "mossa");
     coap.response(callback_response);
     coap.start(5680); //port for UDP connection
```

SEGNO

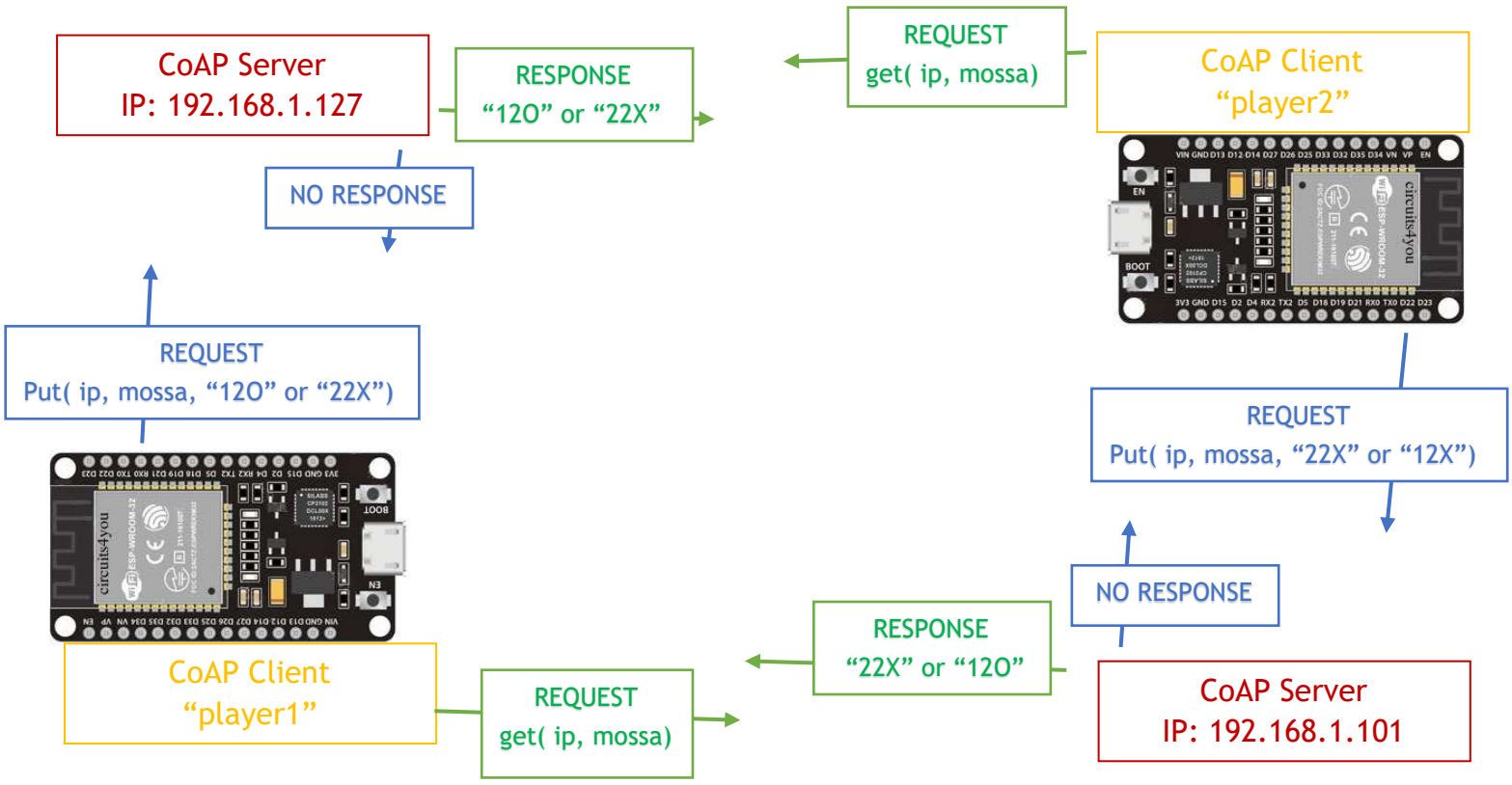


Quando setto le callback per i server, il metodo *coap.server(callback, "value")* mi genera un URI, ovvero mi crea una risorsa all'interno del server che ha come identificativo univoco la stringa "value". I due dispositivi implementano due risorse identificate dagli URI "mossa" "segno".

```
case 2: //CoAP_____________________________________________________________________
  if( (x<=20 && x>=0) && (y<=20 && y>=0) ){
    protocolScreen();
    caso = 3;
    segno = '-';
    msgIn="";
    mossaOut="";
    coap.end();

  }
  //se non scelgo io, chiedo al server coap avversario il  segno, cioè se l'avversario lo ha settato io ho l'opposto.
  if( (x>320 || x<0) && (y>240 || y<0) ){
    int getid = coap.get(IPAddress(192, 168, 1, 101), 5680, "segno", COAP_CON); //ip = 192.168.1.127 per player2
    if(segno == '-' && msgIn == "O") {
      segno = 'X';
      caso = 8;
      turn = false; //opponent turn
      gameScreen();
      msgIn="";
    }
    if(segno == '-' && msgIn == "X") {
      segno = 'O';
      caso = 8;
      turn = false; //opponent turn
      gameScreen();
      msgIn="";
    }
  }
```

```
  //se scelgo io, carico nel mio server coap il mio segno.
  if(segno == '-' && (x<=170 && x>=120) && (y<=140 && y>=100)){
    int putid = coap.put(IPAddress(192, 168, 1, 127), 5680, "segno", "O", COAP_CON);//ip = 192.168.1.101 per player2
    caso = 8;
    segno = 'O';
    turn = true; //my turn
    gameScreen();
}
 if(segno == '-' && (x<=220 && x>=170) && (y<=140 && y>=100)){
    int putid = coap.put(IPAddress(192, 168, 1, 127), 5680, "segno", "X", COAP_CON);//ip = 192.168.1.101 per player2
    caso = 8;
    segno = 'X';
    turn = true; //my turn
    gameScreen();
}
coap.loop();
break;
```

MOSSA




```
case 2: //CoAP

  opponentMoves(true);

  if(turn)
    for(int i=0; i<3; ++i)
      for(int j=0; j<3; ++j)
        if( ((matrix[i][j] != 'X') && (matrix[i][j] != 'O')) && ( x<=((j+1)*50)+95 && x>=(j*50)+95 && y<=((i+1)*50)+60 && y>=(i*50)+60 )
          String msgS = String(i) + String(j) + String(segno);
          char msg[msgS.length()+1];
          msgS.toCharArray(msg, msgS.length()+1);

          int putid = coap.put(IPAddress(192, 168, 1, 127), 5680, "mossa", msg, COAP_CON);

          setBtn(String(segno), (j*50)+95, (i*50)+60, 50, 50, 3, TFT_ORANGE, TFT_BLACK, TFT_WHITE);
          matrix[i][j] = segno;
      }

  coap.loop();
  break;
```

```
void opponentMoves(boolean isCoap){
   //OPPONENT MOVES
   uint8_t r,c;  //row column sign, y=row , x=column
   char s;
   if(!turn)
     if(isCoap) int getid = coap.get(IPAddress(192, 168, 1, 101), 5680, "mossa", COAP_CON);

     if(msgIn.length() == 3){
       r = uint8_t(msgIn[0])-48;
       c = uint8_t(msgIn[1])-48;
       s = char(msgIn[2]);

      if(matrix[r][c] != 'X' && (matrix[r][c] != 'O') && (matrix[r][c] == '-')){
          matrix[r][c] = s;
          setBtn(String(s), (c*50)+95, (r*50)+60, 50, 50, 3, TFT_ORANGE, TFT_BLACK, TFT_WHITE);
          turn = true;
          if(win()) {
             winScreen("You Lost");
             caso = 9;
          }
          if(parity()){
             winScreen("No one won");
             caso = 9;
          }
       }
       msgIn="";
       mossaOut="";
   }
}
```

- ***Metodi utili al gioco***

Per i fini del gioco viene utilizzata una matrice 3x3 di char, inizializzata con '-' dove memorizzeremmo rispettivamente per player1 e player2 la propria mossa con rispettivo segno 'O' 'X', ad ogni mossa ricevuta o fornita i player controllano se c'è una vincita o se si è terminato in parità. Di seguito sono forniti i metodi principali utilizzati per controllare, dal punto di vista del codice, l'andamento del gioco.

```
volatile char segno = '-';
volatile char matrix[3][3];
```

```
void emptyMatrix(){
  for(int i = 0; i<3; ++i)
    for(int j = 0; j<3; ++j)
      matrix[i][j] = '-';
}

boolean win(){

  char c = '-';
  for(int i = 0; i<3; ++i)
      if(String(matrix[i][0]).equals(String(matrix[i][1])) && String(matrix[i][0]).equals(String(matrix[i][2]))) {
        c = matrix[i][2];
        return (c=='O' || c=='X')?true:false;
      }

  for(int j = 0; j<3; ++j)
      if(String(matrix[0][j]).equals(String(matrix[1][j])) && String(matrix[0][j]).equals(String(matrix[2][j]))){
        c = matrix[2][j];
        return (c=='O' || c=='X')?true:false;
      }

  if(String(matrix[0][0]).equals(String(matrix[1][1])) && String(matrix[0][0]).equals(String(matrix[2][2]))){
    c = matrix[2][2];
    return (c=='O' || c=='X')?true:false;
  }


  if(String(matrix[0][2]).equals(String(matrix[1][1])) && String(matrix[0][2]).equals(String(matrix[2][0]))){
    c = matrix[0][2];
    return (c=='O' || c=='X')?true:false;
  }

  return false;
}

boolean parity(){
 // scanMatrix();
   for(int i = 0; i<3; ++i)
      for(int j = 0; j<3; ++j)
         if(String(matrix[i][j]).equals("-")) return false;
   return true;
}
```