Report IoT

Francesco Franco

July 2023

1 Introduzione

Un dispositivo IoT può utilizzare diverse tecnologie per comunicare. Le tecnologie di rete più comuni utilizzate nei dispositivi IoT includono **Wi-Fi** e **Bluetooth**. I dispositivi IoT comunicano tra di loro per consentire lo scambio di informazioni, dati e comandi. Questa comunicazione tra i dispositivi è essenziale per consentire il funzionamento coordinato di un sistema IoT e per consentire il controllo, il monitoraggio e l'automazione delle diverse applicazioni. Questo report si propone di valutare le performance delle tecnologie di comunicazione Wi-Fi utilizzando i protocolli **MQTT** e **CoAP**, nonché le performance del Bluetooth, concentrandosi su tre metriche chiave: **latenza**, **throughput** e **packet loss**.

2 Tecnologie Wireless

Di seguito una breve descrizione delle tecnologie utilizzate per i nostri esperimenti.

2.1 WiFi

Il Wi-Fi è una tecnologia di connessione wireless che consente ai dispositivi di comunicare tra loro e di accedere a internet. L'utilizzo del WiFi nel mondo IoT garantisce un'**ampia copertura** e un'**alta velocità** di connessione. I principali svantaggi sono il **consumo energetico**, un problema per i sistemi alimentati a batteria e le **interferenze** che potrebbero influire sulla qualità e sulla stabilità della connessione.

2.2 Bluetooth

Il Bluetooth è una tecnologia wireless a corto raggio. Uno dei principali vantaggi del bluetooth nel mondo IoT è il **consumo energetico ridotto** consentendo ai dispositivi IoT a batteria di funzionare per periodi più lunghi. Tra i principali svantaggi troviamo la **copertura limitata**, infatti, il bluetooth ha una portata relativamente breve (solitamente intorno ai 10 metri), la **velocità di trasferimento dati limitata** che è inferiore al Wi-Fi e come per il Wi-Fi le **inteferenze**.

3 Protocolli

Per la valutazione delle performance della tecnologia Wi-Fi sono stati considerati due protocolli: MQTT e CoAP. Di seguito una breve presentazione di entrambi i protocolli.

3.1 MQTT

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero progettato per la comunicazione tra dispositivi IoT. È basato su un modello **publish/-subscribe**, in cui i dispositivi inviano (pubblicano) e ricevono (si sottoscrivono) messaggi su specifici argomenti (topics).
Gli attori principali in MQTT sono i seguenti:

- **Publisher**: È il dispositivo o l'applicazione che invia i messaggi su specifici argomenti nel broker MQTT;
- **Subscriber**: È il dispositivo o l'applicazione che riceve i messaggi una volta sottoscritto agli argomenti di interesse;
- **Broker MQTT**: È un intermediario che riceve i messaggi dai publisher e li inoltra ai subscriber appropriati. Gestisce la registrazione dei client, la gestione degli argomenti e la consegna dei messaggi ai client sottoscritti corrispondenti.

La struttura di un pacchetto MQTT è composta da tre parti principali:

- **Header**: L'header contiene informazioni di controllo, come il tipo di pacchetto (pubblicazione, sottoscrizione, conferma) e le opzioni di QoS (Quality of Service) per garantire la consegna dei messaggi. L'header specifica anche il flag di "Retained" che indica se il messaggio dovrebbe essere conservato dal broker e inviato a nuovi subscriber;
- **Payload**: Il payload contiene i dati effettivi del messaggio, come il valore di un sensore o un comando da eseguire. Il formato e il contenuto del payload sono specifici all'applicazione;
- **Topic**: Il topic rappresenta il nome dell'argomento a cui il messaggio è associato. Il topic viene utilizzato dal broker MQTT per instradare il messaggio ai subscriber corretti.

3.2 CoAP

CoAP (Constrained Application Protocol) è un protocollo applicativo progettato per le reti a basso consumo energetico e con risorse limitate. Si basa sul modello **client/server**, utilizza il protocollo di trasporto UDP (User Datagram Protocol) per la comunicazione e utilizza i metodi HTTP (GET, POST, PUT, DELETE) per consentire la comunicazione tra dispositivi.
Gli attori principali in CoAP sono:

- **Client**: È il dispositivo o l'applicazione che fa richieste a un server CoAP per ottenere informazioni o eseguire operazioni;
- **Server**: È il dispositivo o l'applicazione che fornisce le risorse richieste dai client. Riceve le richieste dai client, elabora le richieste e invia le risposte corrispondenti.

La struttura di un pacchetto CoAP è composta da quattro parti principali:

- **Header**: L'header contiene informazioni di controllo, come il tipo di pacchetto (richiesta o risposta), il codice di stato, l'ID del messaggio e le opzioni aggiuntive;
- **Token**: Il token è un campo opzionale che può essere utilizzato per correlare le richieste e le risposte;
- **Options**: Le opzioni trasportano informazioni aggiuntive associate alla richiesta o alla risposta;
- **Payload**: Il payload contiene i dati effettivi associati alla richiesta o alla risposta.

4 Implementazione

4.1 Dispositivi utilizzati

Per l'analisi delle prestazioni della rete WiFi è stato utilizzato un dispositivo ESP32 insieme a un computer che funge da MQTT Broker, oltre a un client Python che simula sia un MQTT Client che un CoAP Server. Per l'analisi delle prestazioni della rete Bluetooth sono stati adottati due dispositivi ESP32. Tale decisione è stata presa a causa delle difficoltà incontrate durante l'installazione della libreria *pybluez*, che avrebbe consentito l'utilizzo del computer come parte dell'analisi della rete Bluetooth.

4.2 Eclipse Mosquitto

Eclipse Mosquitto è un message broker open source che implementa varie versioni del protocollo MQTT. È leggero e può essere utilizzato su tutti i dispositivi. Mosquitto offre due comandi molto utili per l'interazione con il broker MQTT:

- mosquitto_pub: consente di pubblicare un messaggio su un determinato topic MQTT;
- mosquitto_sub: consente di sottoscriversi a uno o più topic MQTT e ricevere i messaggi pubblicati su quei topic.

Durante la fase di testing e lo sviluppo della dashboard, i comandi `mosquitto_pub` e `mosquitto_sub` si sono rivelati estremamente utili. Sono stati impiegati per tenere traccia dei messaggi inviati dai client MQTT e per inviare dati a Node-RED. Ciò ha facilitato il monitoraggio e il debug delle comunicazioni, nonché la generazione di dati di test per la creazione e l'ottimizzazione della dashboard.

4.3 Liberie ESP32

Le principali librerie utilizzate su ESP32 per valutare le performance di rete sono le seguenti:

- **WiFi.h**: fornisce una serie di funzioni e metodi che consentono di stabilire una connessione WiFi e interagire con reti wireless;
- **coap-simple.h**: fornisce una semplice implementazione del protocollo applicativo CoAP;

- **PubSubClient.h**: fornisce un'interfaccia semplice e intuitiva per implementare la comunicazione MQTT;
- **BluetoothSerial.h**: fornisce un'implementazione della comunicazione Bluetooth.

4.4 Librerie Python

Le principali librerie utilizzate per simulare il comportamento di un altro dispositivo sono:

- **aiocoap**: consente di creare client e server CoAP asincroni in Python. Fornisce una serie di funzionalità per la gestione delle richieste e delle risposte CoAP;
- **paho**: permette di creare un client MQTT in Python per pubblicare e sottoscriversi a topic MQTT. Consente di inviare e ricevere messaggi MQTT e gestire la connessione al broker MQTT.

4.5 Nodered

Node-RED è stato utilizzato per creare una visualizzazione grafica dei risultati delle misurazioni. I dati sono stati ottenuti tramite il protocollo MQTT. È stata creata una dashboard per visuliazzare i risultati delle misurazioni.

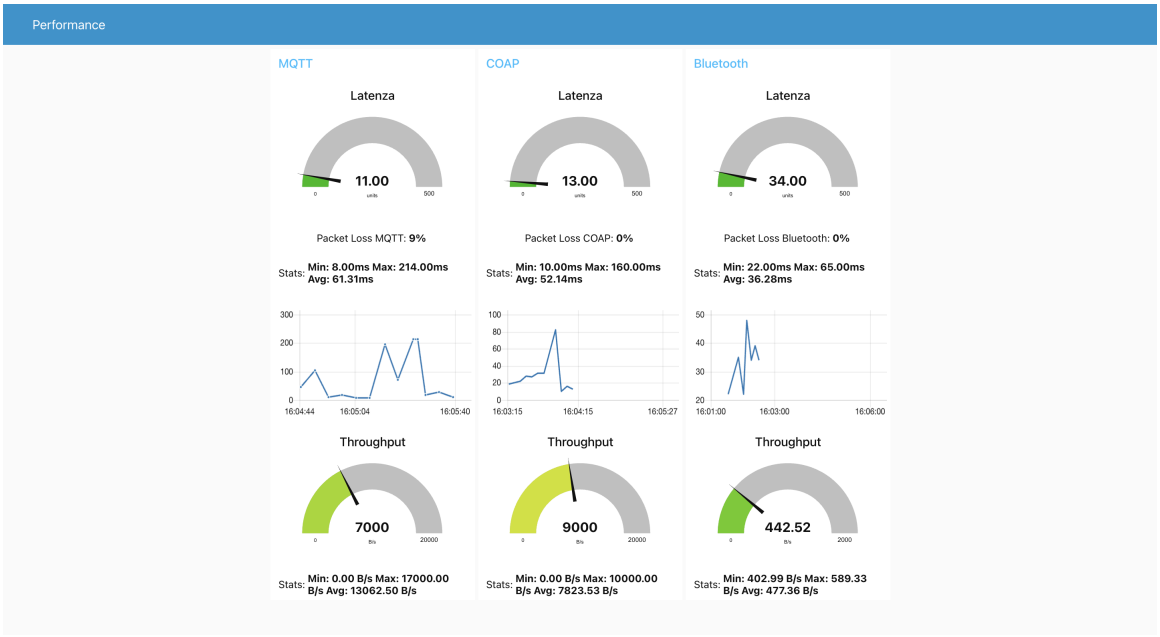

Figure 1: Dashboard Node-RED

5 Calcolo delle performance

Di seguito viene fornita una breve spiegazione delle metriche utilizzate per valutare le performance delle tecnologie di rete e degli snippet di codice. Le prestazioni sono valutate periodicamente per monitorare e aggiornare i dati visualizzati sulla dashboard.

5.1 Latenza

Per misurare la latenza è stato inviato un pacchetto contenente la stringa "*latency*". Il pacchetto viene inviato dall'ESP32 al client in esecuzione sul computer e quando il messaggio arriva a destinazione, invia una conferma di ricezione (ACK) al mittente. Il tempo impiegato per l'intero processo rappresenta il **RTT** (Round Trip Time) e dividendolo per due si ottiene un'approssimazione della latenza unidirezionale.

$$RTT = tMessaggioRicevuto - tMessaggioInviato$$

$$Latenza = RTT/2$$

Listing 1: Calcolo throughput

```
mqtt_client.loop();
if (millis() - start_time_latency > TIMEOUT)
{
    start_time_latency = millis();
    mqtt_client.publish(LATENCY_TOPIC, "latency", 1);
}
// mqtt.cpp
void latency_callback(char *topic, byte *payload, unsigned int length)
{
    unsigned long received_time = millis();
    float latency = (received_time - start_time_latency) / 2;
    Serial.print("Latenza: ");
    Serial.print(latency);
    Serial.println(" ms");
    mqtt_client.publish(NODERED_LATENCY, String(latency).c_str(), 1);
}
```

5.2 Throughput

Per calcolare il throughput sono stati inviati in modo consecutivo dei pacchetti contenenti la stringa "*throughput*" per un periodo di 3 secondi. Per ogni messaggio inviato è stato aggiornato un contatore in modo da tenere traccia dei byte inviati. Al termine del ciclo, il throughput è stato calcolato in base al numero totale di byte inviati durante il periodo monitorato.

$$throughput = byteInvati/(tEnd - tStart) * 1000.0$$

Listing 2: Calcolo throughput

```
unsigned long current_time = millis();
unsigned long elapsed_time = current_time - start_time_thr;
int packet_sent = 0;
if (elapsed_time > TIMEOUT)
{
    unsigned int t_start_loop = millis();
    while (millis() - t_start_loop < SECONDS)
    {
      mqtt_client.publish(THROUGHPUT, "throughput", 1);
      packet_sent += strlen("throughput");
```

```
    }
    float throughput = (float)packet_sent /
        (float)(millis() - t_start_loop) * 1000.0;
    mqtt_client.publish(NODERED_THROUGHPUT,
        String(throughput).c_str(), 1);
    Serial.print("Throughput MQTT: ");
    Serial.print(throughput);
    Serial.println(" B/s");

    start_time_thr = millis();
}
```

5.3 Packet Loss

Per calcolare il packet loss sono stati inviati in modo consecutivo pacchetti contenenti la stringa "*packetloss*" per un periodo di 3 secondi. Sono stati mantenuti due contatori in modo da tenere traccia dei pacchetti inviati e ricevuti. Il packet loss è stato calcolato confrontando il numero totale dei pacchetti inviati con il numero dei pacchetti effettivamente ricevuti.

$$packetLoss = (packetSent - packetReceived/packetSent) * 100$$

Listing 3: Packet Loss

```
if (elapsed_time_pl > TIMEOUT)
  {
    unsigned int t_start_loop = millis();
    while (millis() - t_start_loop < SECONDS)
    {
      mqtt_client.publish(PACKET_LOSS, "packetloss", 1);
      messages_sent_pl++;
      mqtt_client.loop();
    }

    packet_received = packet_received > messages_sent_pl ?
        messages_sent_pl : packet_received;
    float packet_loss = packet_received == 0 ? 100 :
    ((float)(messages_sent_pl - packet_received) /
        (float)messages_sent_pl) * 100;
    mqtt_client.publish(NODERED_PACKET_LOSS, String(packet_loss).c_str(), 1);

    messages_sent_pl = 0;
    packet_received = 0;
    start_time_pl = millis();
  }
// mqtt.cpp
void packet_loss_callback(char *topic, byte *payload, unsigned int length)
{
    packet_received += 1;
}
```

6 Risultati

Sono stati eseguiti degli esperimenti a distanze di 1 metro, 10 metri con ostacoli e senza.

6.1 WiFi

Dai vari esperimenti è emerso che l'utilizzo del protocollo CoAP offre risultati migliori in termini di **latenza** rispetto al protocollo MQTT quando si utilizza una connessione WiFi. Questa osservazione si applica sia quando i dispositivi IoT sono fisicamente vicini tra loro, sia quando sono posizionati a distanza l'uno dall'altro. Ciò può essere attribuito a diverse caratteristiche del protocollo CoAP che contribuiscono a una latenza inferiore rispetto a MQTT.

6.1.1 Distanza di 1 metro

Protocollo	Metriche	Minimo	Massimo	Media
MQTT	**Latenza (ms)**	8.00	163.00	58.00
	Throughput (B/s)	0	16000.00	12581.89
	Packet loss	-	-	0
COAP	**Latenza (ms)**	5.00	136.00	52.08
	Throughput	0	16000.00	10896.10
	Packet loss	-	-	0

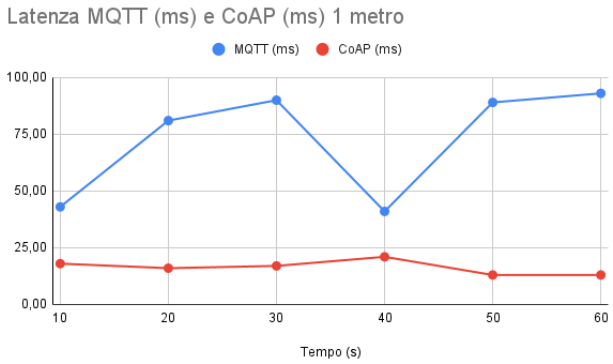

Figure 2: Distanza di 1 metro

6.1.2 Distanza di 10 metri senza ostacoli

Protocollo	Metriche	Minimo	Massimo	Media
MQTT	**Latenza (ms)**	14.00	244.00	76.00
	Throughput (B/s)	10457.00	14300.00	11498.68
	Packet loss	-	-	0
COAP	**Latenza (ms)**	6.00	199.00	58.08
	Throughput	8000.00	15430.00	6680.16
	Packet loss	-	-	0

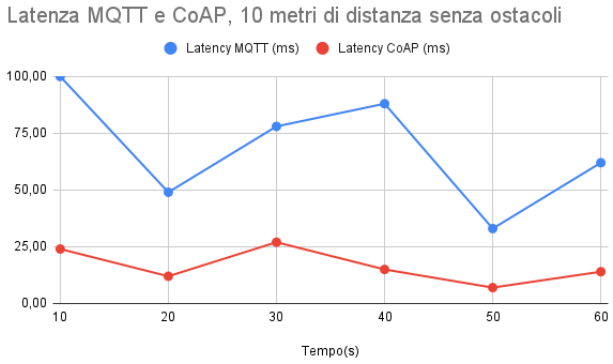

Figure 3: Distanza di 10 metri senza ostacoli

6.1.3 Distanza di 10 metri con ostacoli

Protocollo	Metriche	Minimo	Massimo	Media
MQTT	**Latenza (ms)**	31.00	3518.00	575.33
	Throughput (B/s)	6820.37	12706.67	11661.19
	Packet loss	-	-	0
COAP	**Latenza (ms)**	26.00	1601.00	361.82
	Throughput	2000.00	11000.00	5957.00
	Packet loss	-	-	0

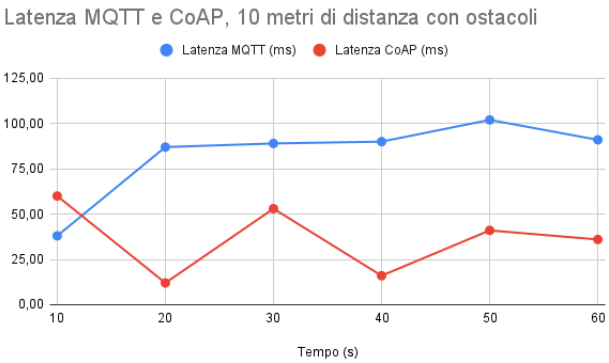

Figure 4: Distanza di 10 metri con ostacoli

6.2 Bluetooth

Dai vari esperimenti è emerso che le performance del Bluetooth sono piuttosto buone in termini di latenza sia a breve che "lunga" distanza. La latenza aumenta leggermente quando la distanza tra i dispositivi aumenta, questo può essere attribuito alla maggior dispersione del segnale, che richiede un maggiore tempo per il trasferimento dei dati. Come prevedibile il throughput raggiunge valori più alti quando i dispositivi sono vicini ma comunque nettamente inferiori rispetto al WiFi. Non è stato possibile fare misurazioni a distanza di 10 metri con ostacoli in quanto la connessione tra i due dispositivi saltava.

6.2.1 Distanza di 1 metro

Metriche	Minimo	Massimo	Media
Latenza (ms)	22.00	67.00	39.50
Throughput (B/s)	15.22	1990.68	1034.37
Packet loss	-	-	0

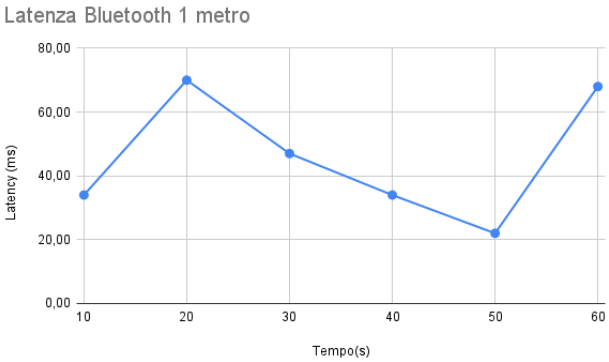

Figure 5: Distanza di 1 metro

6.2.2 Distanza di 10 metri senza ostacoli

Metriche	Minimo	Massimo	Media
Latenza (ms)	23.00	81.00	48.50
Throughput (B/s)	3.24	1549.98	875.37
Packet loss	-	-	0

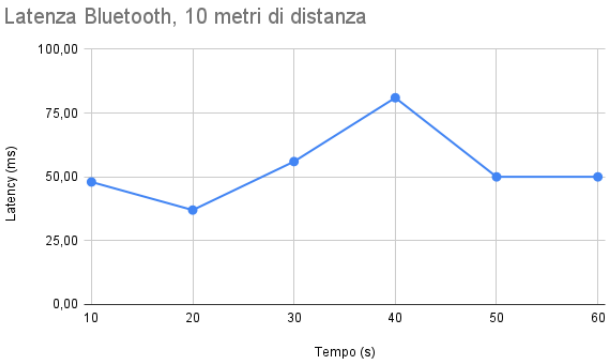

Figure 6: Distanza di 10 metri

7 Conclusioni

Le analisi condotte forniscono un'indicazione delle prestazioni delle tecnologie prese in considerazione, tuttavia è importante considerare che i risultati ottenuti potrebbero variare in contesti e condizioni diverse. Dai risultati ottenuti è possibile fare diverse considerazioni. Il WiFi si è dimostrato una ottima soluzione in termini di copertura e velocità, consentendo una trasmissione veloce dei dati tra i dispositivi IoT. Per quanto riguarda CoAP, è emerso come un protocollo di comunicazione con latenze estremamente basse, probabilmente grazie all'utilizzo del protocollo UDP. Questo lo rende una scelta interessante per applicazioni che richiedono tempi di risposta rapidi e comunicazioni in tempo reale. MQTT ha mostrato risultati soddisfacenti, sebbene con una latenza leggermente più alta rispetto a CoAP. Tuttavia, i tempi di latenza rimangono accettabili per molte applicazioni IoT e MQTT offre vantaggi come la scalabilità e la gestione affidabile dei messaggi. Il Bluetooth ha fornito risultati accettabili, ma la sua portata limitata può essere un vincolo significativo per molte applicazioni IoT.
In conclusione, la scelta della tecnologia di comunicazione dipende dalle esigenze specifiche dell'applicazione IoT. È importante considerare attentamente i requisiti di copertura, velocità, latenza e scalabilità per determinare la soluzione più adatta. La valutazione delle prestazioni in contesti e condizioni reali è essenziale per prendere decisioni informate sulla tecnologia da utilizzare nell'implementazione di un sistema IoT.