DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN: INGEGNERIA ELETTRONICA-TELECOMUNICAZIONI

STUDIO, IMPLEMENTAZIONE E TEST DI ARCHITETTURE INTERNET OF THINGS BASATE SU MQTT 5.0

Study, implementation and test of IoT architectures based on MQTT v5

Studente :
FIORDARANCIO DIEGO

Relatore :
PIERLEONI DOTT.SA PAOLA

ANNO ACCADEMICO 2022-2023

SOMMARIO

ABSTRACT

L'Internet of Things (IoT) descrive la rete di oggetti fisici, ossia le "cose", integranti sensori, attuatori o altri dispositivi in grado di comunicare e scambiare dati con altri dispositivi e sistemi su Internet. Le disponibilità computative di tali oggetti fisici potrebbero essere limitate, sia per ragioni di dimensione che per ragione di consumi. È quindi necessario introdurre dei protocolli che permettano la creazione di queste reti, tenendo conto dei limiti ai quali i dispositivi vanno in contro. Lo scopo di questo elaborato è di analizzare le prestazioni di uno di questi protocolli, l'MQTT (Message Queue Telemetry Transport), per applicazioni di tipo Request/Response, funzionalità solo recentemente integrata nel protocollo. Nella trattazione verrà inizialmente mostrato il metodo di funzionamento del protocollo, le sue caratteristiche particolari e le migliorie apportate nel corso del tempo. Verrà infine mostrata l'implementazione del protocollo in applicazioni di tipo Request/Response, e verranno effettuati dei test per studiarne i tempi di latenza e l'utilizzo di risorse computative, che verranno paragonate con le prestazioni di http (Hipertext Transfer Protocol), il protocollo oggi più utilizzato in internet per realizzare architetture basate sul modello Request/Response.

Capitolo 1
I PROTOCOLLI DELL'IOT

Ad oggi, i dispositivi elettronici sono presenti in ogni ambito della nostra vita, sia in quello domestico (basti pensare ad ogni tipo di elettrodomestico, alle autovetture, alla videosorveglianza…) che quello industriale. Data la grande diversità di situazioni in cui l'IoT viene impiegato, è importante trovare la soluzione implementativa adatta al campo di utilizzo, e per questa ragione sono stati creati diversi protocolli che ci permettano di utilizzare al meglio la tecnologia a nostra disposizione. I principali protocolli ad oggi utilizzati nella connessione degli "oggetti" sono il LwM2M e l'MQTT, protocolli che svolgono lo stesso compito ma in maniere differenti , ognuno con i propri pregi e i propri difetti.
Vediamoli ora in dettaglio:

1.1 **LwM2M (Lightweigth machine to machine) :**

Basato su CoAP, LwM2M è un protocollo a livello applicativo che si appoggia alla rete internet utilizzando vari protocolli, tra cui UDP, UDP, TCP, Non-IP Data delivery, SMS, CIoT & LoRaWAN. Implementa un'architettura di tipo Request/Response, con una connessione diretta tra client e server. Permette di gestire dispositivi da remoto, in particolare di ricevere letture da sensori, abilitare o disabilitare le letture, configurare il dispositivo o aggiornarlo e controllare la sicurezza della connessione.

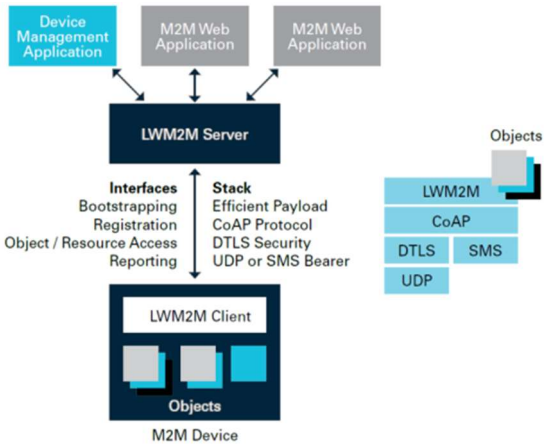

Figura 1-1 : Architettura LwM2M

Per raggiungere questo scopo presenta un set di oggetti, ad ognuno dei quali è affidata una funzionalità specifica, che sono definiti e strutturati dallo standard, con la possibilità di creare nuovi tipi di oggetti che si adattino alle necessità del dispositivo di utilizzo e allo scopo per il quale verrà utilizzato. La comunicazione avviene tramite lo scambio di questi oggetti, che sono formattati in maniera tale da poter trasportare le informazioni dei dispositivi e le impostazioni per la connessione. D'altro canto, l'implementazione di questo protocollo genera un gran numero di limiti imposti dal protocollo stesso riguardo il tipo di azioni che possono essere compiute, la gestione della connessione e il mantenimento dei dispositivi utilizzati.

1.2 **MQTT (Message Queue Telemetry Transport) :**

Lo standard de facto per l'IoT è oggi l'MQTT, protocollo a livello di sessione che utilizza lo stack TCP/IP per connettersi ad internet. Si basa su un'architettura di tipo Publish/Subscribe, nel quale gli utenti si interfacciano ad un broker sul quale possono pubblicare messaggi in determinati Topic e dal quale possono iscriversi a Topic per ricevere i messaggi pubblicatici.

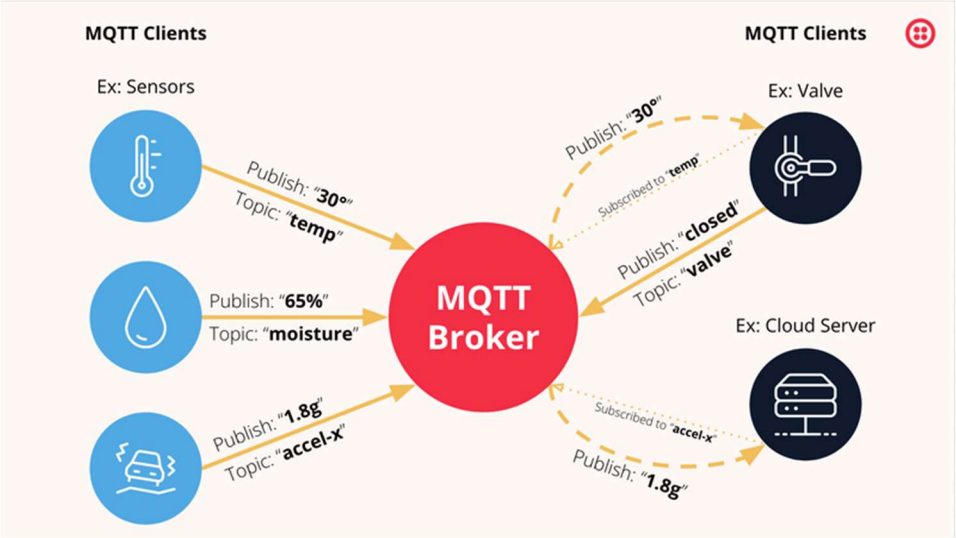

Figura 1-2 : Architettura MQTT

Questa caratteristica di separare fisicamente gli end point della comunicazione attraverso un broker gli permette di gestire le comunicazioni in maniera asincrona, facilitando la distribuzione dei messaggi a numerosi client. Nell'architettura publish/subscribe proposta da MQTT il broker svolge il ruolo di centro di controllo, instaurando le connessioni con i vari client che si collegano alla rete, immagazzinando tutti i messaggi pubblicati per poi ridistribuirli a chi ha fatto richiesta di usufruire di tale servizio. L'archiviazione e la ridistribuzione dei messaggi avvengono attraverso un sistema ad argomenti (Topic) : la pubblicazione di un messaggio da parte di un client avviene sempre in un topic specificato dal client stesso, l'immagazzinamento viene effettuato dal broker che crea un'istanza per ogni topic e la redistribuzione avviene da parte del broker verso i client che si sono iscritti al topic nel quale è stato pubblicato il messaggio. Inoltre, lo standard per MQTT non impone nessuna limitazione riguardo il payload dei messaggi, rendendo il protocollo funzionante con ogni tipo di struttura dati, anche proprietario.

Per queste ragioni MQTT è il protocollo maggiormente impiegato nelle applicazioni IoT e nelle reti di smart devices.

Gli ambiti di applicazione di questo protocollo vanno da quello domestico, per le smart home, a quello industriale, nella connessione di sensori per aziende.
Vi sono esempi di applicazioni di MQTT su piattaforme Amazon, nel quale dei messaggi possono essere utilizzati per controllare Amazon Alexa in ambienti domestici, tramutando gli input vocali in input testuali.

Vediamo ora in dettaglio le specifiche del protocollo, gli strumenti che utilizza e le tecniche che utilizza per mettere in comunicazioni gli end-point della connessione.

Capitolo 2
IL PROTOCOLLO MQTT

Ogni tipo di azione viene effettuata attraverso lo scambio di pacchetti di controllo, i quali contengono le informazioni necessarie a eseguire un determinato compito. Vi sono 14 tipi di pacchetti, ognuno dei quali è strutturato in maniera differente in base al compito affidatogli.

Reserved	0	Reserved for future use
CONNECT	1	Client request to connect to server
CONNACK	2	Connect acknowledgement
PUBLISH	3	Publish message
PUBACK	4	Publish message acknowledgement
PUBREC	5	Publish received (QoS=2)
PUBREL	6	Publish release (QoS=2)
PUBCOMP	7	Publish complete (QoS=2)
SUBSCRIBE	8	Client subscribe request
SUBACK	9	Subscribe acknowledgement
UNSUBSCRIBE	10	Client unsubscribe request
UNSUBACK	11	Unsubscribe acknowledgement
PINGREQ	12	Ping request
PINGRESP	13	Ping response
DISCONNECT	14	Client disconnection request
Reserved	15	Reserved for future use

figura 2.1 : Tipi di pacchetti di controllo

2.1 **Instaurazione connessione**

Per potersi scambiare messaggi, i vari client devono collegarsi allo stesso broker, il quale può essere locato sia sulla stessa macchina di uno dei client sia su una macchina esterna. Il collegamento avviene attraverso l'invio da parte del client di un pacchetto di tipo CONNECT, attraverso il quale dichiara il suo identificatore e presenta un user name ed una eventuale password. Il broker risponde con un pacchetto di tipo CONNACK per stabilire la sessione.

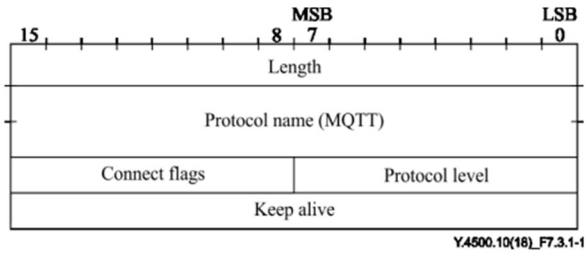

Figura 2-1.1 : Header CONNECT

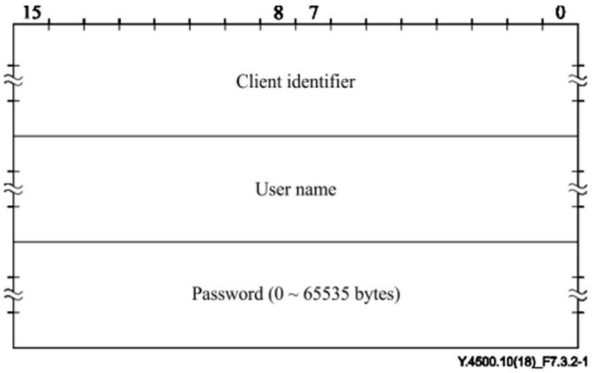

Figura 2-1.2 : Payload CONNECT

2.2 **Invio e ricezione messaggi**

2.2.1 *Invio messaggi*

L'invio di un messaggio da parte di un client avviene attraverso un pacchetto di tipo PUBLISH, il quale informa il broker riguardo che tipo di pacchetto è arrivato, il topic su cui deve essere pubblicato il messaggio e la qualità del servizio richiesta. Normalmente la QoS è settata a 1.

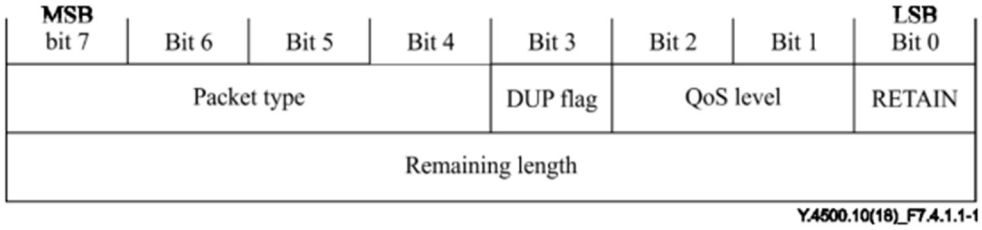

figura 2-2-1.1 : Header fisso PUBLISH

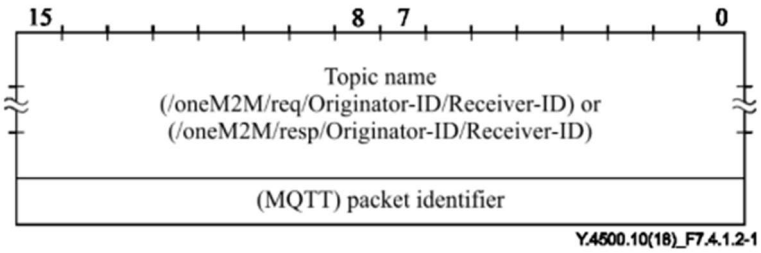

figura 2-2-1.2 : Header variabile PUBLISH

Il messaggio è contenuto nel payload, e non ha restrizioni in termini di codifiche o formattazione dati specifiche. L'utilizzo di tali è dipendente dall'applicazione di utilizzo del protocollo.

2.2.2 *Ricezione messaggi*

Per predisporsi alla ricezione dei messaggi, i client devono iscriversi ai topic di loro interesse, e per fare ciò utilizzano dei pacchetti di tipo SUBSCRIBE, i quali contengono nel loro payload il topic filter da utilizzare per iscriversi ai vari topic. Una volta ricevuto il SUBACK di conferma, i client riceveranno dal broker dei pacchetti PUBLISH contenenti i messaggi pubblicati sul topic. È possibile per un client anche di disiscriversi da un topic, nel caso la risorsa non sia più necessaria, attraverso un pacchetto di tipo UNSUBSCRIBE.

Description	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Topic Filter								
byte 1	Length MSB							
byte 2	Length LSB							
bytes 3..N	Topic Filter							
Subscription Options								
	Reserved		Retain Handling		RAP	NL	QoS	
byte N+1	0	0	X	X	X	X	X	X

figura 2-2-2.1 : Payload SUBSCRIBE

2.3 **Mantenimento connessione**

Nel caso si verifichino lunghi periodi di inattività, nei quali non vengono scambiati pacchetti di controllo, i client sono tenuti a mantenere attiva la connessione notificando la loro presenza al broker, responsabile di tenere attiva la sessione. Questo mantenimento avviene tramite lo scambio di pacchetti di tipo PINGREQ e PINGRESP : il client invia al broker un PINGREQ e attende di ricevere in risposta dal broker un PINGRESP. Non è imposto dal protocollo di seguire questo meccanismo, ma è altamente consigliato per non incorrere in problemi dovuti ad un'anomala disconnessione, scenario probabile considerato l'ambito di utilizzo di MQTT.

2.4 **Chiusura connessione**

La disconnessione tra client e broker avviene attraverso l'invio di un pacchetto di tipo DISCONNECT, che sancisce la fine della connessione e informa il broker che la sessione può essere chiusa. È buona norma usare questa procedura per la disconnessione in quanto la semplice terminazione delle attività del client lascerebbe attive delle risorse sul broker, creando sprechi di energia e capacità computazionale.

Capitolo 3
MQTT v3.1 vs v5

La nuova versione di MQTT (versione 5) implementa caratteristiche che rendono il protocollo più robusto rispetto alla versione iniziale (versione 3.1) dal punto di vista della scalabilità di utilizzo, della sicurezza e del controllo degli errori, della connessione e degli utilizzatori, aggiungendo nuovi tipi di pacchetti di controllo e aggiungendo dei campi nei pacchetti già esistenti.
Vediamo ora in dettaglio queste caratteristiche.

3.1 **Scalabilità di utilizzo**

Con scalabilità di utilizzo si intende la capacità del protocollo di sopportare ed avere prestazioni efficienti quando applicato a sistemi di grandi dimensioni, con un ampio flusso di messaggi relativo a diversi argomenti (topic), popolato da un elevato numero di utilizzatori. Le migliorie apportate in quest'ottica sono l'aggiunta di campi del variable header e del payload di alcuni tipi di pacchetti, quali Payload format, Content type, Topic alias, Subscriptions ID e la possibilità di avere delle Shared Subscriptions.

3.1.1 *Payload format :*

Il campo payload format è presente nell'header dei pacchetti di tipo PUBLISH e nel payload di pacchetti di tipo CONNECT, ed è un byte che indica se il payload dei pacchetti publish sarà di tipo binario (raw bytes) o se sarà una stringa codificata.

3.1.2 *Content type :*

Il campo content type è presente nell'header di pacchetti di tipo PUBLISH e nel payload di pacchetti CONNECT ed è un campo a lunghezza variabile che indica il tipo di dato che sarà presente nei pacchetti PUBLISH e la sua codifica. Ad ora l'unica

restrizione che si ha su questo campo è che la stringa sia codificata in UTF-8, anche se il tipo maggiormente utilizzato è di tipo MIME (standardizzato da IANA).

3.1.3 *Topic Alias :*

Il campo topic alias è presente nell'header o nel payload di vari pacchetti, e ha lo scopo di assegnare ad un certo topic un numero intero che lo identifichi. L'assegnazione può avvenire indipendentemente sia da un client che dal broker, a seguito di autorizzazioni da parte del client che può informare il broker riguardo la possibile capacità di alias che potrà gestire, attraverso il campo Maximum Topic Alias presente in pacchetti CONNECT (0 se non supporta il servizio).Dopo l'assegnazione di un alias, sia il broker che il client comunicheranno non più attraverso il campo Topic (che rimarrà inutilizzato nelle comunicazioni successive), ma attraverso l'alias che hanno concordato, per poter risparmiare spazio nel pacchetto non dovendosi scambiare stringhe che possono avere lunghezze significative. Il topic alias è unico per una singola connessione Broker-Client, non all'interno del broker stesso, e questo comporta che due client diversi possono avere due topic alias diversi per riferirsi allo stesso topic.

3.1.4 *Subscription ID* :

Il campo subscription ID è presente nell'Header di pacchetti PUBLISH e SUBSCRIBE, ed è utilizzato dal client per poter risalire a quale topic ha causato l'arrivo del messaggio. All'iscrizione ad un topic, il client può assegnare un ID a tale iscrizione e quando riceverà un messaggio dovuto ad una pubblicazione in quel topic il pacchetto PUBLISH che riceverà conterrà anche l'informazione relativa al subscription ID. Questo meccanismo garantisce a client che sono iscritti a un gran numero di topic la possibilità di eliminare confusioni riguardo la provenienza dei messaggi.

3.1.5 *Shared subscriptions :*

Una shared subscription è una iscrizione ad un topic divisa da varie sessioni di vari client, e può essere vista come un solo client che funge da proxy per altri client. Ciò

può essere utile per non congestionare una rete nella quale molti client hanno bisogno di accedere alla stessa risorsa del broker.

3.2 **Sicurezza**

In termini di sicurezza della comunicazione, MQTT v3.1 si basava su altri protocolli a livello di trasporto (in termini ISO-OSI) come TLS, e l'unico metodo di autenticazione interno al protocollo stesso era l'assegnazione di username e password relativi ad un client, che dovevano essere forniti al broker per potersi connettere. La nuova versione MQTT v5 aggiunge a questo metodo di base la possibilità di avvalersi a tecniche di autenticazione avanzate di tipo challenge/response, che vengono specificate dal client o dal server in base a quali sono le tecniche supportate dall'implementazione degli utilizzatori (client e broker). Questo avviene alla creazione di una connessione: con l'invio del pacchetto CONNECT da parte del client, esso informa il broker riguardo il proprio Username la fornisce la password e dichiara quale tipo di autenticazione avanzata supporta, attraverso il campo Authentication method. Il broker risponderà con un pacchetto di tipo AUTH, con il quale può inviare informazioni riguardo il metodo di autenticazione avanzata o può chiedere al client di fornire più informazioni. Nel caso il broker non supporti il metodo di autenticazione richiesto dal client, esso manderà un pacchetto CONNACK con un reason code che indica la mancata autenticazione e chiuderà la connessione. Le informazioni riguardo l'autenticazione avanzata vengono scambiate attraverso pacchetti AUTH, seguono sempre un pacchetto CONNECT e il loro flusso viene interrotto da un pacchetto CONNACK, che sancisce l'instaurazione della connessione o che la interrompe.

3.3 **Controllo degli errori**

Nella vecchia versione di MQTT non era prevista dal protocollo nessuna maniera di scambiare, tra client e server, informazioni riguardo errori o problemi che potevano verificarsi durante qualsiasi momento della connessione. La versione 5 aggiunge ai pacchetti di acknowledgement un campo chiamato Reason code, che permette di specificare quale errore si è verificato attraverso dei codici di errore definiti dal protocollo, e un campo Reason string, che permette di aggiungere una stringa per

completare e chiarificare cosa può essere andato storto nel caso non basti il reason code. All'interno dei Reason code è presente anche il codice che indica l'avvenuto successo dell'operazione.

REASON CODE		NOME	PACCHETTI
DECIMALE	ESADECIMALE		
0	0x00	Success	CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UNSUBACK, AUTH
0	0x00	Normal Disconnection	DISCONNECT
0	0x00	Granted QoS 0	SUBACK
1	0x01	Granted QoS 1	SUBACK
2	0x02	Granted Qos 2	SUBACK
4	0x04	Disconnect with Will Message	DISCONNECT
16	0x10	No matching subscribers	PUBACK, PUBREC
17	0x11	No subscription existed	UNSUBACK
24	0x18	Continue authentication	AUTH
25	0x19	Re-authenticate	AUTH
128	0x80	Unspecified error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
129	0x81	Malformed Packet	CONNACK, DISCONNECT
130	0x82	Protocol error	CONNACK, DISCONNECT
131	0x83	Implementation specific error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
132	0x84	Unsupported Protocol Version	CONNACK
133	0x85	Client Identifier not valid	CONNACK
134	0x86	Bad username o password	CONNACK
135	0x87	Not authorized	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
136	0x88	Server unavailable	CONNACK
137	0x89	Server busy	CONNACK, DISCONNECT
138	0x8A	Banned	CONNACK

REASON CODE		NOME	PACCHETTI
DECIMALE	ESADECIMALE		
139	0x8B	Server shutting down	DISCONNECT
140	0x8C	Bad authentication method	CONNACK, DISCONNECT
141	0x8D	Keep alive timeout	DISCONNECT
142	0x8E	Session taken over	DISCONNECT
143	0x8F	Topic Filter invalid	SUBACK, UNSUBACK, DISCONNECT
144	0x90	Topic Name invalid	CONNACK, PUBACK, PUBREC, DISCONNECT
145	0x91	Packet Identifier in use	PUBACK, PUBREC, SUBACK, UNSUBACK
146	0x92	Packet Identifier not found	PUBREL, PUBCOMP
147	0x93	Receive Maximum exceeded	DISCONNECT
148	0x94	Topic Alias invalid	DISCONNECT
149	0x95	Packet too large	CONNACK, DISCONNECT
150	0x96	Message rete too high	DISCONNECT
151	0x97	Quota exceeded	CONNACK, PUBACK, PUBREC, SUBACK, DISCONNECT
152	0x98	Administrative action	DISCONNECT
153	0x99	Payload format invalid	CONNACK, PUBACK, PUBREC, DISCONNECT
154	0x9A	Retain not supported	CONNACK, DISCONNECT
155	0x9B	QoS not supported	CONNACK, DISCONNECT
156	0x9C	Use another server	CONNACK, DISCONNECT
157	0x9D	Server moved	CONNACK, DISCONNECT
158	0x9E	Shared Subscriptions not supported	SUBACK, DISCONNECT
159	0x9F	Connection rate exceeded	CONNACK, DISCONNECT
160	0xA0	Maximum connect time	DISCONNECT
161	0xA1	Subscription Identifier not supported	SUBACK, DISCONNECT
162	0xA2	Wildcard Subscriptions not supported	SUBACK, DISCONNECT

tabella 3-3.1 : Reason codes con significato e pacchetti di appartenenza

3.4 **Controllo e gestione della connessione**

La connessione tra broker e client viene migliorata con l'aggiunta di vari campi nei pacchetti CONNECT e PUBLISH, tra cui :

3.4.1 *Clean start :*

Bit che indica se all'avvio della connessione deve essere avviata una nuova sessione o se può essere ripristinata una ipotetica sessione precedentemente avviata.

3.4.2 *Session expiry interval :*

Campo numerico che indica dopo quanto tempo può essere chiusa la sessione a seguito di una disconnessione da parte del client o del server.

3.4.3 *User properties :*

Coppia di stringhe che permette al client di informare il server riguardo particolari proprietà riguardo la connessione. Non vi è alcuno standard preciso sul significato di queste proprietà.

3.4.4 *Assigned client ID :*

Stringa che identifica il client ID assegnato dal server ad un client che non ne ha fornito uno all'avvio della connessione.

3.4.5 *Server keep alive :*

Campo numerico contenente un intervallo di tempo stabilito dal server come keep alive per il client.

3.4.6 *Message expiry interval :*

Intervallo di tempo che definisce il tempo di vita di un messaggio all'interno del broker.

3.4.7 *Server reference :*

Stringa codificata in UTF-8 che indica il riferimento ad un altro server al quale il client può connettersi in caso di problemi con la sessione corrente. È accompagnato da un reason code di tipo 0X9C/D (Use another server o Server moved) .

Capitolo 4
ARCHITETTURA REQUEST/RESPONSE E TEST SPERIMENTALI

All'interno di una rete, è possibile che vi sia la necessità di richiedere informazioni ad un dispositivo che funge da server, il quale deve poter rispondere singolarmente ad ogni richiesta. Un ulteriore aggiornamento che è stato aggiunto alla versione 5 è la possibilità di instaurare architetture di tipo Request/Response appoggiandosi sull'architettura publish/subscribe già implementata da MQTT.

4.1 **Connessione Request/Response**

Il meccanismo che utilizza MQTT per inoltrare richieste e risposte nella rete si basa sull'utilizzo di più topic distinti : un topic comune nel quale i client possono pubblicare le loro richieste verso un server connesso al broker ed un topic per ogni client nel quale il server pubblica le risposte alle richieste.

Il server si iscrive al topic comune (comunemente denominato org/common), dal quale riceve le richieste dei vari client. Le richieste sono pacchetti PUBLISH nel cui header sono settati il campo Response topic, nel quale è indicato il topic sul quale il server dovrà pubblicare la propria risposta (comunemente denominati org/responses/_client_, dove _client_ è un identificatore del client che ha effettuato la richiesta), ed il campo Correlation data, utilizzato da client e server per collegare la richiesta alla risposta. La richiesta della risorsa è contenuta nel payload del pacchetto PUBLISH, e la sua formattazione non è normata ed è rimandata al consumatore.

Le risposte sono anch'esse pacchetti PUBLISH, pubblicati sul topic indicato dal client nel Response topic, contenenti lo stesso valore del Correlation data presente nella richiesta, nel cui payload viene inserito il valore della risorsa richiesta.

Il nome del response topic adottato da ogni client non è normato, e può essere deciso in base alle scelte implementative della rete, facendo prendere la decisione al client, il quale può avere completa libertà di scelta o può basarsi su delle limitazioni impostegli dal broker, il quale può essere impostato per assegnare degli alberi di sotto-topic ad ogni client. Il broker assegna gli alberi di sotto-topic attraverso il campo Response information dei pacchetti CONNACK previa richiesta del client, il quale può settare a 1 lo stesso campo nel suo pacchetto CONNECT all'avvio della connessione.

4.2 **Implementazione modalità Request/Response su MQTT**

La modalità Request/Response è stata introdotta in MQTT per poter permettere a dispositivi con scarse capacità di calcolo di poterne usufruire, in quanto questa architettura è attualmente realizzata attraverso il protocollo HTTP, il quale si è dimostrato richiedere un maggior sforzo in termini di memoria e CPU utilizzata.

4.2.1 *Dispositivi e architettura della rete*

Per l'implementazione il client, il server ed il broker sono allocati sulla stessa macchina (un moderno laptop) con processore da 1.8 GHz, e sono 3 programmi lanciati da riga di comando. La rete utilizzata è una rete domestica wireless (WLAN).

4.2.2 *Broker*

Come broker è stato utilizzato il programma mosquitto, scritto in C, che offre un broker compatibile con la versione 5 e che implementa tutte le nuove funzionalità, tra cui la gestione di end-point che necessitano di inoltrare richieste e risposte. È un programma open source ed è di libero utilizzo.

4.2.3 *Paho MQTT*

Paho è la libreria usata per scrivere il client ed il server. Scritta in python, offre la struttura e le funzioni per istanziare un client MQTT che sia compatibile con la versione 5. È gratuita, open source e liberamente scaricabile.
Utilizzando Paho, la creazione di un client è semplice:

- Il costruttore è Client(), e riceve come parametri l'identificatore del client, la versione del protocollo e il protocollo di trasporto.

- Con la funzione connect() ci si collega al broker, passandogli come valori il nome dell'host (indirizzo IP o server DNS), la porta TCP, il tempo di keepAlive ovvero il tempo massimo tra una trasmissione e l'altra e il clean start.

  Questa funzione lancia il callback on_connect(), da implementare nel codice, che restituisce il risultato della connessione presente nel CONNACK mandato dal broker.
- La funzione subscribe() permette di iscriversi ad un topic all'interno del broker, e prende come parametri il topic filter, il QoS e le proprietà aggiuntive del pacchetto SUBSCRIBE.
- La funzione publish() permette di pubblicare messaggi sui topic, prendendo come parametri il topic di pubblicazione, il messaggio da pubblicare, il QoS e le proprietà aggiuntive del pacchetto PUBLISH
- La funzione loop_ attiva l'ascolto sulla porta TCP indicata, e lancia la funzione on_message() all'arrivo di un pacchetto di tipo PUBLISH. La funzione on_message() non è definita in Paho, ma è da implementare nel codice in base ai compiti che deve svolgere.

```
def on_connect(client, userdata, flags, rc, properties):
        print("Result of connection :  "+str(rc))
        pass
def on_message(client,userdata,message):
        ##
        ##  Implementazione variabile in base all'applicazione
        ##
        pass

# istanziamento client#
client=mqtt.Client(client_id="", clean_session=None, userdata=None,
       protocol=MQTTv311, transport="tcp", reconnect_on_failure=True)

#connessione al broker#
client.connect( host, port=1883, keepalive=60, bind_address="", bind_port=0,
                clean_start=MQTT_CLEAN_START_FIRST_ONLY, properties=None)

#iscrizione ad un topic#
client.subscribe( topic, qos=0, options=None, properties=None)

#pubblicazione su un topic#
client.publish(topic, payload=None, qos=0, retain=False, properties=None)

#ascolto sulla porta#
client.loop_forever()
```

figura 4-2-3.1 : implementazione base client MQTT con Paho

4.2.4 *Client in request mode*

In modalità Request/Response il client, una volta collegatosi al broker, deve iscriversi al proprio Response topic e successivamente inoltrare la richiesta al topic comune, allegando al pacchetto il response topic e il correlation data. Per fare questo si deve creare l'oggetto proprietà e modificarne gli attributi, per poi passare questo oggetto alla funzione publish come parametro. La richiesta viene inserita nel payload del messaggio.

```
resp_topic="org/responses/Client_ID"
corr_data=## da definirsi in base all'applicazione
client.subscribe(resp_topic)
properties=Properties(PacketTypes.PUBLISH)
properties.ResponseTopic=resp_topic
properties.CorrelationData=corr_data
client.publish("org/common",message,properties=properties)
```

figura 4-2-4.1 : settaggio pacchetto di richiesta

Inviata la richiesta, si mette in ascolto della risposta tramite loop_(). All'arrivo della risposta è premura del client controllare che essa sia conforme alla richiesta, attraverso il correlation data ricevuto. Tutto ciò è demandato alla funzione on_message().

```
def on_message(client, userdata, message):
        response=message.payload
        correlationData=message.properties.CorrelationData
        if correlationDat == requestedCorrelationData :
                print("Response = ",response)
        else :
                print("Wrong Response!!")
```

Figura 4-2-4.2 : controllo sulla risposta

4.2.5 *Server in response mode*

Il server, dopo essersi collegato al broker, si iscrive al topic comune di invio richieste "org/common", e si mette in ascolto. All'arrivo di un pacchetto PUBLISH, viene chiamata la funzione on_message(), la quale è incaricata di svolgere i controlli sul messaggio arrivato e decidere come comportarsi successivamente. Come prima cosa salva il messaggio ricevuto, nel quale è presente una eventuale richiesta. Effettua poi un controllo sulle proprietà del pacchetto, e stabilisce se esso è una richiesta dalla presenza del campo ResponseTopic. Nel caso sia presente, vengono salvati sia il ResponseTopic che il CorrelationData. L'ultimo step di controllo è l'interpretazione della richiesta. A questo punto il server è pronto per inviare la risposta, in quanto il pacchetto PUBLISH che verrà inviato avrà come topic il ResponseTopic salvato in

precedenza, come payload l'informazione richiesta e come proprietà il CorrelationData presente nella richiesta.

```
def on_message(client, userdata,message):
        request=message.payload
        responseTopic=message.properties.ResponseTopic
        correlationData=message.properties.CorrelationData
        if responseTopic != "" :
                response = parseResponse(request)
                properties=Properties(PacketTypes.PUBLISH)
                properties.CorrelationData=correlationData
                client.publish(responseTopic,response,properties=properties)
```

figura 4-2-5.1 : interpretazione richiesta e invio risposta

4.3 **Test prestazioni**

Viste le caratteristiche e l'implementazione di questa modalità, si possono studiarne le prestazioni in termini di latenza di trasmissione, di utilizzo di CPU e di consumo di memoria.

4.3.1 *Latenze*

Con latenze si intende il tempo che intercorre tra l'invio di una richiesta e la ricezione della risposta, ovvero il round trip time(RTT) della comunicazione.

Per fare ciò è stato semplicemente inviato nel payload della richiesta un timestamp con il tempo di invio, il quale viene comparato con un timestamp del tempo di ricezione della risposta per ottenere l'RTT.

I timestamp sono stati presi attraverso perf_counter(), un metodo presente nella libreria standard time di python. Perf_counter() ritorna un numero decimale in virgola mobile di secondi calcolati da un performance counter, ovvero dal clock interno con la maggiore risoluzione. Il momento di inizio del counter non è definito, e quindi è utilizzabile solo comparando due letture.

Il timestamp di inizio è stato preso subito prima del comando publish(), ed è stato inserito nel payload della richiesta inviata dal client. Il server è stato istruito a rispondere con lo stesso payload presente nella richiesta.

```
for i in range(n_invii)
        client.connect(host,1883,60)
        client.subscribe("org/responses/client")
        properties=Properties(PacketTypes.PUBLISH)
        properties.ResponseTopic="org/responses/client"
        properties.CorrelationData=bytes("time","UTF-8")
        timestamp1=str(perf_counter())
        client.publish("org/common",timestamp1,properties=properties)
        client.loop_forever()
```

figura 4-3-1.1 : inserimento e invio del primo timestamp

Il timestamp finale è stato preso all'arrivo del messaggio, nella funzione on_message(), subito dopo il controllo sul campo Correlation Data. A questo punto è possibile fare la sottrazione tra il timestamp presente nel payload e quello preso all'arrivo del messaggio per risalire all'RTT, il quale viene salvato all'interno di un file.

```
def on_message(client, userdata, message):
        msg=str(message.payload.decode("utf-8"))
        cd = message.properties.CorrelationData
        if cd == bytes("time","UTF-8"):
                t2 = perf_counter()
                elapsed= t2 - float(msg)
                file = open("C:\\Desktop\latenze_MQTT.txt","a")
                file.write(str(elapsed))
                file.write("\n")
                file.close()
        client.disconnect()
```

figura 4-3-1.2 : calcolo del tempo trascorso e inserimento in un file

Questa procedura viene ripetuta per un numero ragionevole di volte, al fine di avere un pool di misurazioni che permetta di stimare accuratamente il RTT medio.
Di seguito è presente il codice intero con il quale è stato implementato il client che esegue le richieste e calcola il tempo medio di invio.

```
import paho.mqtt.client as mqtt
import paho.mqtt.properties as prop
from time import perf_counter

def on_connect(client, userdata, flags, rc, properties):
    print("Result of connection :  "+str(rc))


def on_message(client, userdata, message):
    #interpretazione messaggio ricevuto
    msg=str(message.payload.decode("utf-8"))
    cd = message.properties.CorrelationData
    if cd == bytes("time","UTF-8"):
        t2 = perf_counter()
        elapsed= t2- float(msg)
        file = open("C:\\Users\Diego\Desktop\latenze_MQTT.txt","a")
        file.write(str(elapsed))
```

```
        file.write("\n")
        file.close()
    else :
        print(msg)
        print("\n")

    client.disconnect()


client = mqtt.Client(client_id="Diego",protocol=5)
client.on_connect=on_connect
client.on_message = on_message
#ciclo di ripetizione test
for i in range(1000):
    client.connect("192.168.1.74", 1883, 60)
    properties=prop.Properties(prop.PacketTypes.PUBLISH)
    properties.ResponseTopic="org/responses/Diego"
    properties.CorrelationData=bytes("time","UTF-8")
    client.subscribe("org/responses/Diego")
    contenuto = str(perf_counter()) # modifica da testare
    client.publish("org/common",contenuto,properties=properties)
    client.loop_forever()
#fine ciclo ripetizione test
#calcolo media tempi trasmissione
sommaT=0
count=0
file = open("C:/Users/Diego/Desktop/latenze_MQTT.txt" , "r")
for line in file :
    tempo=float(line)
    sommaT+=tempo
    count+=1
file.close()
mediaT = sommaT/count
print("-------------------------------------------------------------")
```

```
print("Il tempo medio di ricezione a partire dalla richiesta è
:",mediaT, "s")
print("N° campioni : ",count)
print("-------------------------------------------------------------")
print(“---FINE TEST---“)
```

Il risultato del test mostra come il tempo medio di Round Trip sia di 27.43 ms.

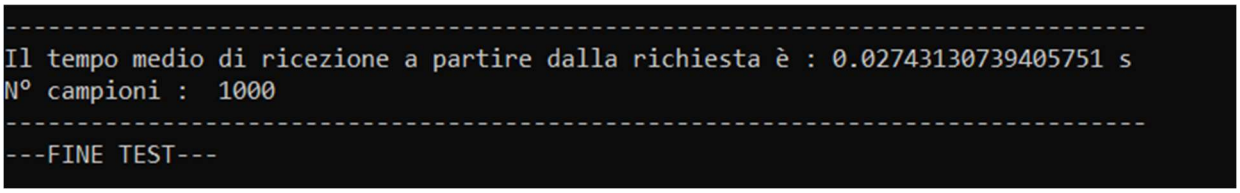

figura 4-3-1.3 : risultato del test

4.3.2 *Utilizzo CPU*

Per stimare l’utilizzo della CPU in trasmissioni MQTT è stato monitorato il processore durante l’esecuzione dello script python. Lo script del client usato per questo test è una versione alleggerita del codice utilizzato in precedenza, in cui è stato rimosso il processo di immagazzinamento delle letture effettuate e il calcolo della media.

```
import paho.mqtt.client as mqtt
import paho.mqtt.properties as prop
from time import perf_counter
```

```
def on_connect(client, userdata, flags, rc, properties):
    print("Result of connection :  "+str(rc))

def on_message(client, userdata, message):
    #interpretazione messaggio ricevuto
    msg=str(message.payload.decode("utf-8"))
    t2 = perf_counter()
    cd = message.properties.CorrelationData
    if cd == bytes("time","UTF-8"):
        elapsed= t2- float(msg)
        #calcolo memoria allocata
        print("Il tempo di trasmissione è :",elapsed,"s")
    elif cd == bytes("stop","UTF-8") :
        print("Messaggio di chiusura arrivato")
    else :
        print(msg)
        print("\n")

    client.disconnect()

def req_resp():
    #connecting to broker
    client.connect("192.168.27.48", 1883, 60)
    #subscribing to response topic
    client.subscribe("cubo/responses/Diego")
    #packet setup
    properties=prop.Properties(prop.PacketTypes.PUBLISH)
    properties.ResponseTopic="cubo/responses/Diego"
    properties.CorrelationData=bytes("time","UTF-8")
    contenuto = str(perf_counter()) # modifica da testare
    #packet publish
    client.publish("cubo/common",contenuto,properties=properties)
    #listening for response
    client.loop_forever()
```

```
    print("trasmissione avvenuta")
    pass

start = input("iniziare il programma ?")
#client setup
client = mqtt.Client(client_id="Diego",protocol=5)
client.on_connect=on_connect
client.on_message = on_message
#iterazione test
for i in range(n° trasmissioni):
    req_resp()
    pass
#conclusione test
stop = input("concludere il programma ?")
if stop == "si" :
    client.connect("192.168.27.48",1883,60)
    client.subscribe("cubo/responses/Diego")
    properties=prop.Properties(prop.PacketTypes.PUBLISH)
    properties.ResponseTopic="cubo/responses/Diego"
    properties.CorrelationData=bytes("stop","UTF-8")
    client.publish("cubo/common","FINE TEST",properties=properties)
    client.loop_forever()
```

Dal lato server invece viene utilizzato sempre lo stesso codice per lo svolgimento di tutti i test.

```
import paho.mqtt.client as mqtt
import paho.mqtt.properties as prop
```

```
def on_connect(client, userdata, flags, rc, properties):
    print("Result of connection :  "+str(rc))


def on_message(client, userdata,message):
    msg=str(message.payload.decode("utf-8"))
    print('RECV Topic = ',message.topic)
    print('RECV MSG =', msg)
    cd=message.properties.CorrelationData
    response_topic = message.properties.ResponseTopic
    properties=prop.Properties(prop.PacketTypes.PUBLISH)
    properties.CorrelationData=cd
    if cd == bytes("time","UTF-8") :
        print("Correlation Data ok")
        print('Responding on response topic:', response_topic)
        print("\n")
        client.publish(response_topic,msg,properties=properties)
    elif cd == bytes("stop","UTF-8"):
        print("conclusione test")
        client.publish(response_topic,msg,properties=properties)
        print("inviata conclusione")
        client.disconnect()
    else :
        print("Errore di richiesta!!")
        print("\n")
        client.publish(response_topic,"Errore di
richiesta!!",properties=properties)



client=mqtt.Client(client_id="Server",protocol=5)
client.on_connect=on_connect
client.on_message=on_message
client.connect("192.168.1.74", 1883, 60)
```

```
client.subscribe("org/common")
client.loop_forever()
```

Per monitorare il processore è stato utilizzato il programma procexp, il quale offre letture di buona qualità per il controllo dei processi e del loro consumo in termini hardware.

I processi di client e server monitorati eseguono mille trasmissioni, e il loro utilizzo di CPU è mostrato dalle catture :

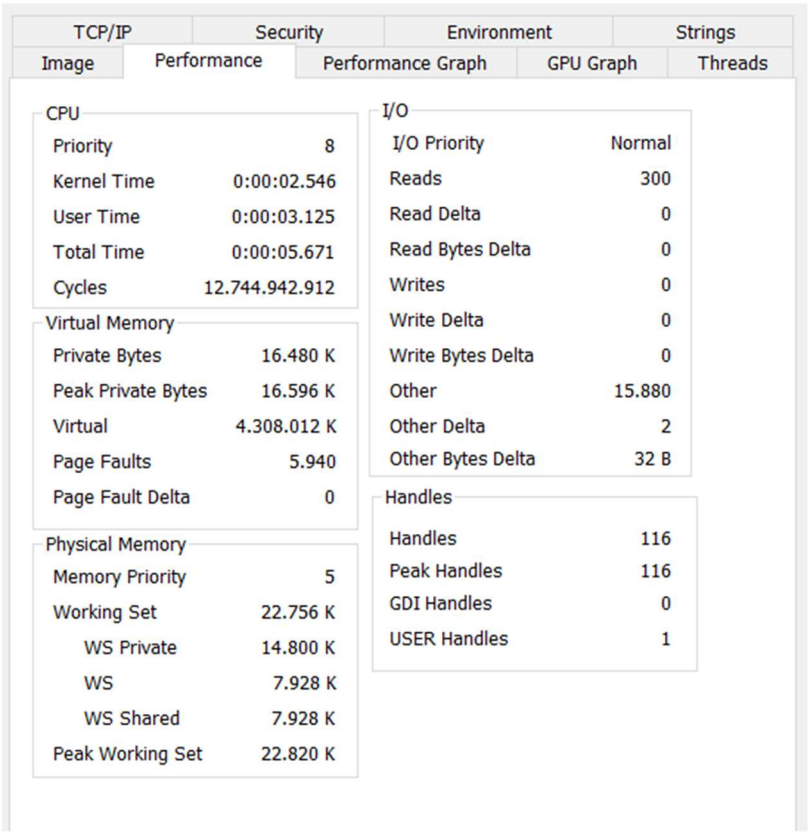

Figura 4-3-2.1 : Utilizzo CPU client per 1000 trasmissioni

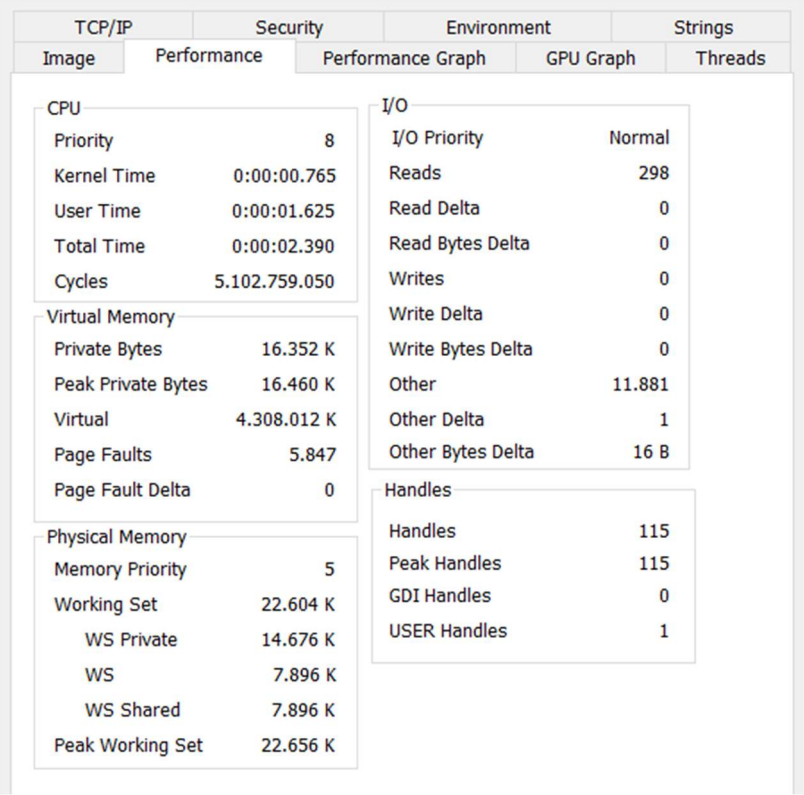

figura 4-3-2.2 : Utilizzo CPU server per 1000 trasmissioni

4.3.3 *Consumo memoria*

Per il calcolo di memoria utilizzata dai processi è stato utilizzato un tool messo a disposizione da python, il memory profiler, che monitora l'utilizzo di memoria e permette di graficare le letture.

Il codice utilizzato per il client e il server è lo stesso utilizzato per il monitoraggio della CPU. Il numero di trasmissioni effettuate è 100, in quanto non serve più un ampio numero di campioni per avere una buona statistica ma solo un numero sufficiente di cicli per distinguere lo stato transitorio di caricamento delle librerie dell'interprete python dallo stato a regime delle trasmissioni.

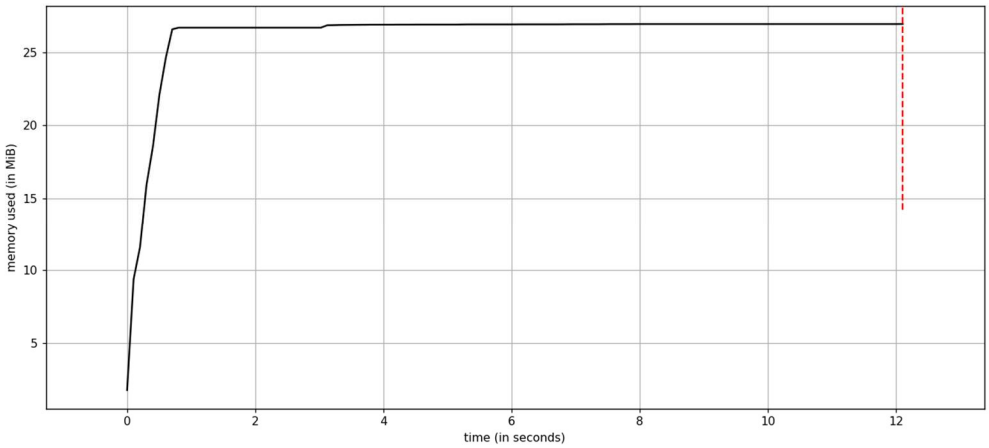

figura 4-3-3.1 : Consumo memoria client

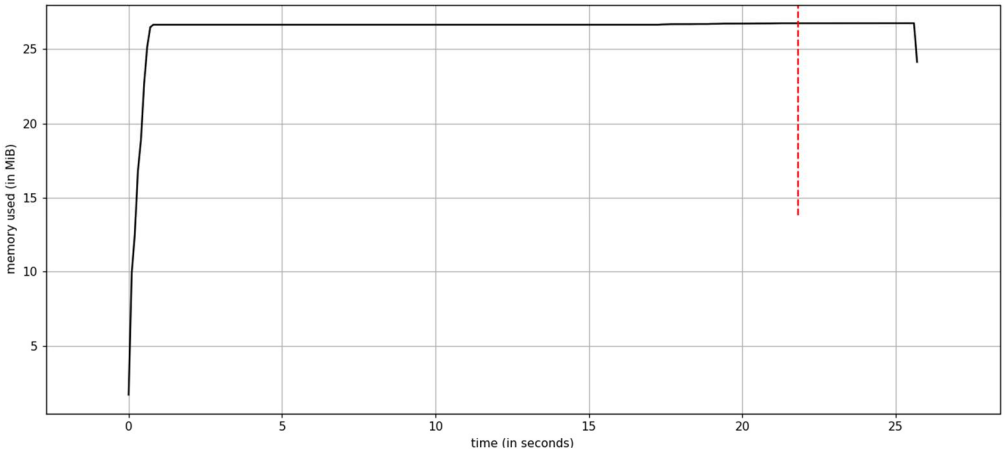

Figura 4-3-3.2 : Consumo memoria server

Come si può notare dai grafici, la memoria allocata dai processi si assesta attorno al valore di 27 MiB (Mebibyte), che corrisponde a 28 MB di memoria.

Dalla sperimentazione eseguita, vediamo i risultati, comparati con i risultati ottenuti da test già effettuati su http.

	MQTT	**HTTP**
RTT (Round trip time)	27 ms	70 ms
MEMORIA	28 Mb	56 Mb
CPU (tempo di utilizzo)	5.6 s	17.5 s

CONCLUSIONI

A fronte degli esperimenti effettuati, è possibile confrontare le prestazioni delle funzionalità di tipo Request/Response del protocollo MQTT con le prestazioni del protocollo attualmente più utilizzato per i traffici internet di questo tipo, il protocollo http.
Studi effettuati su http mostrano come il Round trip time medio di una comunicazione sia di 70 ms, l'utilizzo di CPU sia attorno ai 17.5 secondi per la trasmissione di 1000 messaggi e la memoria necessaria sia 56 MB.
È facile notare quindi che il protocollo MQTT sia più conveniente sotto vari punti di vista :

- I tempi di trasmissione sono minori rispetto ad http (27 ms contro 70 ms)
- L'utilizzo di CPU è minore (5.6 s contro 17.5 s)
- La memoria richiesta è minore (28 MB contro 56 MB)

Ciò che emerge da questo studio è che MQTT è UN protocollo versatile e leggero permette di essere adottato in svariati campi e si adatta facilmente alle infrastrutture sulle quali viene implementato, incluse architetture con risorse limitate. Dimostra di evolversi con il progredire della tecnologia e delle richieste del mercato, ed è per questo che viene adottato ad oggi come lo standard per l'IoT.

RINGRAZIAMENTI

Arrivato al termine della trattazione, mi sembra doveroso ringraziare alcune persone, senza le quali non sarei riuscito a raggiungere l'obiettivo di apprendere le conoscenze che mi hanno permesso di scrivere questa tesi.

In primo luogo, devo ringraziare i miei genitori, i quali mi hanno supportato, sopportato e sostenuto e continueranno sicuramente a farlo per il resto della mia vita.

Il secondo ringraziamento va a tutti i membri della mia famiglia, che mi hanno motivato nel percorso di studi.

Ringrazio i miei compagni di corso, con cui ho condiviso le gioie e le fatiche dello studio e della vita universitaria.

Ringrazio i miei amici, che sempre mi sono stati vicini nel bene e nel male e con i quali ho passato gli anni migliori della mia vita.

L'ultimo ringraziamento va ad Auser di Fano, che ha messo a disposizione la struttura fisica che mi ha permesso di raggiungere questo traguardo.

BIBLIOGRAFIA

- Oracle Italia, Che cos'è l'internet of things (IoT)?
- Sivas informatica, I 12 protocolli per l'IoT
- Standard, O. A. S. I. S. "MQTT Version 5.0." *Retrieved June* 22 (2019): 2020.
- Friendly tecnologies, 2022, LwM2M vs MQTT: A Head-to-Head Comparison
- Raschbichler F., 2019, MQTT essential , HiveMQTT
- Luoto, Antti, and Kari Systä. "Fighting network restrictions of request-response pattern with MQTT." *Iet Software* 12.5 (2018): 410-417.
- Esposito, Marco, et al. "Design and Implementation of a Framework for Smart Home Automation Based on Cellular IoT, MQTT, and Serverless Functions." *Sensors* 23.9 (2023): 4459.
- Steve's Internet Guide, 2022, Understanding And Using MQTT v5 Request Response
- Ghithub , eclipse/Paho.mqtt.python
- Spero, Simon E. "Analysis of http performance problems." *http://www. w3. org/Protocols/HTTP/1.0/HTTPPerformance. html* (1994)
- Figura 1-1: https://docs.devicewise.com/Content/GettingStarted/Overview-of-LWM2M.htm
- Figura 1-2 : https://www.twilio.com/blog/what-is-mqtt