ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA, INFORMATICA E TELECOMUNICAZIONI-Ambito ElettronicoTelecomunicazioni

TITOLO DELL'ELABORATO

Comunicazione tra dispositivi IoT e Cloud tramite protocollo MQTT

Elaborato in

Applicazioni e Tecniche di Telecomunicazioni

Relatore

Walter Cerroni

Presentato da

Fabrizio Rinaldini

Anno Accademico 2015/16

INDICE

1-INTERNET OF THINGS. L'Internet delle Cose

Con Internet of Things, abbreviato spesso con l'acronimo IoT, che in italiano viene tradotto come Internet delle Cose, stiamo ad indicare lo scenario in cui non sono solamente le persone in grado di connettersi alla Rete Globale mediante particolari dispositivi, come Computer, Smartphone o Tablet; ma sono gli oggetti stessi, di uso più comune tutti i giorni, direttamente connessi all'uso di Internet.

Il termine è stato coniato dall'ingegnere inglese Kevin Ashton per descrivere un sistema in cui il mondo fisico è connesso ad internet mediante dei sensori. Gli oggetti, in questo modo, avranno una propria disposizione all'interno della rete, che gli permetterà di essere visti sia da loro stessi, che dagli altri oggetti, che dagli utenti, uno degli scopi dell'Internet of Things è quello infatti, di creare una replica virtuale del mondo reale in cui viviamo.

Non esiste una particolare tipologia di cose che possono o non possono entrare a far parte del mondo IoT, a seconda delle necessità e del contesto in cui abbiamo bisogno di operare, possiamo implementare svariate soluzioni per connettere, e quindi rendere visibile nel mondo virtuale, ciò che ci interessa, l'importante è fornire all'oggetto due caratteristiche: come primo requisito un indirizzo IP (Internet Protocol) in modo da avere una collocazione nella rete e come seconda, la possibilità di comunicare senza bisogno che sia l'essere umano a dirgli di farlo. Se riusciamo a fare in modo che il nostro oggetto abbia queste due caratteristiche, allora esso diventerà il candidato ideale per entrare a far parte del mondo dell'Internet delle Cose.

Figura 1-Rappresentazione Astratta di un Sistema Internet of Things [1.a]

Secondo una previsione della Gartner, società leader mondiale nella ricerca e l'analisi nel campo dell'Information Technology, il numero di oggetti connessi in circolazione nel 2017 raggiungerà quota 8,3 miliardi, un aumento del 31% rispetto al 2016, per arrivare a quota 20,4 miliardi di unità nel 2020; rimanendo nel 2017 la spesa globale in terminali IoT e servizi connessi sarà di circa 2 mila miliardi, tutto questo ci fa capire di quanto l'industria dell'informazione sia propensa a svilupparsi in questo settore. [1.b]

Come detto in precedenza con le opportune caratteristiche ogni oggetto potrà quindi entrare a far parte della rete, questo significa che l'Internet of Things potrà trovare applicazione in tutti i settori della nostra vita, vediamo di citarne alcuni esempi per chiarirne meglio i concetti.
La Domotica, che è la scienza che si occupa delle tecnologie volte al fine di migliorare la qualità della vita all'interno della nostra abitazione, il cui nome nasce appunto dall'unione della parola Domus (casa in latino) e Robotica, è uno dei principali scenari applicativi dell'IoT. Fino a poco tempo fa infatti, un impianto di domotica necessitava di una specifica centralina subordinata al controllo delle parti ad esso collegato, quindi poteva risultare complesso gestire un numero elevato di device. Ora, fortunatamente, si trovano in commercio dispositivi ed elettrodomestici già in grado di connettersi ad

internet, questo significa che possiamo controllare a distanza la maggior parte degli oggetti presenti all'interno della nostra abitazione.

Figura 2-Rappresentazione Astratta di un Impianto di Domotica IoT [2.a]

È già stato lanciato sul mercato, ad esempio, un termostato intelligente in grado di analizzare le previsioni del tempo ed in base a questo regolare la temperatura in casa per un comfort ideale, in più, essendo connesso, può essere quindi controllato a distanza. Possiamo quindi attivarlo tramite smartphone con un comando manuale o far sì che lui si accorga del nostro imminente arrivo a casa quando il GPS del nostro telefono supera una distanza minima prestabilita.

Figura 3-Termostato Intelligente [3.a]

Tutto questo non è limitato al termostato, sempre grazie al GPS del nostro smartphone è possibile "fare svolgere una qualsiasi azione a casa nostra", ad

esempio possiamo fare aprire il nostro cancello automatico mentre stiamo per arrivare.

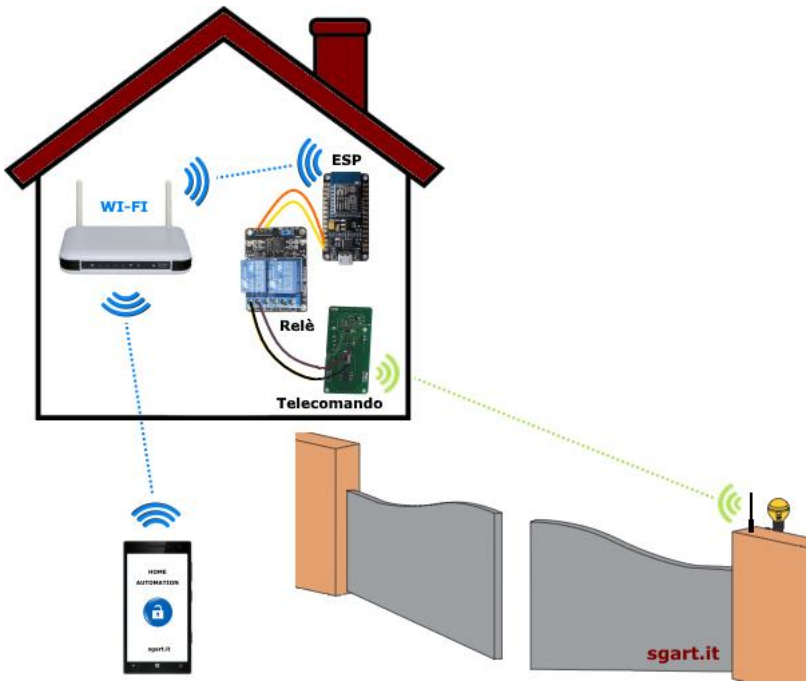


Figura 4-Componenti di un Impianto per il Controllo di un Cancello Automatico [4.a]

in campo di domotica sono già tantissimi in commercio gli elettrodomestici forniti di una connessione internet, il caso più comune sono le smart tv, ma tutto questo vale anche per forni, lavastoviglie e lavatrici, quest'ultime ad esempio, tramite il telefono, sono in grado di essere controllate da remoto e di informarci in base ai cicli di lavaggio.

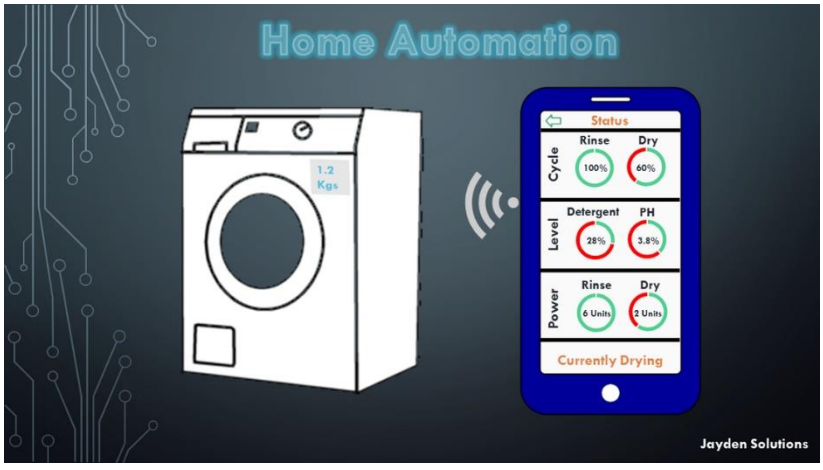


Figura 5-Lavatrice Smart [5.a]

Uscendo dalla domotica e proseguendo con l'analisi di altri campi, l'Internet of Things viene utilizzato per rintracciare bambini ed animali domestici. Un paio di scarpe con GPS integrato e relativa applicazione sullo smartphone, sono in grado di monitorare i movimenti di un bambino ed evitare di perderlo di vista.

Figura 6-Scarpe da Bambino con GPS Integrato [6.a]

La stessa cosa vale anche per collari per animali domestici, soprattutto cani e gatti, dotati di scheda SIM e GPS in modo che chi li indossa possa essere ritrovato dai loro padroni.

Figura 7-Collare GPS per Animali Domestici [7.a]

Molto curiosi anche i vari utilizzi in campo medico. Nel caso in cui, per alcuni pazienti la non assunzione di una medicina potrebbe risultare molto rischiosa per la loro salute, sono in fase di sviluppo delle pillole contenenti un trasmettitore in grado di inviare un segnale su un dispositivo remoto una volta raggiunto lo stomaco di chi le assume, in questo modo l'avvenuta somministrazione può essere verificata dal medico, dal paziente stesso o dai suoi familiari.

Figura 8-Pillole Ingeribili con RadioTrasmettitore [8.a]

Sempre in campo medico, in specifiche situazioni dove è fondamentale il controllo delle condizioni di una persona, pensiamo ad esempio ad un individuo diabetico che deve costantemente monitorare il proprio livello di glicemia nel sangue, nascono dei dispositivi da applicare sul corpo del paziente in grado di effettuare con intervalli di tempo prestabili le analisi necessarie e di inviare in tempo reale i risultati a chi “indossa" il monitor o a chi per lui.

In alcuni casi è il dispositivo stesso che si occupa di ristabilire la stabilità dei parametri.

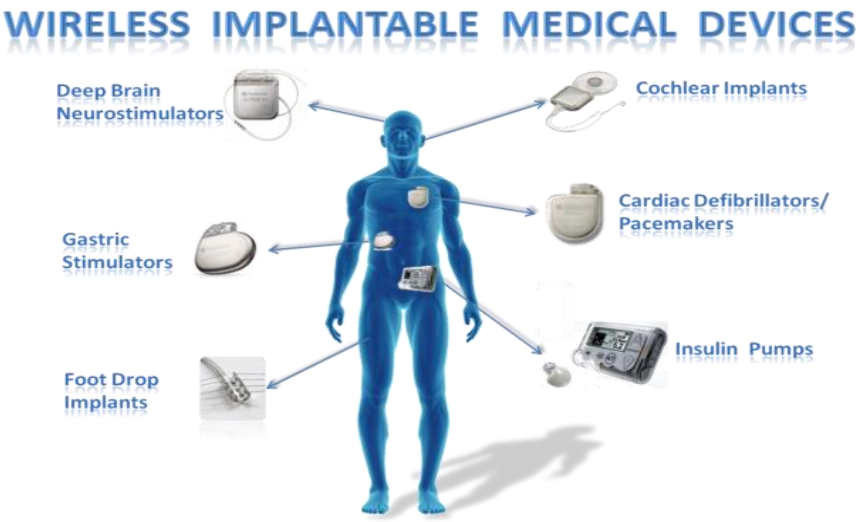


Figura 9-Dispositivi Medici Wireless Impiantabili [9.a]

Svariati i dispositivi IoT che possono essere installati sulle nostre automobili. Al fine di realizzare quello che gli studiosi di tecnologia dell’informazione chiamano Smart City, la nostra auto potrà ricevere informazioni riguardo al traffico e alla viabilità, aiutandoci a scegliere i percorsi più rapidi o le zone

dove si trova più parcheggio, d'altro canto l'automobile stessa potrà comunicare alla città dove si trova in modo che i semafori si regolino in base al flusso di viaggiatori; tutto questo potrà notevolmente ridurre l'inquinamento e gli sprechi di carburante.

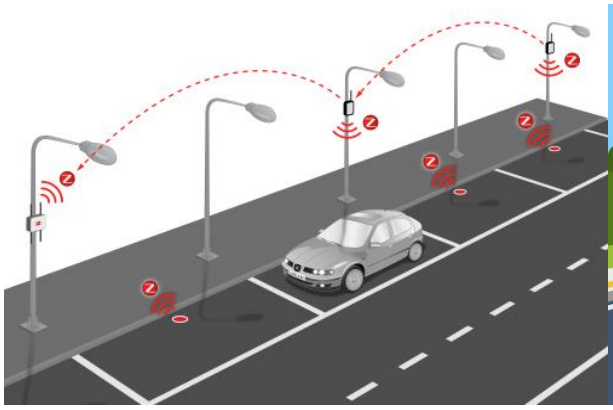


Figura 10-Gestione Parcheggi Smart City [10.a]

Figura 11-Gestione Traffico Smart City [11.a]

Le stesse case di produzione di automobili si stanno preoccupando di inserire a bordo dei propri prodotti, dei sensori che le tengono informate sullo stato dei veicoli, per monitorare i consumi e verificare se necessitano di manutenzione.

Figura 12-Rappresentazione Astratta di un'Automobile in grado di comunicare col mondo esterno [12.a]

Persino l'agricoltura potrà usufruire dell'internet delle cose, appositi sensori volti al controllo delle condizioni di campi e serre, saranno in grado analizzare

i livelli di umidità, temperatura e tutto quello che è necessario per il naturale sviluppo di una coltura.

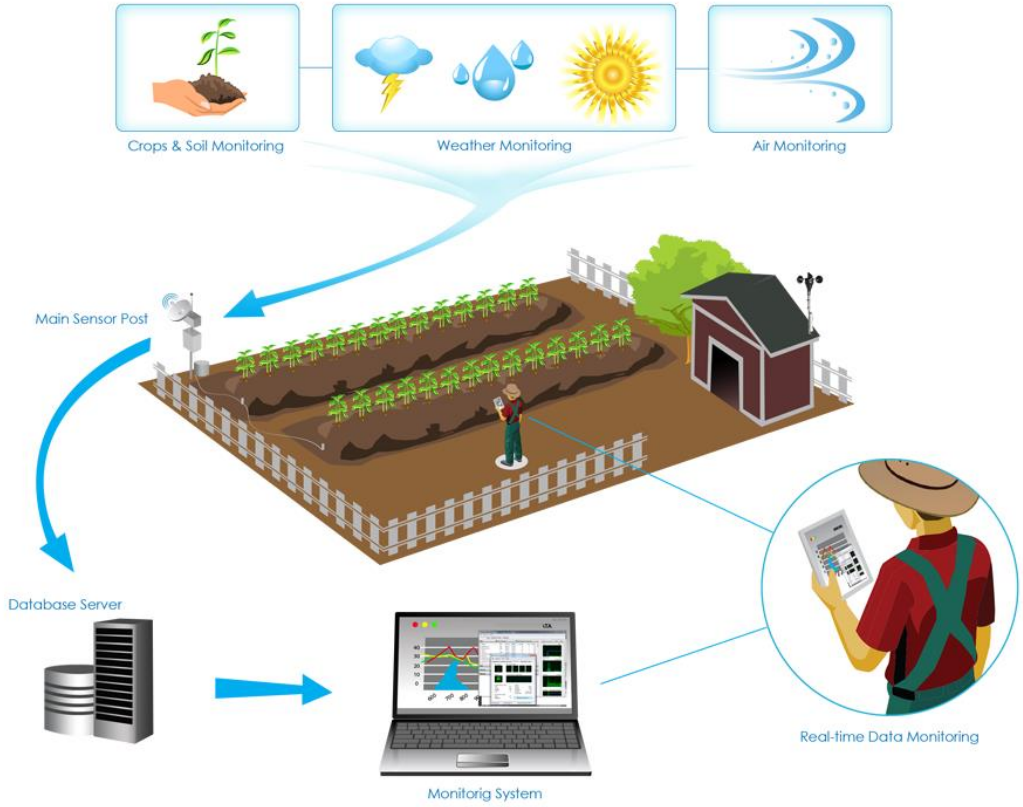


Figura 13-Esempio di Internet of Things applicato al Settore dell'Agricoltura [13.a]

Oppure lunghi condotti o gasdotti impiegati per il trasporto di sostanze saranno dotati dispositivi che posti tra loro a distanza strategica potranno analizzare lo stato della struttura e cercare di prevedere eventuali guasti.

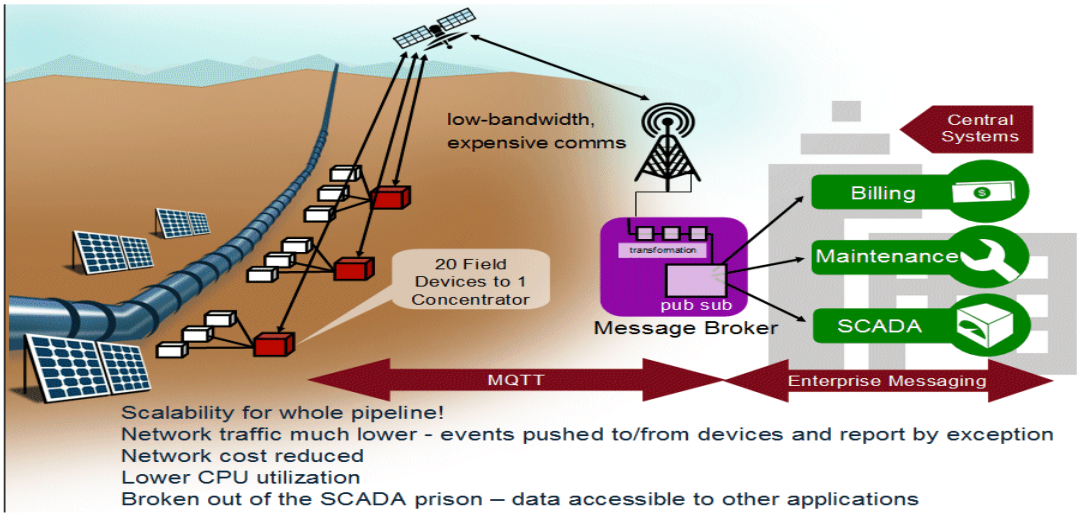


Figura 14-Sicurezza e Manutenzione Predittiva di un Condotto [14.a]

L’unico punto a sfavore dell’IoT è il fatto che vivere in un mondo completamente connesso può rappresentare una minaccia per la nostra privacy ed aumentare la presenza di attacchi sgradevoli da parte di terzi. Speriamo che vengano apportate soluzioni a questi problemi il prima possibile.

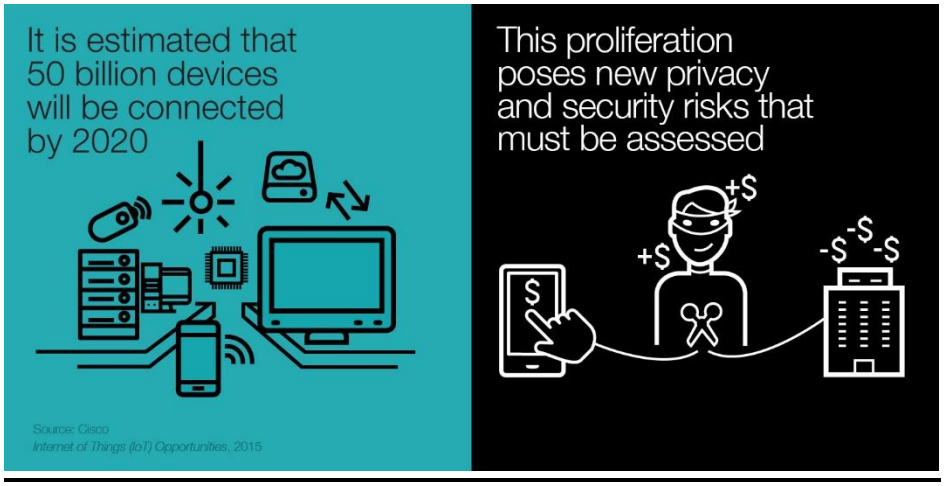


Figura 15-Prevenzione Attacchi Informatici [15.a]

2-Obiettivi della Tesi

Dopo questa presentazione che si occupa di fornire uno sguardo di insieme sul mondo dell'Internet delle Cose, possiamo finalmente addentrarci nella parte più tecnica e pratica della questione. Affinché avvenga una comunicazione tra dispositivi, questi dovranno essere dotati di opportune componenti hardware e implementazioni software per portare a termine una corretta trasmissione, ogni singolo "ingranaggio" volto al funzionamento di tutto il sistema necessita di una progettazione e di uno studio individuale. Questa Tesi si concentra sulla parte riguardante i protocolli di rete, che non sono altro che le regole e le convenzioni rispettate durante la trasmissione, al fine di ottenere un corretto dialogo tra dispositivi, in particolare sui protocolli dello strato applicazione (questo concetto sarà ripreso nel dettaglio nel capitolo successivo). In un mondo dove l'invio di dati è soggetto ad una spesa, non solo a livello di informazione, ma soprattutto a livello monetario, lo scopo del nostro lavoro è quello di occuparci di un determinato protocollo, chiamato MQTT, studiare le parti di una comunicazione tra dispositivi che lo implementano e verificare se veramente può farci risparmiare bit rispetto ad altri protocolli concorrenti. Per prima cosa, nel prossimo capitolo chiariremo meglio il compito dei protocolli di rete e spiegheremo perché il protocollo MQTT risulta molto sfruttato in campo IoT. Una volta fatto ciò sarà necessario studiare la struttura del protocollo stesso, grazie al documento che riporta lo standard MQTT, ci sarà possibile identificare la composizione dei vari messaggi MQTT e riconoscerli in una eventuale trasmissione (Capitolo 4). Nel Capitolo 5 effettueremo una comunicazione MQTT tra un PC e un tablet e con l'uso di un analizzatore di protocollo, un programma il cui lavoro è quello di controllare il traffico dati sulle porte di rete del nostro computer, potremmo notare la realizzazione pratica delle conoscenze acquisite nei capitoli precedenti e comprendere meglio le varie fasi e regole di una trasmissione MQTT. Proseguendo, nel Capitolo 6, al solo scopo di ricerca, sono state analizzate alcune piattaforme preconfigurate nate per la gestione di dispositivi IoT e ne sono state riportate le varie caratteristiche. Infine, nel Capitolo 7, si passerà al confronto, grazie

sempre all'uso di un analizzatore di rete, tra una trasmissione con protocollo MQTT e una trasmissione con protocollo HTTP, usato con il metodo di richiesta POST, la scelta del protocollo concorrente è data dal fatto che anche HTTP-Post è una soluzione molto usata nell'Internet of Things. Nell'ultimo capitolo riporteremo le conclusioni e faremo il punto della situazione riguardo al lavoro svolto in questo documento, sperando di confermare i risultati attesi.

3-Perchè il Protocollo MQTT

Ora che abbiamo una conoscenza di tutto ciò che riguarda l'Internet of Things vediamo di spiegare perché viene data tanta importanza al Protocollo MQTT. Come detto nel capitolo precedente ognuno dei nostri dispositivi sarà connesso ad internet, il quale ha una struttura cosiddetta a strati, o livelli, necessaria per una corretta comunicazione.

Figura 16-Architettura a Strati di Internet

Il livello di un device è in grado di comunicare con i livelli omologhi di altri devices grazie ai protocolli di rete, che stabiliscono le regole e le convenzioni del dialogo (come enunciato nel capitolo precedente). Uno dei protocolli più usati per le comunicazioni allo strato Applicazione è il famosissimo HTTP (HyperText Transfer Protocol) perfetto per lo scambio di informazioni sul web visto che è implementato su un'architettura client/server. Ad esempio quando vogliamo consultare una pagina web, il client esegue una richiesta, il server restituisce una risposta, il funzionamento del tutto è accompagnato da un consistente scambio di informazioni tra il client ed il server.
Nel mondo nell'Internet of Things però, dove sono i dispositivi a comunicare tra loro, il protocollo http inizia a mostrare le prime debolezze, per il fatto che

lo scambio di dati tra due macchine è in genere composto da un limitato numero di bit e quindi si rischia che le informazioni necessarie per creare una connessione siano maggiori del messaggio stesso; in più non è possibile stabilire una comunicazione da one-to-many, ovvero da uno a molti.
Entra quindi in gioco il Protocollo MQTT (Message Queue Telemetry Transport), pensato circa 15 anni fa, per le comunicazioni M2M, ovvero Machine to Machine, ha iniziato a guadagnare popolarità solamente ora per via dell'esplosione del mercato IoT. Rappresenta un validissimo sostituto del Protocollo HTTP in applicazioni M2M, visto che in genere, per questo tipo di comunicazioni si utilizza il metodo di richiesta POST del Protocollo HTTP. Un grande vantaggio è dato dal fatto che il Protocollo MQTT, al posto del modello client/server dell'HTTP, adotta un meccanismo di pubblicazione e sottoscrizione (publish/subscribe) e scambia i messaggi tramite un apposito Broker o Hub. Ogni device si sottoscrive appunto a determinati argomenti detti topics, i mittenti invece pubblicano, quindi inviano, il contenuto dei messaggi accompagnati da un topic, il broker, che si trova fisicamente al centro della comunicazione, si occupa di consegnare il messaggio solamente ai devices sottoscritti al topic in questione.
Grazie a questo meccanismo una comunicazione one-to-many risulta semplicissima.

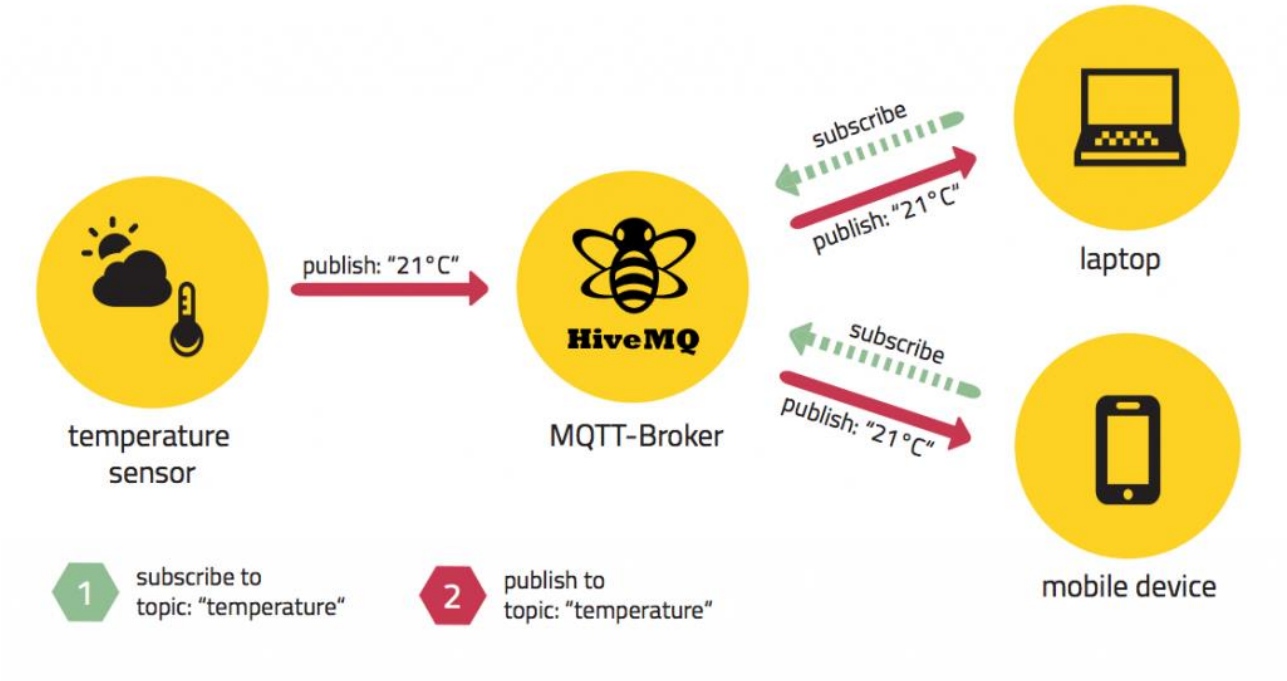


Figura 17-Meccanismo Publish-Subscribe [16.a]

L'altro punto di forza del protocollo MQTT sta nel fatto che è composto da diversi tipi di messaggi, i cosiddetti Type Message. Ognuno di essi sarà dedicato ad un tipo di operazione diversa, come ad esempio l'instaurarsi di una connessione, il subscribe, il publish o la disconnessione, possiamo quindi risparmiare dei bit inutili e inviare sulla rete solo quello che ci fa comodo e a chi vogliamo noi. È anche possibile all'interno dei tipi di messaggio selezionare la qualità, e quindi il numero dei rinvii, del dato che vogliamo trasmettere.
L'unica piccolissima difficoltà sta nel imparare la funzione e la struttura di ogni diverso Type Message per analizzare in dettaglio una comunicazione con il protocollo MQTT.
Il prossimo capitolo si occupa appunto della struttura dei vari Type Message.

4-Struttura dei messaggi MQTT

Le immagini ed i contenuti di questo capitolo sono stati presi dalla documentazione dello Standard MQTT v3.1.1 [2.b]

La forma generale dei pacchetti è data da:

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

Figura 18-Struttura Generale Type Message

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							

Figura 19-Struttura Fixed Header

MQTT Control Packet indica con un valore da 0 a 15 il tipo di pacchetto che stiamo usando:

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Figura 20-Valori Control Packet

I bit di flag invece, come indicato in figura, variano a seconda del pacchetto che stiamo usando

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP[1]	QoS[2]	QoS[2]	RETAIN[3]
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

Figura 21-Valori Bit di Flag

Dal secondo byte in poi il Fixed Header indica la Remaining Length, ovvero la lunghezza in Byte che rimane del pacchetto, inclusi i dati del Variable Header e del Payload (la Remaining Length non include se stessa). È codificata usando uno schema a lunghezza variabile che sfrutta un byte per valori fino a 127. I valori maggiori sono così codificati: i 7 bit meno significativi di ogni byte codificano il valore, mentre il bit più significativo segnala se ci sono byte seguenti. Quindi ogni singolo byte codifica 128 valori più un bit di continuità. Il massimo numero di byte nella Remaining Length è 4.

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

Figura 22-Gestione dei Valori Remaining Length

VARIABLE HEADER:

Il contenuto del Variable Header varia a seconda del tipo di pacchetto che stiamo usando, in seguito lo analizzeremo per ogni singolo caso. Nei pacchetti PUBLISH (se QoS>0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE e UNSUBACK nel Variable Header risiede un Numero di Identificazione dei Pacchetti (Packet Identifier).

Nei casi di SUBSCRIBE, UNSUBSCRIBE e PUBLISH (se QoS>0), ogni volta che il Client, invia uno di questi tipi di pacchetto deve assegnargli un numero non usato di Packet Identifier (esclusivamente diverso da zero). Se ri-invia un particolare tipo di pacchetto, il Packet Identifier assegnato potrà essere riutilizzato solamente quando il Client riceverà da parte del Server una conferma di avvenuta ricezione del pacchetto, altrimenti è costretto ad usare il numero di identificazione seguente. Lo stesso vale se il primo ad inviare è il Server.

I pacchetti PUBACK, PUBREC e PUBREL devono contenere il Packet Identifier relativo al PUBLISH ad esso associato. Lo stesso vale SUBACK e UNSUBACK. Questo concetto sarà chiarito meglio in seguito, quando parleremo di Qualità del Servizio.

Il Server ed il Client assegnano Packet Identifier indipendentemente l'uno dall'altro, può succedere che lo stesso numero viaggi in due comunicazioni diverse

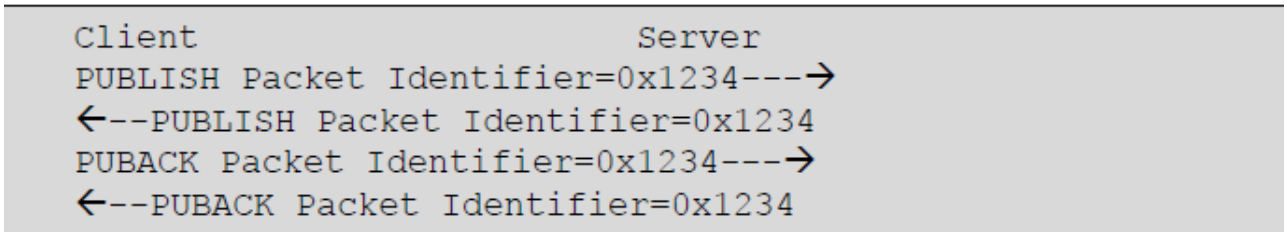


Figura 23-Esempio di Assegnazione dello stesso Packet Identifier in due comunicazioni diverse

PAYLOAD:

il Payload è la parte finale di un pacchetto ed è presente solo nei CONNECT, SUBSCRIBE, SUBACK e UNSUBSCRIBE. Nei PUBLISH è l'Application Message ed è facoltativo.

TIPI DI PACCHETTO MQTT (CONTROL PACKET TYPES)

CONNECT:

appena viene stabilita la connessione del Network il primo messaggio inviato dal Client verso il Server deve essere di tipo CONNECT. Il Client può inviare il messaggio CONNECT sulla rete esclusivamente una volta, il Server interpreta un secondo CONNECT come una violazione di protocollo e disconnette il Client.

La Fixed Header del CONNECT ha la classica forma:

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (1)				Reserved			
	0	0	0	1	0	0	0	0
byte 2…	Remaining Length							

Figura 24-Fixed Header Connect

La Remaining Length contiene il valore 10 Byte dovuto al Variable Header più il valore in Byte del Payload

La Variable Header contiene questi campi:

Protocol Name-6 Byte, indica il nome del protocollo, MQTT scritto in UTF-8

Protocol Level-1 Byte, indica il livello di revisione del protocollo, nel caso del 3.1.1 è 4

Connect Flags-1 Byte, contiene un numero di parametri relativi al comportamento della connessione e alla
presenza di alcuni campi del Payload

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

Figura 25-Parametri Connect Flags

Clean Session = se 0 il Server deve riprendere la Sessione con il Client associato (grazie al Client Identifier). Se la Sessione non esiste deve crearne una. Dopo essersi disconnessi, il Client e il Server, devono salvare la Sessione. Dopo la disconnessione di una Sessione il Server deve salvare prima i messaggi con qualità QoS1 e Qos2 che corrispondono ad una sottoscrizione che il Client aveva al momento della chiusura. Potrebbe succede che salvi anche i QoS0 che appartengono allo stesso criterio

se 1 il Client e il Server devono abbandonare Sessione precedente e iniziarne una nuova

Will Flag = 1 se viene accettata la richiesta di connessione allora il Will Message deve essere salvato sul Server e associato alla connessione del network. Il Will Message deve essere pubblicato quando la connessione del network è successivamente chiusa, salvo che il Will Message sia stato cancellato dal Server dopo aver ricevuto un pacchetto DISCONNECT. Will QoS e Will Retain saranno usati dal Server ed i campi Will Topic e Will Message sono presenti nel Payload

Se 0 Will QoS e Will Retain devono valere 0 e Will Topic e Will Message non sono presenti nel Payload. Il Will Message non sarà pubblicato alla fine della connessione. Il Will QoS potrà valere solo 0

Will QoS = se Will Flag vale 1 può essere 0 1 2 (no 3)

Will Retain = se 0 il Server deve pubblicare Will Message come messaggio non mantenuto, se 1 il Will Message è mantenuto

Password Flag = se 0 la Password non è presente nel Payload, se 1 è presente

User Name Flag = se 0 l'User Name non è presente nel Payload e Password Flag deve essere 0. Se 1 nel Payload è presente il campo User Name

Keep Alive-2 Byte, è il tempo espresso in secondi che intercorre tra il momento in cui il Client finisce la trasmissione di un Control Packet e l'inizio dell'invio del prossimo

Payload:

contiene uno o più campi prefissati, la loro presenza è legata al valore dei flag nel Variable Header. I campi, in ordine di apparizione sono:

Client Identifier, ha la dimensione di una stringa codificata UTF-8, identifica il Client al Server. Ogni Client connesso ha un unico Identifier

Will Topic, presente se Will Flag è impostato a 1, è grande come una stringa UTF-8

Will Message, presente se il Will Flag vale 1, definisce il messaggio di applicazione (Application Message) pubblicato dal Will Topic. La sua dimensione è data da 2 Byte, dove è indicata la lunghezza del messaggio che segue, sommati ai Byte del messaggio vero e proprio

User Name, abilitato da User Name Flag=1 è una stringa UTF-8 usata dal Server per l'autenticazione e la autorizzazione

Password, collegata all'User Name occupa 2 Byte, in cui è espressa la lunghezza della password, sommati ai Byte della password stessa

CONNACK:

il CONNACK è il pacchetto inviato dal Server in risposta al pacchetto CONNECT. Il primo pacchetto inviato dal Server DEVE essere un CONNACK.

Fixed Header:

la Fixed Header si presenta nella forma classica con una Remaining Length di valore 2.

Variable Header:

SP Session Present = se il Server accetta una connessione con Clean Session settato a 1, il Server deve impostare a 0 sia SP che il Return Code.

Se la connessione accettata ha Clean Session uguale a 0 allora il valore di SP dipende dal fatto che il Server abbia già salvato una Sessione di un Client ID già presente, se la Sessione era stata già stata salvata allora SP sarà 1, altrimenti 0. Il Return Code sarà sempre 0.

Return Code = nel caso di una connessione accettata avrà valore 0 altrimenti assumerà i valori come indicato nella tabella seguente

	Description	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Connect Acknowledge Flags		Reserved							SP[1]
byte 1		0	0	0	0	0	0	0	X

Connect Return code									
byte 2		X	X	X	X	X	X	X	X

Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect
6-255		Reserved for future use

Figura 26-Struttura Variable Header Connack e Return Code

Payload:

il campo Payload non è presente in questo tipo di pacchetto

PUBLISH:

questo è il pacchetto che trasporta l' Application Message

Fixed Header:

nella Fixed Header assumono molta importanza i bit 3 2 1 0 del primo Byte, la Remaining Length non ha un valore fissato, ma dipende dal contenuto del

pacchetto

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

Figura 27-Fixed Header Publish

DUP Duplicate Delivery of a PUBLISH Control Packet = se 0 significa che è la prima occasione si prova ad inviare un messaggio di questo tipo, altrimenti vale 1

Qos Level = indica il livello di qualità per il trasporto dell'Application Message

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

Figura 28-Bit Livelli di Qualità

Retain = se uguale ad 1 il Server (o il Client se il PUBLISH è inviato dal Server) deve salvare l'Application Message e la sua Qualità in modo che possano essere trasmessi ai futuri subscribers. Se il Server invia un pacchetto di questo tipo ad un Client, Retain deve valere 1 se il messaggio è inviato come risultato di una nuova sottoscrizione fatta dal Client, mentre deve valere 0 se corrisponde ad una sottoscrizione già stabilita

Variable Header:

contiene i seguenti campi, Topic Name e Packet Identifier. Il Topic Name indica l'argomento che riguarda i dati del Payload, mentre il Packet ID è presente solo in pacchetti con QoS diverso da 0, visto che questi casi comportano un ritorno di dati verso chi ha inviato il pacchetto. Nella tabella è indicato un esempio di Variable Header del PUBLISH

	Description	7	6	5	4	3	2	1	0
Topic Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Packet Identifier									
byte 6	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 7	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0

Figura 29-Esempio di Variable Header Publish

Payload:

è qui che risiede l'Application Message. Non è considerata una violazione di protocollo inviare un PUBLISH con Payload contenente zero Byte

PUBACK:

questo pacchetto è la risposta che ci si aspetta da un da un PUBLISH con QoS=1 (consegnato almeno una volta).

Fixed Header:

identico alla forma classica con Remaining Length di valore 2

Variable Header:

è di 2 Byte e contiene il Packet Identifier del pacchetto di cui dobbiamo confermare l'arrivo

Payload:

non presente

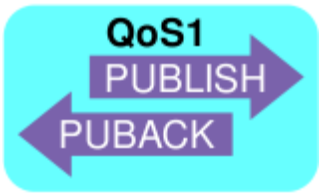


Figura 30-Esempio di Comunicazione Client-Server con QoS1

PUBREC Publish Received

PUBREL Publish Release

PUBCOMP Publish Complete:

questi tre pacchetti compongono insieme al PUBLISH un scambio completo di Qos2. Subito dopo la ricezione di un PUBLISH il destinatario invia un PUBREC

contenete il Packet Identifier del pacchetto interessato, che quando viene ricevuto abilita il mittente all'invio di un PUBREC. Lo scambio si conclude con un PUBCOMP ricevuto dal mittente.

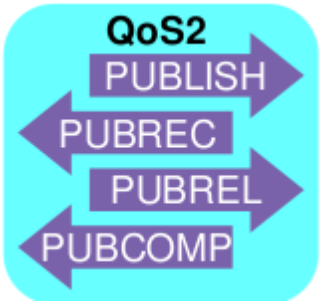


Figura 31-Esempio di Comunicazione Client-Server con Qos2

Fixed Header:

il primo Byte cambia con il tipo di pacchetto, la Remaining Length è sempre uguale a 2

Variable Header:

composta da 2 Byte contenenti il Packet Identifier del pacchetto in questione

Payload:

non presente

SUBSCRIBE:

inviato dal Client al Server per creare una o più sottoscrizioni. Ogni sottoscrizione registra l'interesse di un Client verso uno o più Topics. Il Server invierà i pacchetti PUBLISH ad ogni Client che corrisponde alle sottoscrizioni. Nel SUBSCRIBE è anche specificata la qualità massima con cui il Server deve inviare gli Application Messages al Client.

Fixed Header:

relativo al pacchetto con una Remaining Length di 2 Byte, per la Variable Header, sommati alla lunghezza del Payload

Variable Header:

contiene il Packet Identifier

Payload:

qui si trovano una lista di filtri Topic (codificati in UTF-8) che indicano i vari Topics che il Client vuole sottoscrivere e un byte che indica la Qualità con cui farlo. Deve contenere almeno un filtro, altrimenti un Payload vuoto è visto come una violazione di protocollo.

Qui sotto è indicato l'esempio della sottoscrizione dei seguenti Topics

Topic Name	"a/b"
Requested QoS	0x01
Topic Name	"c/d"
Requested QoS	0x02

Figura 32-Esempio di Sottoscrizione ai Topics

Ed ecco come viene codificato il Payload

	Description	7	6	5	4	3	2	1	0
Topic Filter									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Requested QoS									
byte 6	Requested QoS(1)	0	0	0	0	0	0	0	1
Topic Filter									
byte 7	Length MSB (0)	0	0	0	0	0	0	0	0
byte 8	Length LSB (3)	0	0	0	0	0	0	1	1
byte 9	'c' (0x63)	0	1	1	0	0	0	1	1
byte 10	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 11	'd' (0x64)	0	1	1	0	0	1	0	0
Requested QoS									
byte 12	Requested QoS(2)	0	0	0	0	0	0	1	0

Figura 33-Esempio di Codifica di un Payload del Pacchetto Subscribe

I bit dal 7 al 2 del Requested QoS Byte sono fissati a 0 perché non usati in questa versione del protocollo e riservati per usi futuri

SUBACK:

Quando il Server riceve un pacchetto SUBSCRIBE deve rispondere con un SUBACK che abbia lo stesso Packet Identifier del SUBSCRIBE di cui voglio conferma. Al Server è permesso l'invio di pacchetti PUBLISH che corrispondono alla sottoscrizione anche prima anche prima di inviare un SUBACK. La ricezione da parte del Server di filtri Topic già sottoscritti comporta il rimpiazzo della vecchia sottoscrizione con la nuova, i filtri Topic saranno identici ma potrà

cambiare al massimo la qualità. Il SUBACK contiene nel Payload un codice di ritorno (Return Code) per ogni filtro presente nel SUBSCRIBE nel quale. Il Server potrebbe garantire una Qualità inferiore a quella richiesta.

Fixed Header:

di formato standard con Remaining Length indicante la lunghezza della Variable Header (2 Byte) più la lunghezza del payload

Variable Header:

contiene il Packet Identifier del SUBSCRIBE che voglio confermare

Payload:

contiene una lista di Return Codes che corrispondono alla conferma dei filtri Topics. L'ordine dei Return Codes deve rispettare quello dei Topic Filters nel pacchetto SUBSCRIBE. I codici oltre a confermare il successo dei filtri indicano con quale Qualità sono stati accettati

Bit	7	6	5	4	3	2	1	0
	Return Code							
byte 1	X	0	0	0	0	0	X	X

Allowed return codes:
0x00 - Success - Maximum QoS 0
0x01 - Success - Maximum QoS 1
0x02 - Success - Maximum QoS 2
0x80 - Failure

Figura 34-Payload Suback

I codici diversi da questi sono riservati e NON DEVONO essere usati

UNSUBSCRIBE:

inviato solo dal Client quando vuole cancellare un Topic. Duale al SUBSCRIBE, ne presenta la stessa forma cambiando soltanto:

- il primo Byte della Fixed Header
- i Byte Requested QoS nel Payload non sono presenti

UNSUBACK:

di conseguenza duale al SUBACK, indica la conferma di un UNSUBSCRIBE. Se il Server cancella una Sottoscrizione, deve fermare i nuovi messaggi di consegna al Client e completare la consegna di messaggi Qos>0 che aveva già iniziato a inviare. Il Server deve rispondere con UNSUBACK anche ad un UNSUBSCRIBE che non cancella nessun Topic. Al contrario del SUBACK presenta un Payload vuoto.

PINGREQ Ping Request:

è inviato dal Client al Server e può essere usato per:

-avvertire il Server che il Client è vivo, quando non è necessario l'invio di altri pacchetti
-richiedere conferma che il Server è vivo
-verificare che la connessione di rete sia viva

Fixed Header:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 1	MQTT Control Packet type (12)				Reserved			
	1	1	0	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

Figura 35-Fixed Header Pingreq

Variable Header e Payload:

non presenti

PINGRESP Ping Response:

inviato dal Server al Client in risposta ad un PINREQ, indica che il Server è vivo

Fixed Header:

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (13)				Reserved			
	1	1	0	1	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

Figura 36-Fixed Header Pingresp

Variable Header e Payload:

non presenti

DISCONNECT:

è il Control Packet finale inviato dal Client al Server. Indica che il Client si è disconnesso correttamente. Dopo il suo invio il Client deve:

-chiudere la Network Connection
-non inviare più pacchetti su quella Network Connection

Il Server deve:

-eliminare ogni Will Message associato a quella connessione senza pubblicarlo (vedi parte relativa al Will Flag)

Fixed Header:

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (14)				Reserved			
	1	1	1	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

Figura 37-Fixed Header Disconnect

Variable Header e Payload:

non presenti

ESEMPI DI FLUSSO DI MESSAGGI PER LIVELLI DI QUALITA'[17.a]:

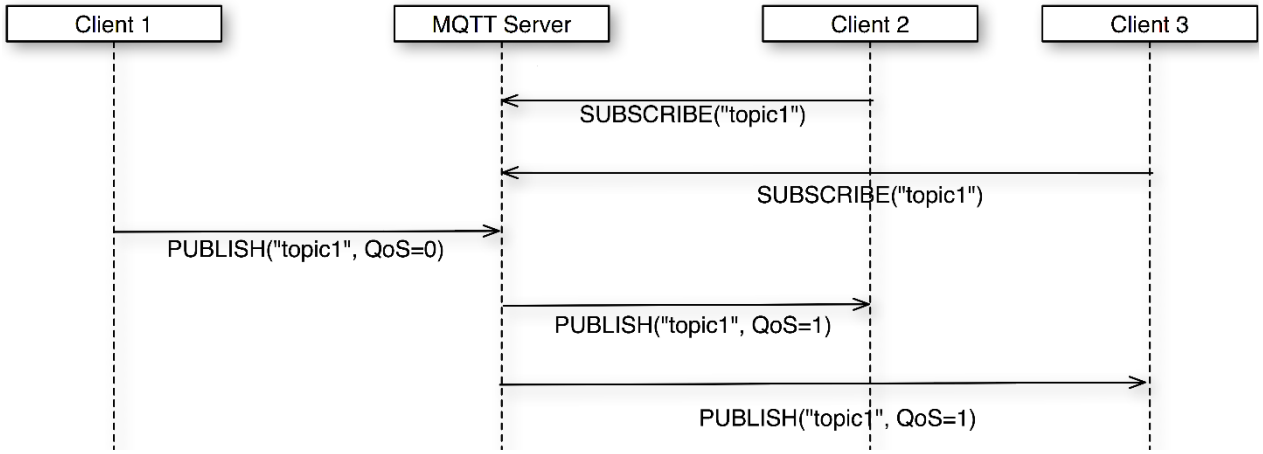


Figura 38-Qos 0 At most once delivery

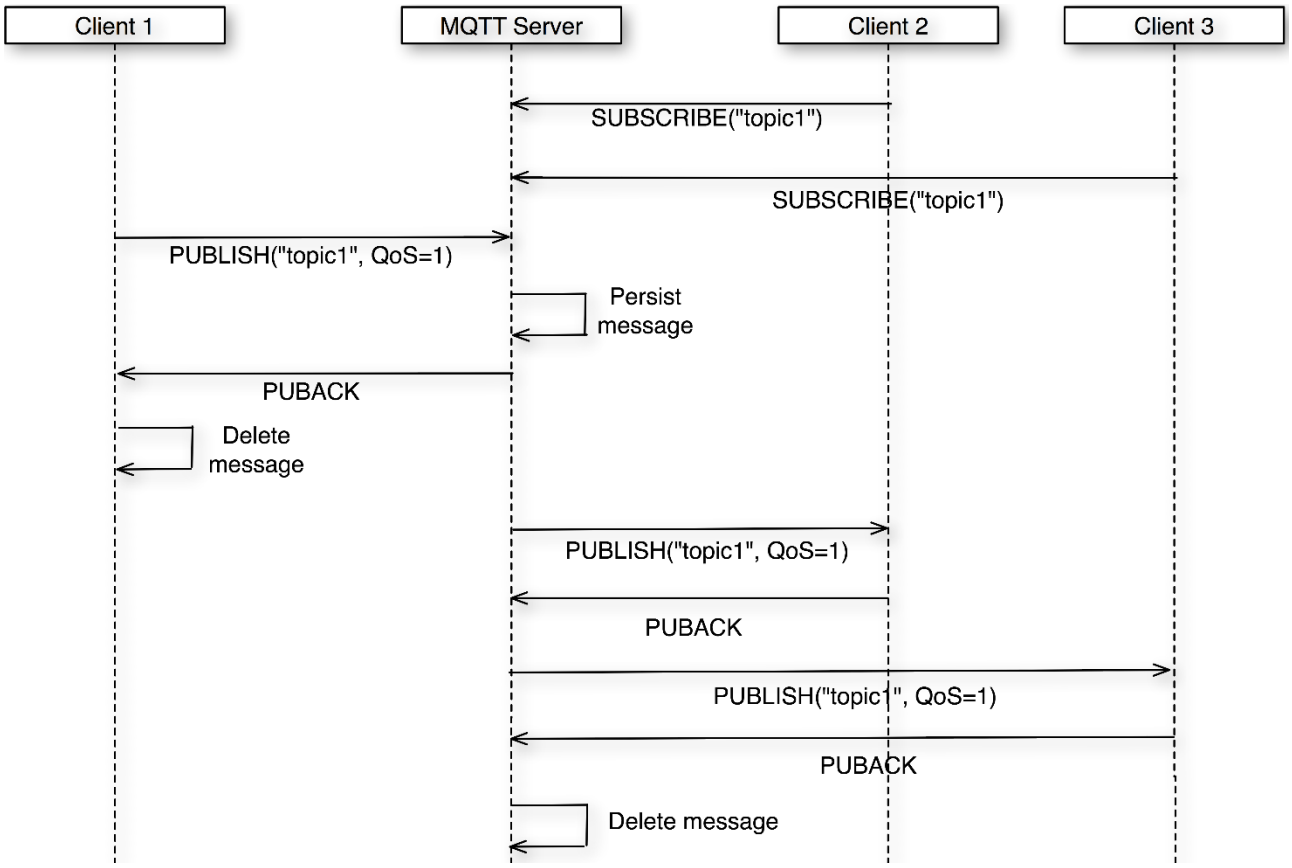


Figura 39-QoS 1 At least once delivery

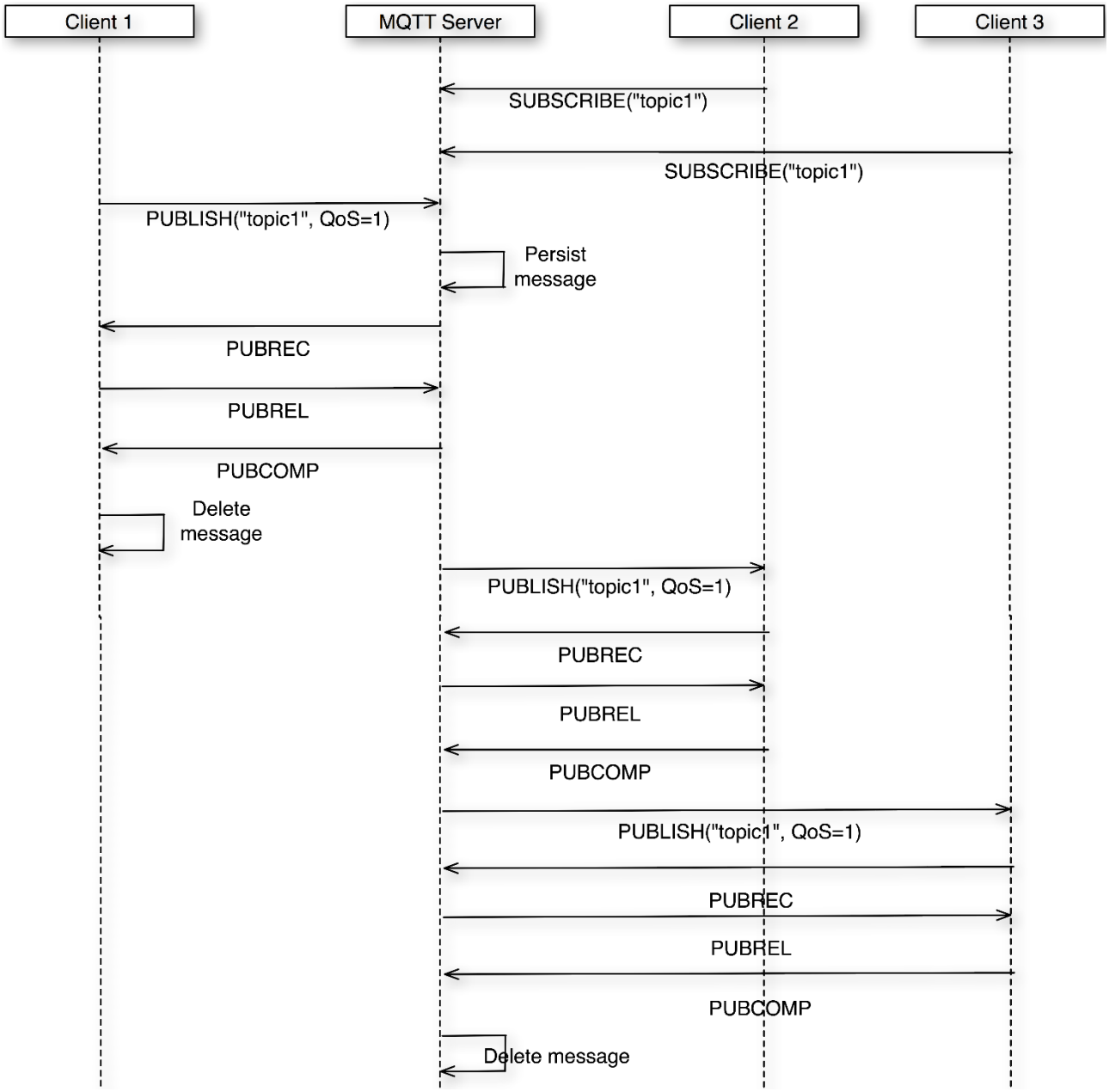


Figura 40-QoS 2 Exactly once delivery

5-Comunicazione tra dispositivi mediante Protocollo MQTT e relativa analisi dei pacchetti

Ora possiamo procedere con la creazione di una comunicazione con protocollo MQTT tra un pc ed un tablet ed analizzarne il contenuto dei pacchetti.
Per procedere ci serve innanzi tutto un broker, fortunatamente sono a nostra disposizione dei broker gratuiti disponibili in rete ed installabili sul nostro pc, dopo alcune prove la nostra scelta è caduta su Mosquitto, un Hub Open Source che implementa la versione v3.1 e v3.1.1 del protocollo MQTT.

Figura 41-Mosquitto Logo [18.a]

Poi dal lato del tablet abbiamo usato l'applicazione gratuita MyMQTT per Android che è in grado di fornirci una console da remoto in cui sottoscrivere topic, pubblicare messaggi e stabilire una connessione con eventuali Broker sia all'interno della nostra rete privata che all'esterno.

Figura 42-MyMQTT App Logo [19.a]

Infine sulla nostra macchina è necessario un analizzatore di protocollo, trattasi di un software in grado di leggere tutto il traffico dati passante in entrambe le direzioni sulla rete del nostro computer. Abbiamo scelto Wireshark un programma molto potente che ci permette di suddividere il traffico per protocolli diversi e di analizzare la struttura di un pacchetto fino al singolo byte.

Figura 43-Wireshark Logo [20.a]

Adesso che è tutto installato possiamo cominciare. Già all'avvio della nostra macchina Linux, il Broker Mosquitto dovrebbe attivarsi in automatico, per verificare che sia così lanciamo il comando ***netstat -an*** che ci permette di verificare lo stato delle connessioni sul computer. Se vediamo che la porta 1883, che è quella standard per il traffico MQTT, è in stato di LISTEN allora Mosquitto funziona

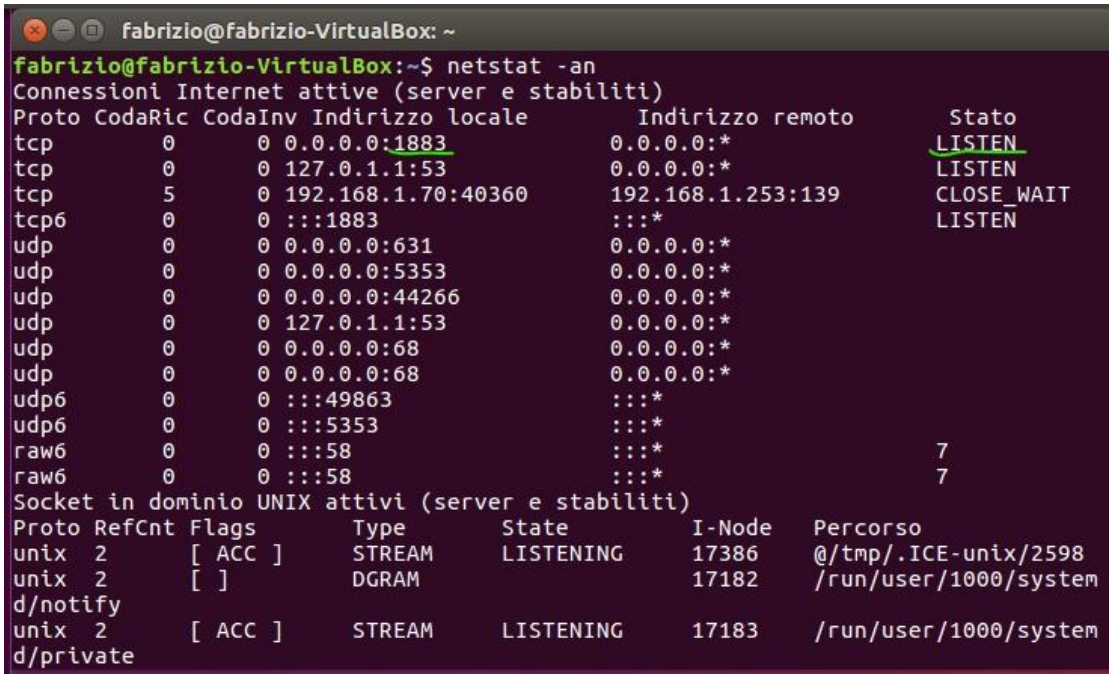


Figura 44-Esecuzione comando netstat -an

Grazie al comando base del broker ***mosquitto_sub*** possiamo sottoscrivere un dispositivo ad un topic, nel nostro caso per evidenziare l'importanza di avare topics diversi creeremo due finestre terminale con due topics differenti in modo da simulare due dispositivi sulla stessa rete ma con diverse sottoscrizioni.

I comandi ***mosquitto_sub -d -t topicsub1*** e ***mosquitto_sub -d -t topicsub2*** indicano rispettivamente la sottoscrizione al ***topicsub1*** e ***topicsub2*** dei due devices. Il topic va inserito dopo ***-t*** mentre ***-d*** indica che vogliamo abilitare il debug dei messaggi

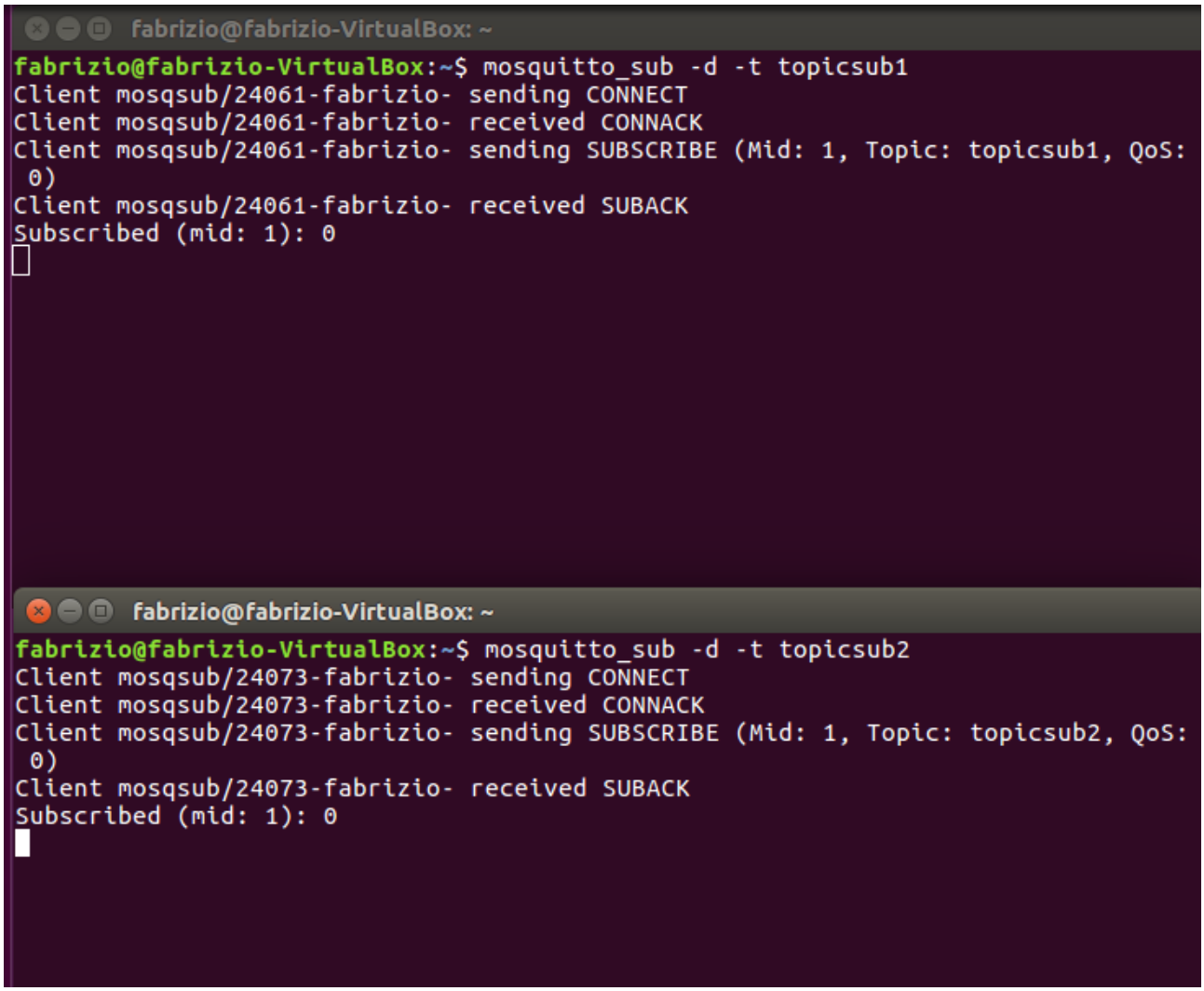


Figura 45-Sottoscrizione da PC a topicsub1 e topicsub2

Dal lato del tablet effettuiamo la connessione al broker inserendo l'indirizzo IP di dove si trova, in questo caso quello del nostro computer

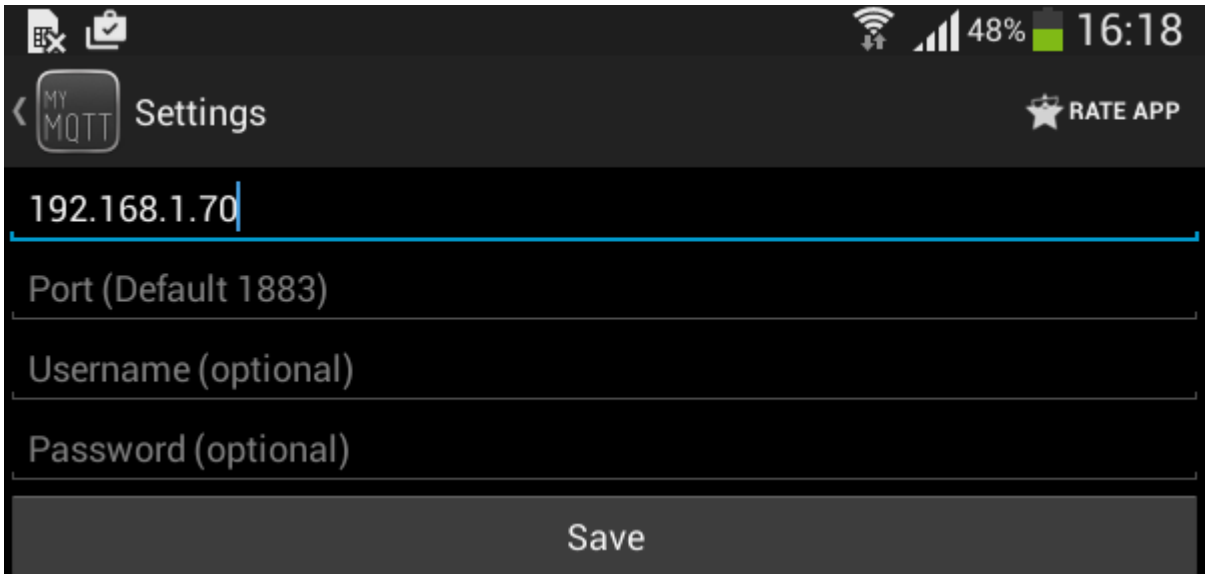


Figura 46-Connessione al Broker da Tablet

A connessione stabilita inviamo dal tablet il messaggio ***primo invio da tablet a pc*** all'argomento ***topicsub1***

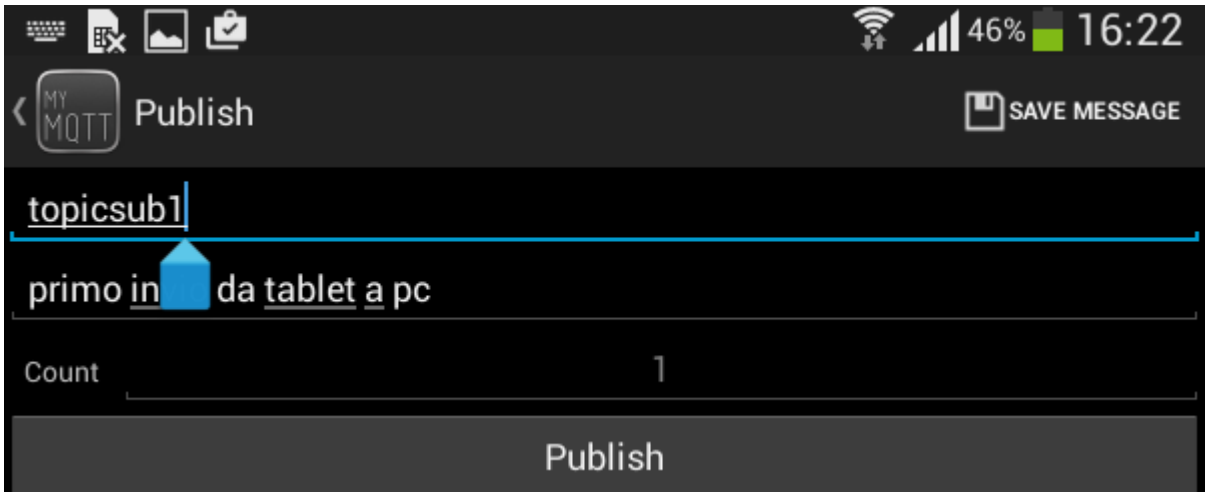


Figura 47-Esecuzione di un Publish dall'Apllicazione

E notiamo che riceverà il messaggio solamente il terminale sottoscritto a ***topicsub1***

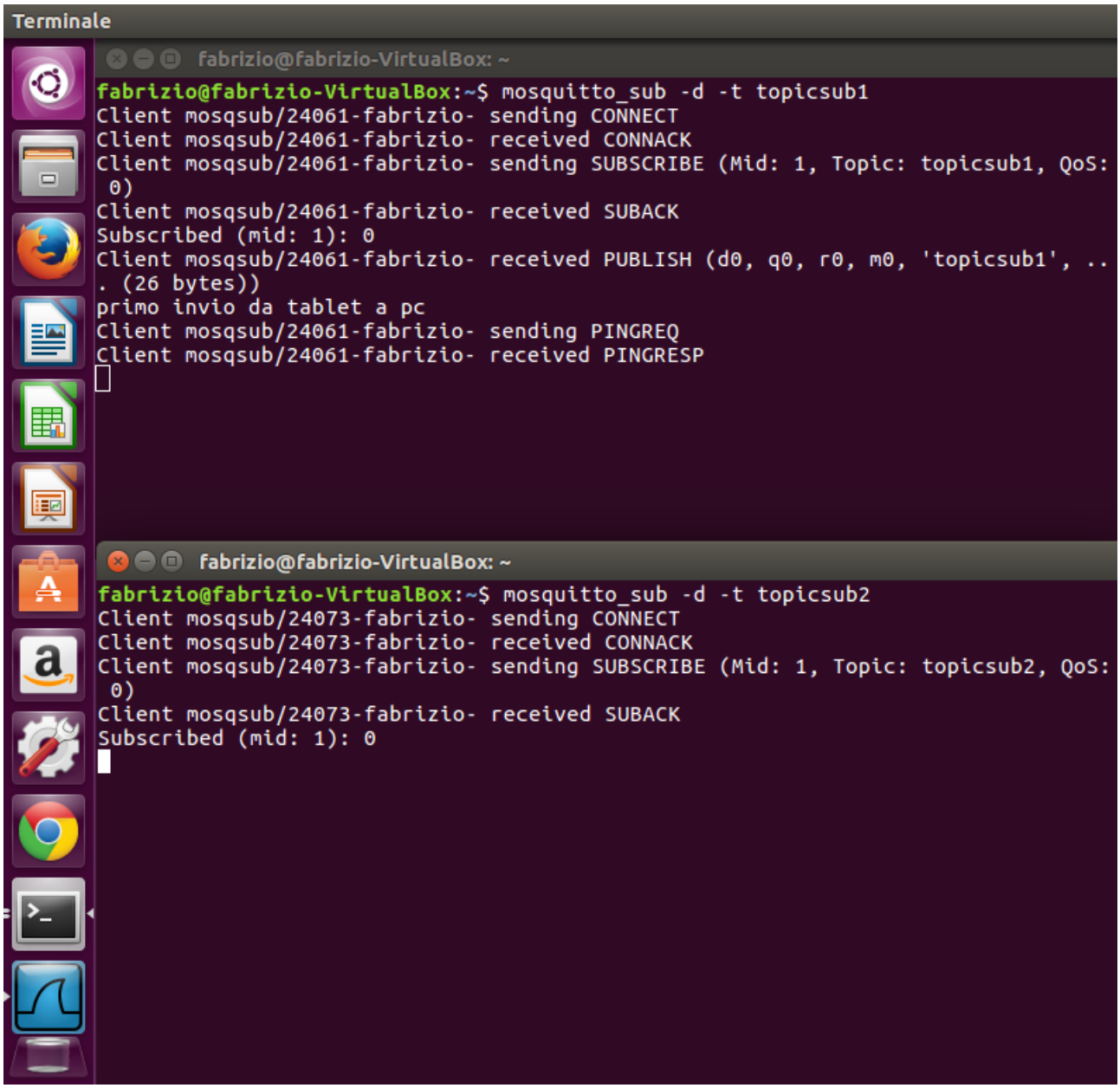


Figura 48-Ricezione di un messaggio inerente a topicsub1

Grazie all'analizzatore di protocollo è possibile vedere che il contenuto del messaggio Publish rispecchia la conformazione relativa allo standard MQTT e soprattutto la zona in cui si trova il messaggio da trasportare e la sua lunghezza in byte

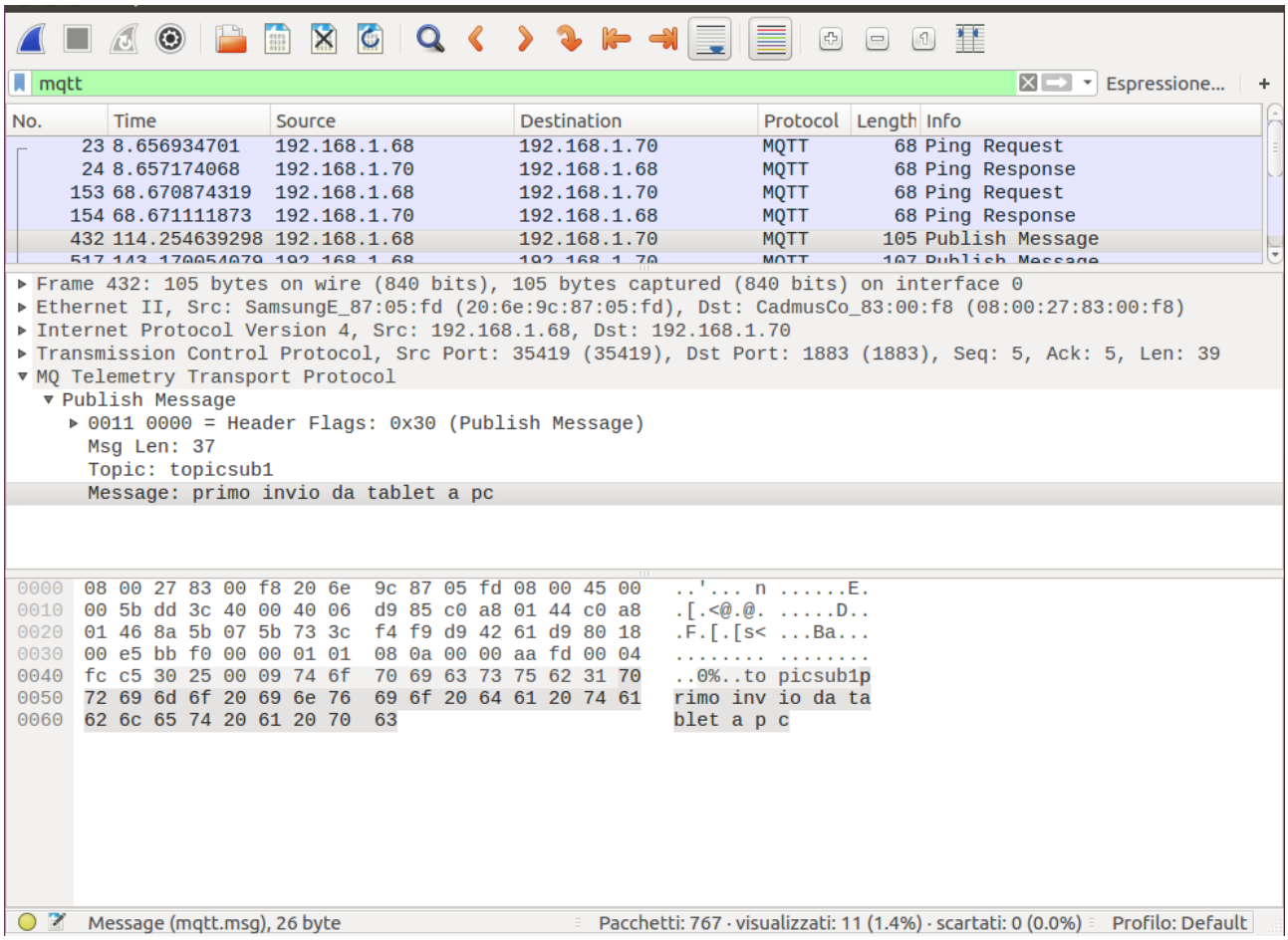


Figura 49-Struttura del primo messaggio Publish analizzata con Wireshark

Ripetiamo la cosa per il secondo topic

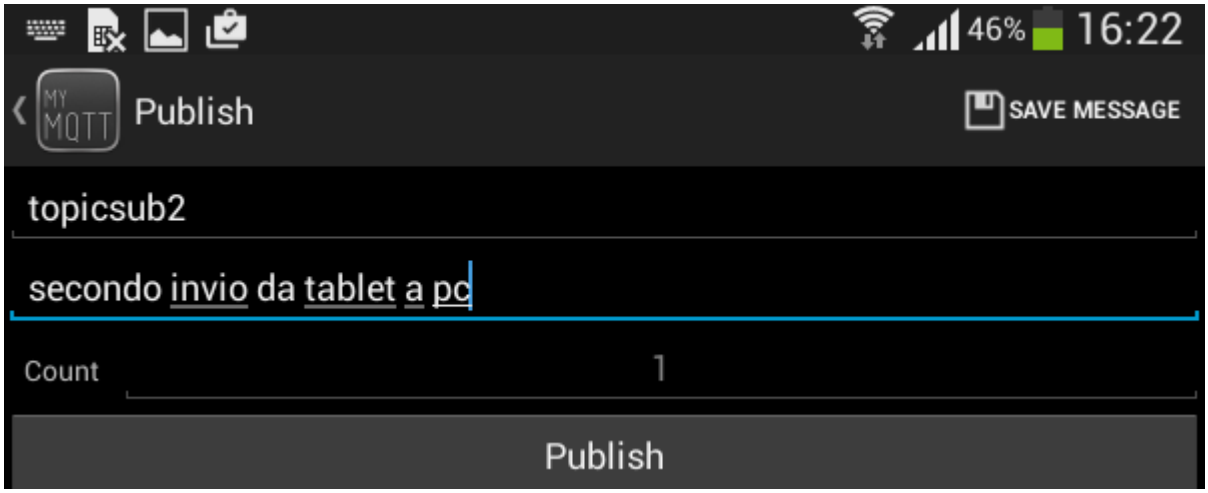


Figura 50-Esecuzione del secondo Publish dall'Applicazione

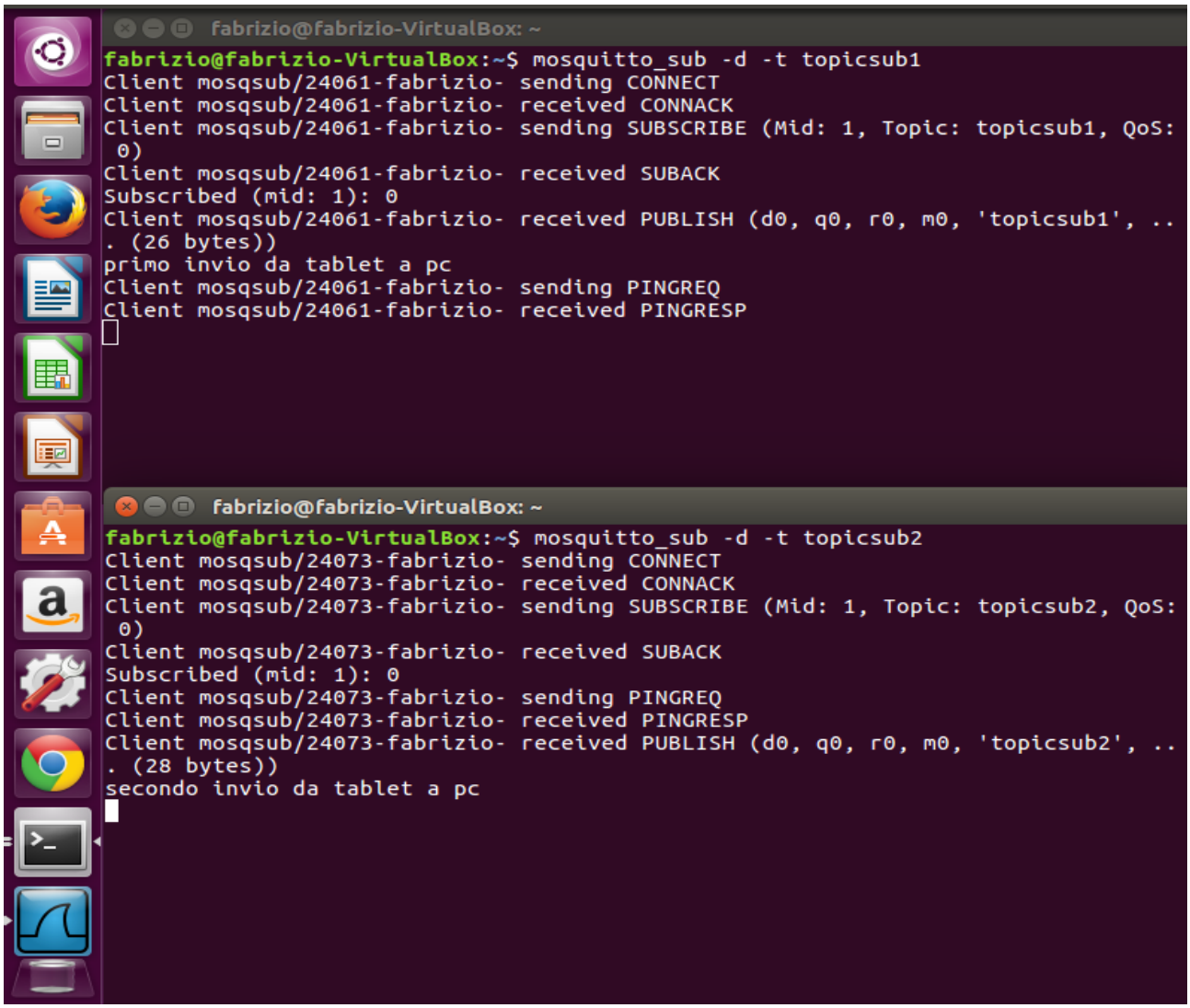


Figura 51-Ricezione del Messaggio inerente a topicsub2

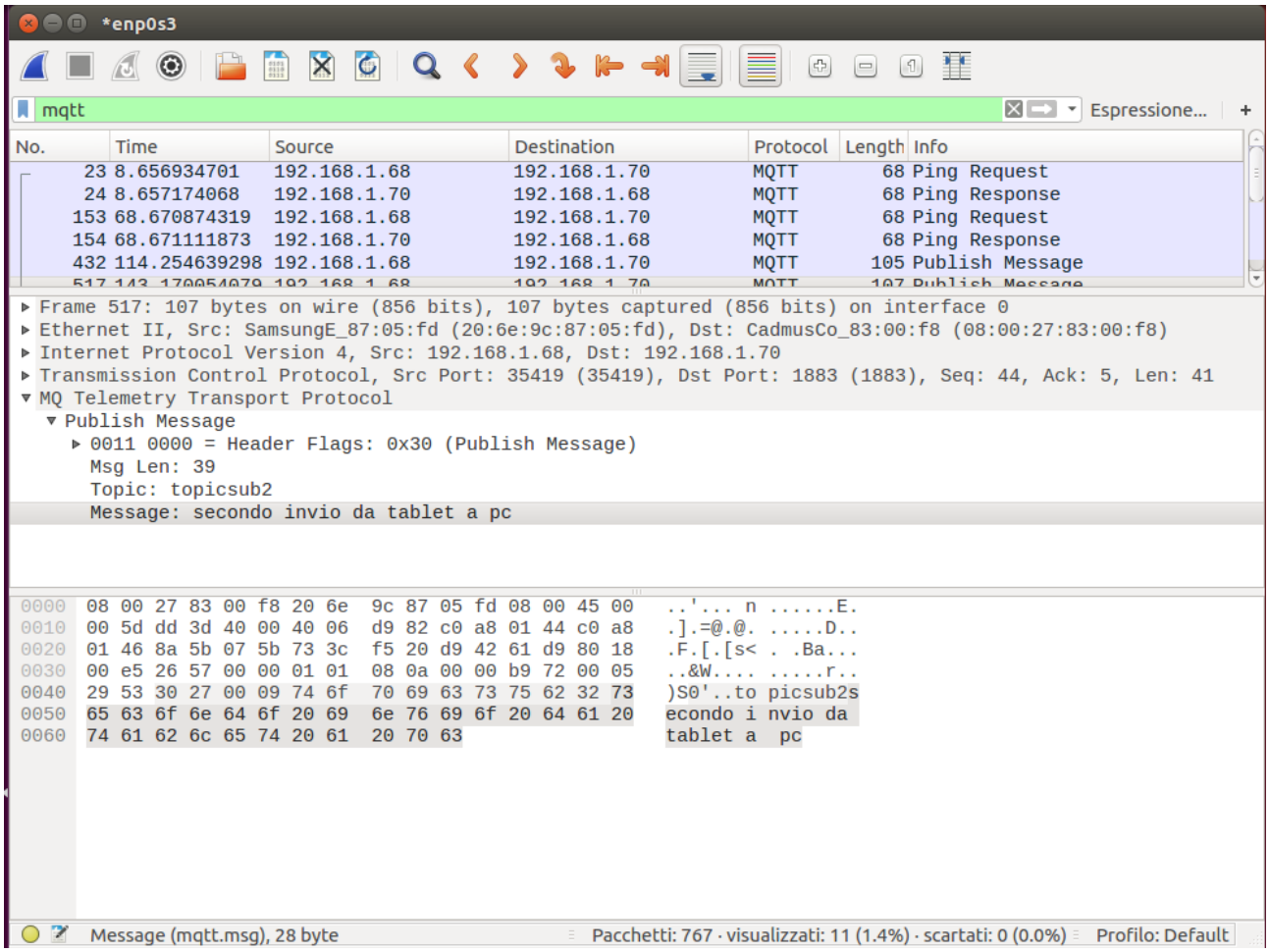


Figura 52-Struttura del secondo messaggio Publish analizzata con Wireshark

Procediamo con la verifica contraria, questa volta sarà il tablet a sottoscriversi a un topic chiamato ***provapub***

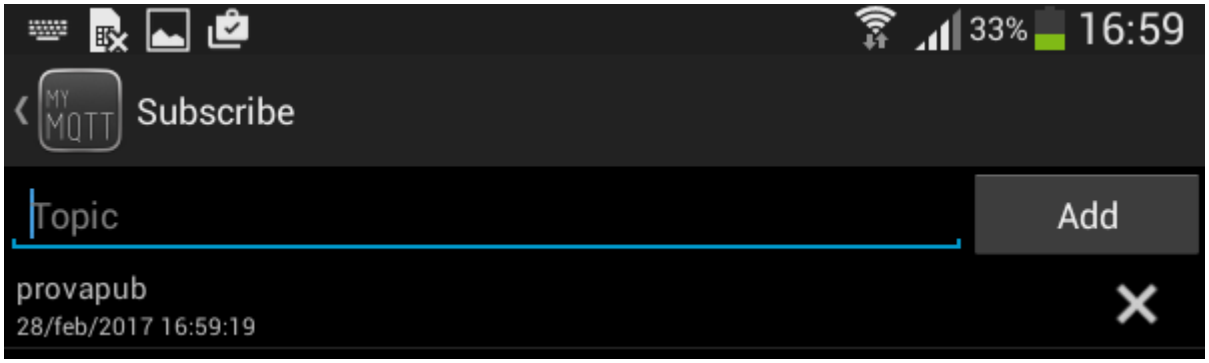


Figura 53-Sottoscrizione a un topic lato tablet

E il pc pubblicherà grazie al comando

mosquitto_pub -d -t provapub -m invio_pc_tablet

il messaggio ***invio_pc_tablet*** al topic ***provapub***

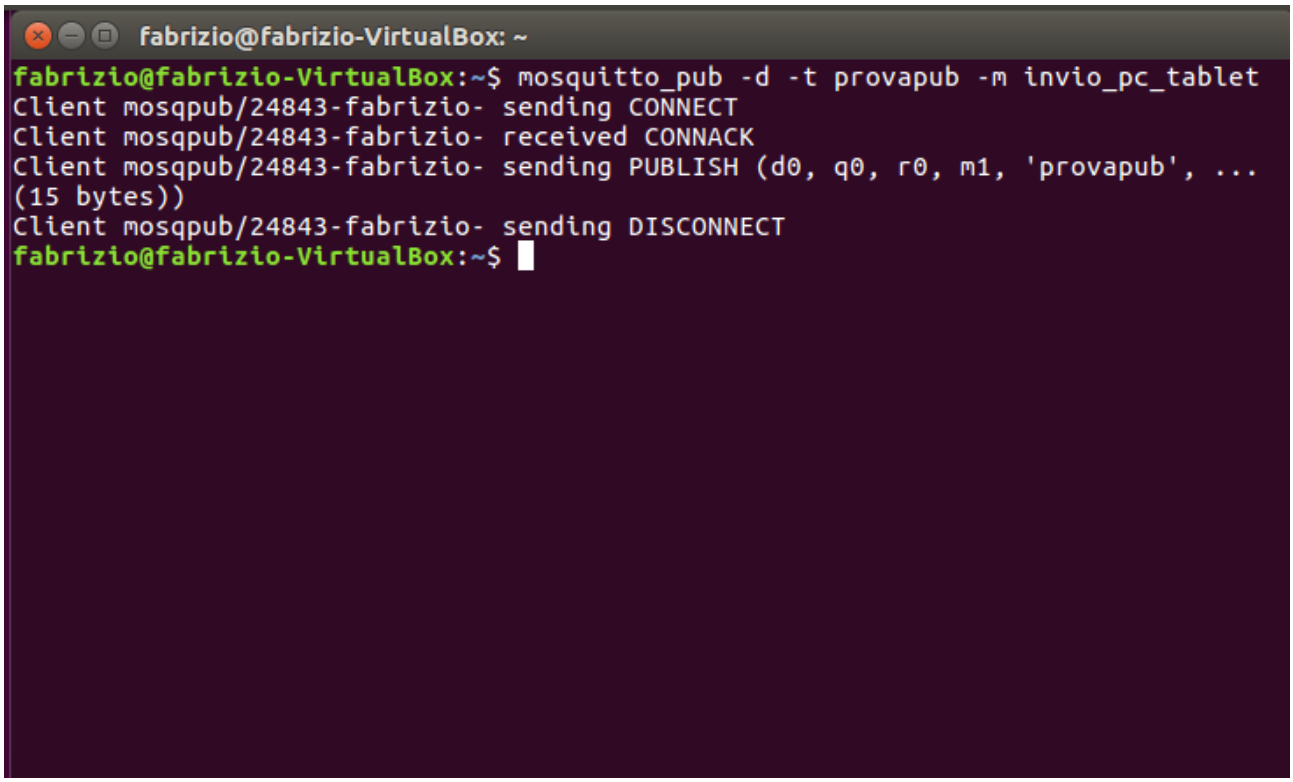


Figura 54-Invio di un Publish da PC a tablet

vediamo che l'applicazione sul tablet ha ricevuto il messaggio

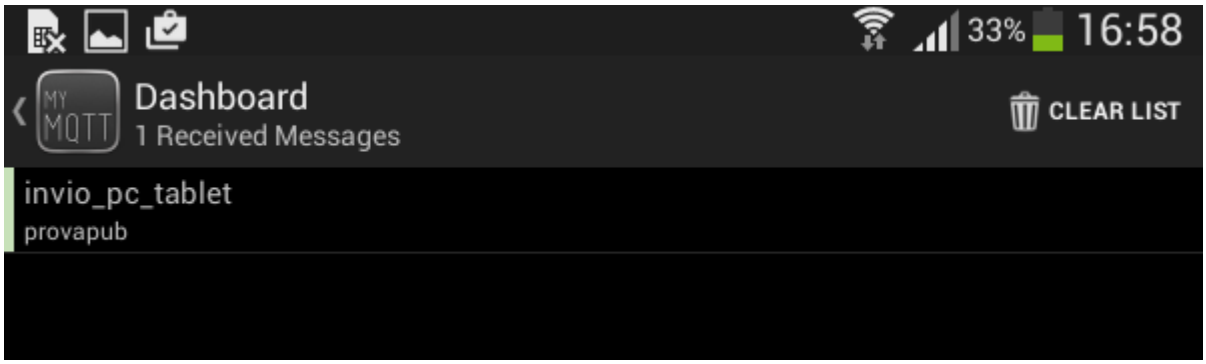


Figura 55-Ricezione del messaggio sul tablet

E relativa analisi dei pacchetti

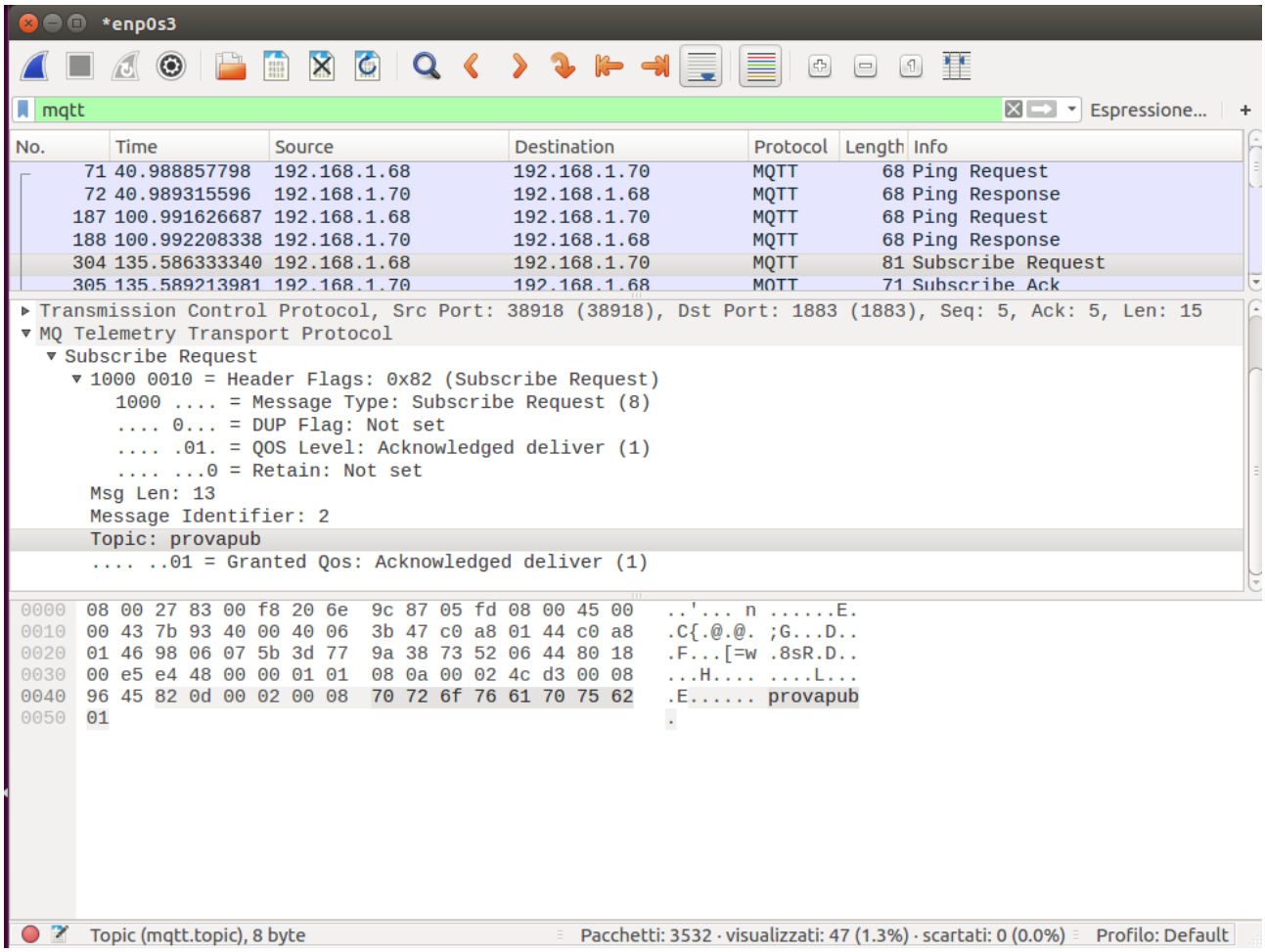


Figura 56-Struttura del pacchetto Subscribe analizzata con Wireshark

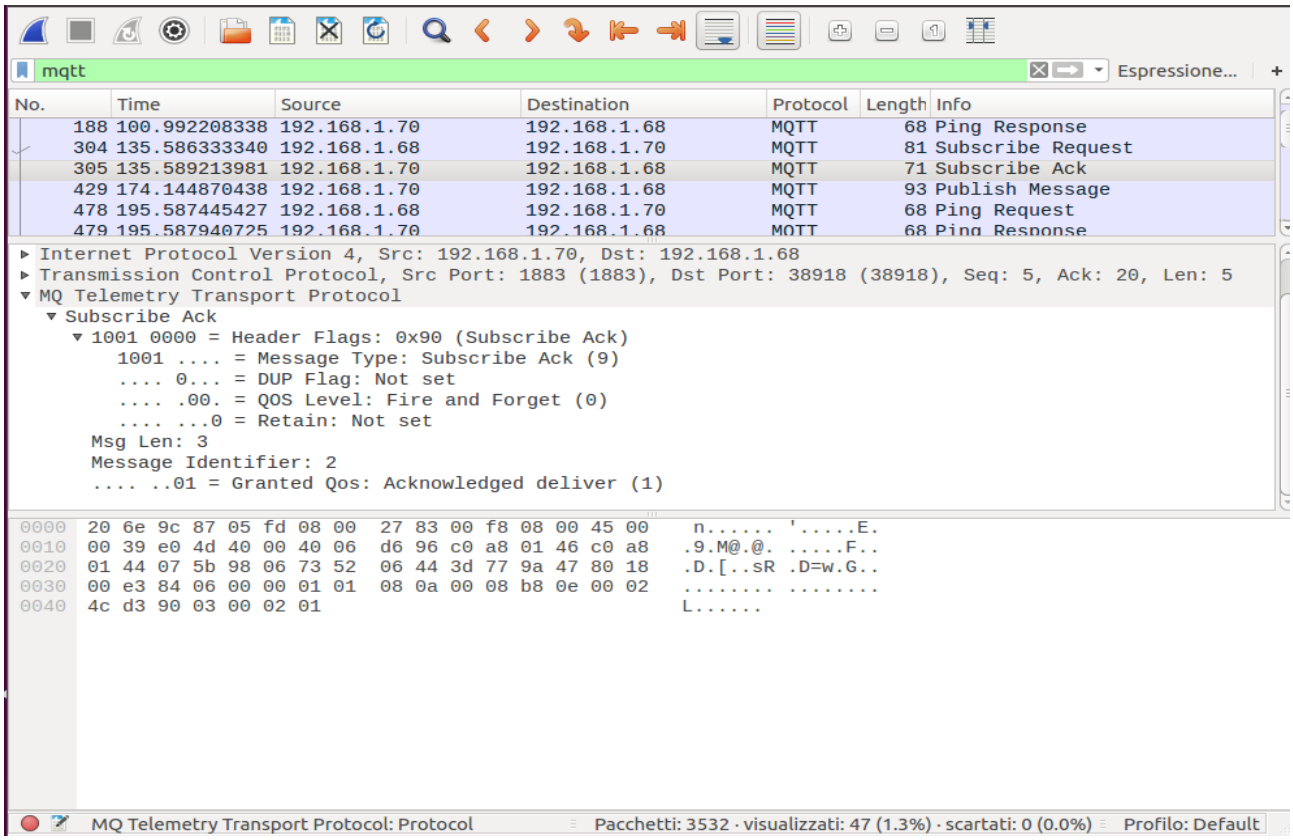


Figura 57-Struttura del pacchetto Suback analizzata con Wireshark

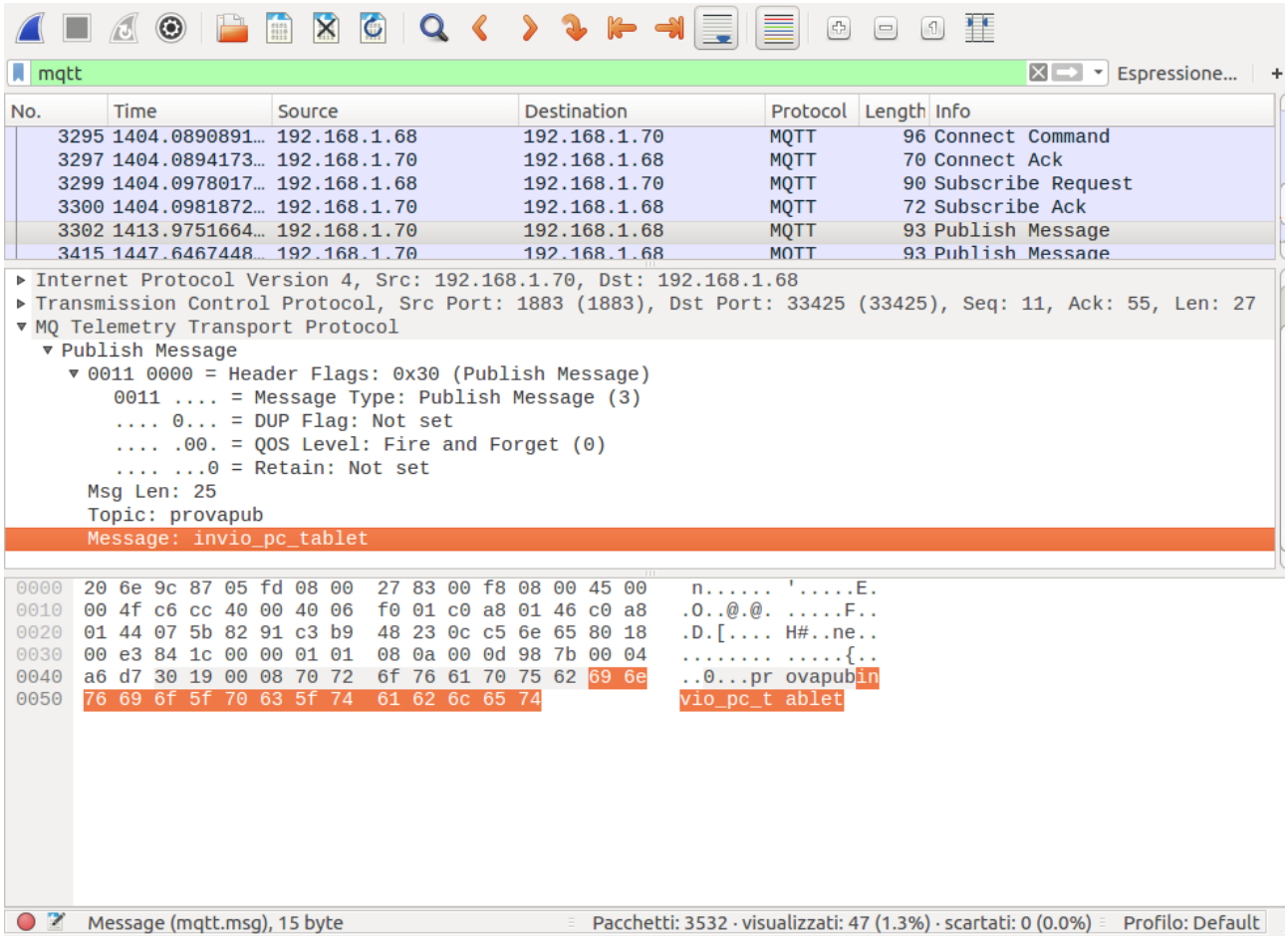


Figura 58-Struttura del pacchetto Publish da PC a tablet analizzata con Wireshark

In conclusione, per completezza riportiamo anche l'analisi dei pacchetti Pingreq, Pingresp e Disconnect, vediamo come il contenuto rispetta alla perfezione lo standard

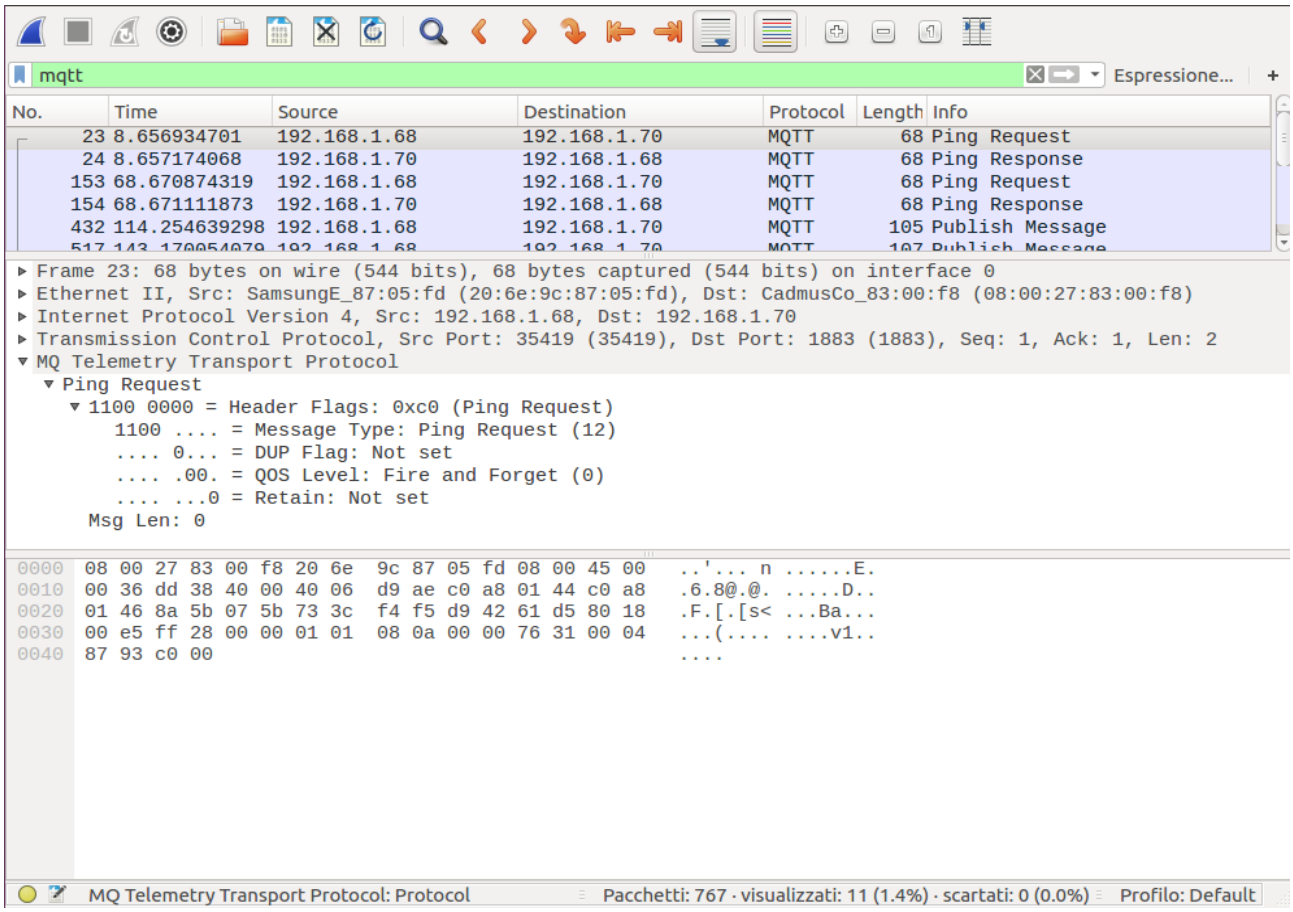


Figura 59-Struttura del pacchetto Pingreq analizzata con Wireshark

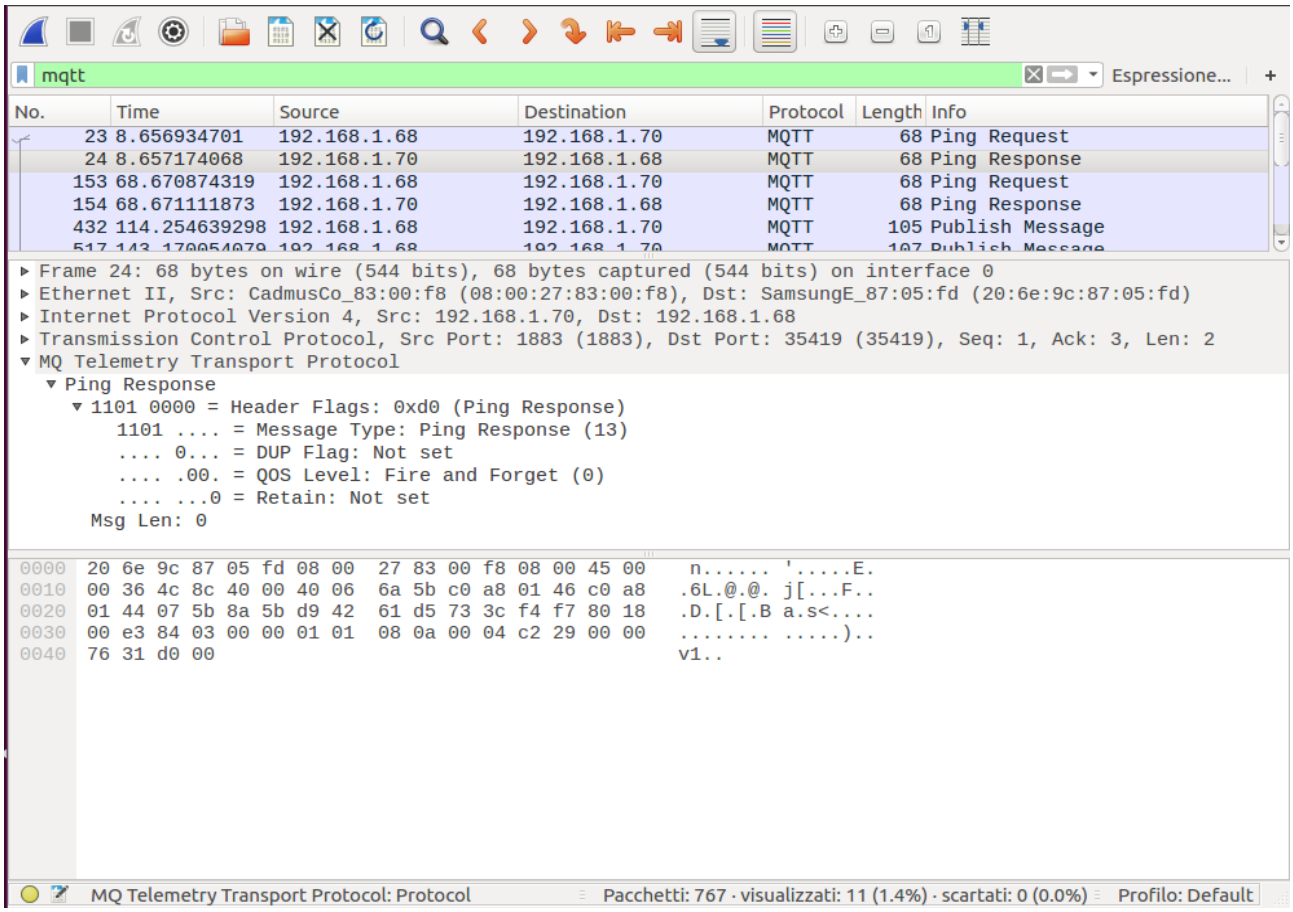


Figura 60-Struttura del pacchetto Pingresp analizzata con Wireshark

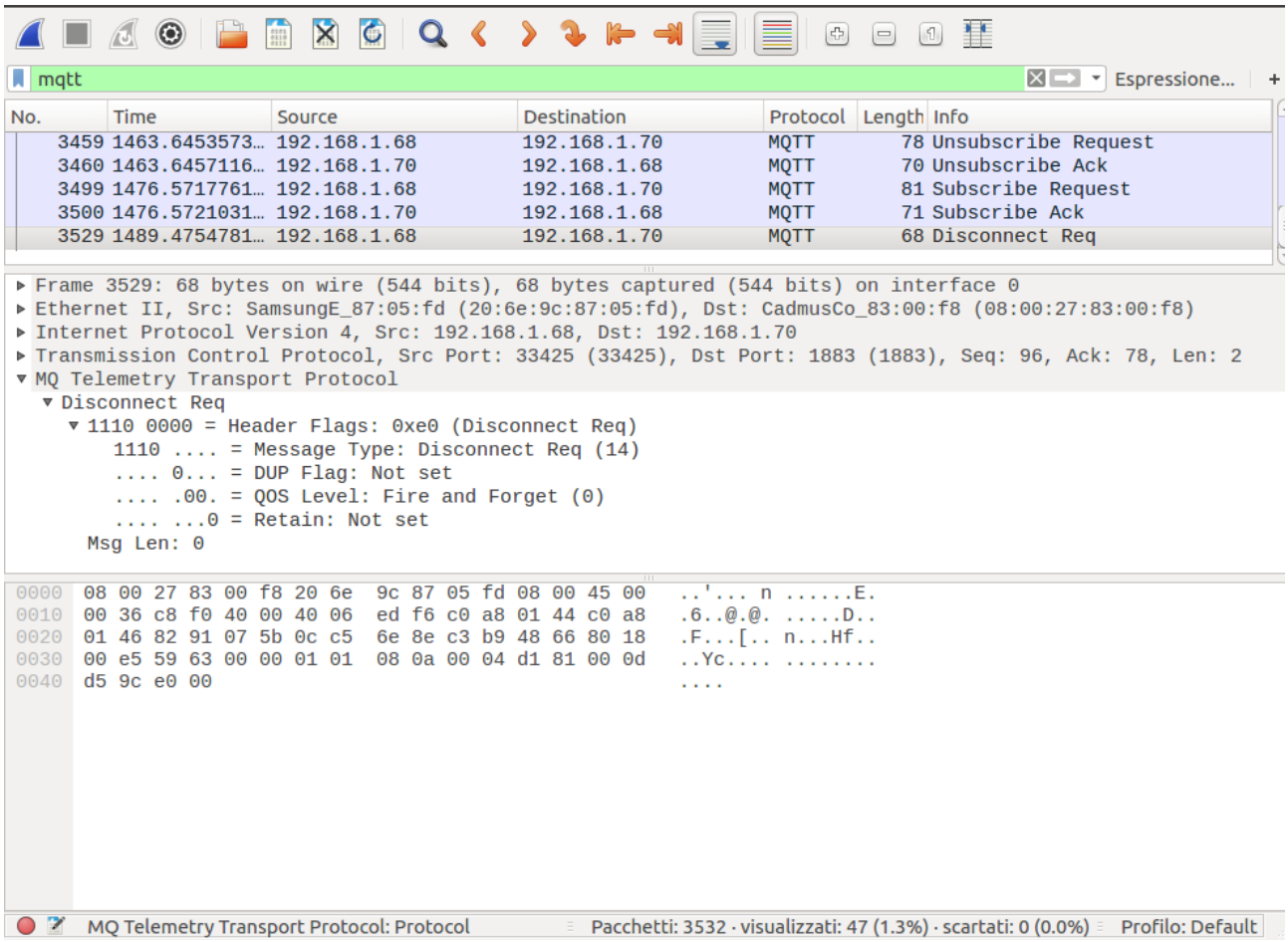


Figura 61-Struttura del pacchetto Disconnect analizzata con Wireshark

Da notare durante l'analisi di protocollo che i due indirizzi IP che si scambiano continuamente il campo di Source e Destination, sono 192.168.1.70 che come indicato al momento delle connessione ad un Broker da tablet è l'indirizzo del nostro pc e 192.168.1.68 che è l'indirizzo IP del tablet

Figura 62-Indirizzo IP del tablet

6-Soluzioni Preconfigurate per Dispositivi IoT

A scopo di ricerca sono state analizzate alcune piattaforme specifiche, fornite dalle principali aziende del settore, dedicate alla connessione di dispositivi nell'ambito dell'Internet of Things in un ambiente preconfigurato. All'interno di esse si trovano gli Hub pensati per la connessione dei dispositivi, ne riportiamo le caratteristiche dei due più comuni.

Microsoft AZURE [3.b]:

l'Hub IoT di Azure supporta in modo nativo la comunicazione tra i protocolli AMQP, MQTT e HTTP/1; nel caso di protocolli non supportati è presente un gateway personalizzabile. Per quanto riguarda l'implementazione del protocollo MQTT v3.1.1 notiamo:

-QoS 2 non è supportato. Quando un Client invia un PUBLISH con QoS 2, l'Hub IoT chiude la connessione di rete. Quando un Client invia un SUBSCRIBE con QoS 2, l'Hub IoT concede il livello QoS 1 massimo nel pacchetto SUBACK

-non supporta Retained Message, se un Client pubblica un messaggio con Retain Flag uguale a 1 il Broker non rende persistente il messaggio di mantenimento (se si utilizza l'applicazione back-end disponibile in Azure allora il messaggio le viene passato)

-esistono già disponibili per i dispositivi Client, degli SDK (Software Development Kit) che supportano il protocollo MQTT. Questi SDK per impostazione predefinita si connettono a un Hub con il flag Clean Session uguale a 0 e una QoS 1 per lo scambio di messaggi con il Broker

AWS IoT[4.b]:

è la piattaforma per IoT di Amazon Web Service, tra i suoi vari componenti troviamo il Message Broker con le caratteristiche elencate di seguito. Supporta l'uso del protocollo MQTT per eseguire PUBLISH e SUBSCRIBE, l'uso del HTTPS per il PUBLISH e il protocollo MQTT su WebSocket. Concentrandoci solo sul protocollo MQTT, che è quello che ci interessa maggiormente notiamo queste specifiche:

-QoS 0 significa che il messaggio sarà inviato 0 o più volte (invece di al massimo 1). Il messaggio potrebbe essere inviato più di una volta, in questo caso potrebbe avere un Packet ID differente e di conseguenza il DUP(Duplicate Delivery of a PUBLISH) non attivo.

-non supporta PUBLISH e SUBSCRIBE con QoS 2 e di conseguenza non invia PUBACK o SUBACK quando richiesta

-quando si risponde ad una richiesta di connessione con un CONNACK, il flag che indica di riprendere una sessione precedente, potrebbe avere un valore sbagliato se due Client si connettono con lo stesso ID contemporaneamente

-dopo un SUBSCRIBE può esserci ritardo tra il momento in cui il Broker invia un SUBACK e il momento che il Client inizia a ricevere nuovi messaggi

-non supporta Retained Message, se vengono richiesti, la connessione si chiude

-quando un Client si connette al Broker con un ID già in uso da un altro Client, viene inviato un CONNACK ad entrambi ed il Client al momento connesso verrà disconnesso

-non supporta sessioni persistenti, se il Client invia un messaggio con Clean Session uguale a 0, il Client verrà disconnesso

-in rare occasioni il Broker potrebbe rinviare lo stesso PUBLISH con un diverso ID packet

-il Broker non garantisce l'ordine in cui messaggi e ACK sono ricevuti

7-Confronto tra MQTT e HTTP

Verifichiamo se veramente la spesa in Byte è inferiore mostrando qui sotto l'invio del messaggio HelloWorld con entrambi i Protocolli e considerando per entrambi i casi, anche dell'invio di alcuni segmenti non relativi allo strato Applicativo ma necessari per la Connessione di Rete

HTTP, con metodo Post:

in questa analisi ci ha aiutato l'uso dell'analizzatore di protocolli WireShark e il seguente diagramma alla pagina[5.b]

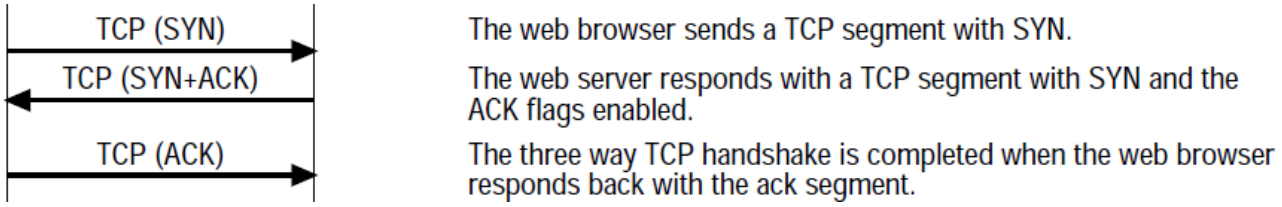
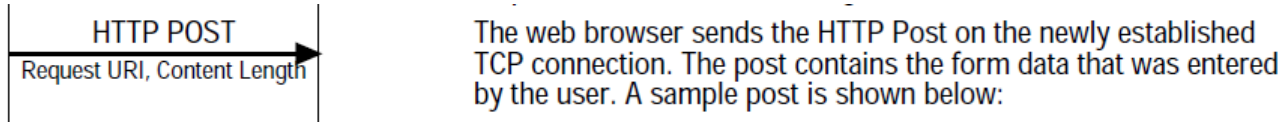


Figura 63-Three Ways Handshake

Questi primi tre messaggi sono fondamentali per aprire una connessione TCP e vengono chiamati Three Ways Handshake, occupano in totale 218 Byte (78+74+66)

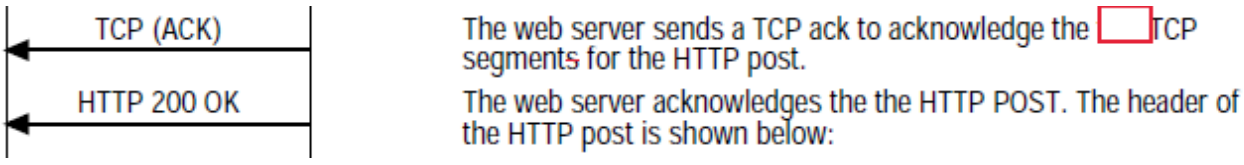


```
POST /form.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, /
Referer: http://www.anydomain.com/FillForm.htm
Accept-Language: en-us
Content-Type: multipart/form-data; boundary=---------------------------8d547442c03ba
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Host: anydomain.com
Content-Length: 10
Connection: Keep-Alive
Cache-Control: no-cache

-----------------------------8d547442c03ba
<Form data goes here>
-----------------------------8d547442c03ba--
```

Figura 64-Invio del Messaggio HelloWorld con metodo di richiesta Post

Adesso viene inviato il messaggio HelloWorld che vale 10 Byte come si può vedere nella casella Content – Length. A questi byte aggiungiamo un Header http per il messaggio Post di 555 Byte, più 66 Byte di Header degli strati inferiori. In totale abbiamo quindi 631 Byte




```
HTTP/1.1 200 OK
Date: Sat, 03 Dec 2005 11:25:07 GMT
Server: Apache/2.0.52 (Fedora)
Last-Modified: Sat, 03 Dec 2005 10:23:07 GMT
ETag: "2b8b2-150-857bb381"
Accept-Ranges: bytes
Content-Length: 416
Keep-Alive: timeout=10, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
```

Figura 65-Conferma di avvenuta ricezione

Due messaggi di conferma, il primo a livello TCP di 66 Byte ed il secondo http di 776 Byte (360 Header 416 Content-Length)

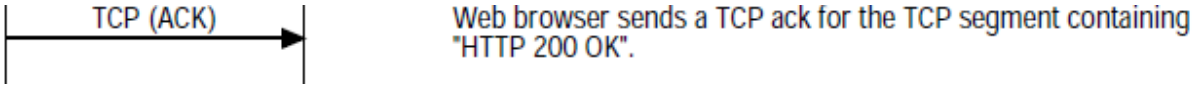


Figura 66-Conferma TCP del Client

Una conferma TCP da parte del Client, 66 Byte

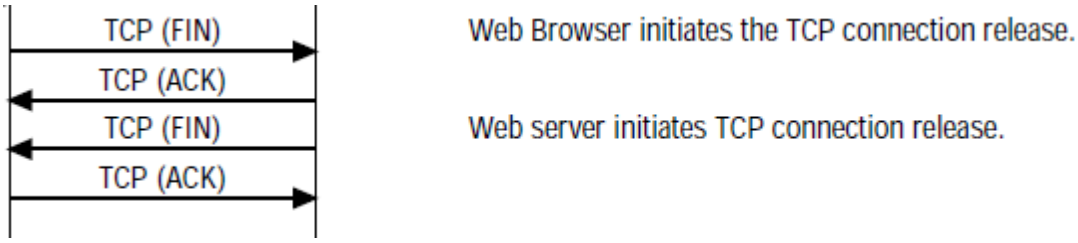


Figura 67-Scambio di messaggi Client-Server per chiusura connessione TCP

Ed infine la chiusura della connessione TCP di 4 messaggi da 66 = 264 Byte

Il Protocollo HTTP Post ha impiegato quindi 2021 Byte

MQTT:

come partenza per lo sviluppo di questo studio ci siamo inizialmente appoggiati ad un video trovato in rete[6.b]

Per l'analisi seguente consideriamo il caso in cui il Topic sia "SampleTopic" , il Client ID occupi 23 Byte e QoS = 0

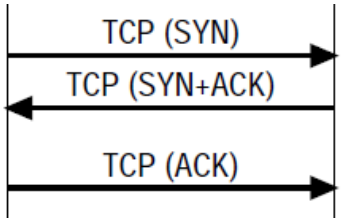


Figura 68-Three Ways Handshake

Three Ways Handshake di apertura TCP 218 Byte

MQTT CONNECT (105 Byte

TCP (ACK) (66 Byte

MQTT CONNACK (70 Byte

TCP (ACK) (66 Byte

MQTT SUBSCRIBE (84 Byte

TCP (ACK) (66 Byte

MQTT SUBACK (71 Byte

TCP (ACK) (66 Byte

MQTT PUBLISH (91 Byte

TCP (ACK) (66 Byte

MQTT DISCONNECT (68 Byte

poi una completa comunicazione MQTT con le rispettive conferme nello strato TCP

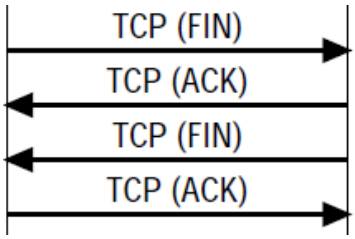


Figura 69-Chiusura connessione TCP

Ed infine la necessaria chiusura di connessione TCP di 264 Byte

Il Protocollo MQTT ha impiegato quindi 1301 Byte

DIFFERENZE RISCONTRATE:

usando il Protocollo MQTT, nonostante la quantità dei messaggi trasmessi sia maggiore, abbiamo risparmiato 720 Byte rispetto al Protocollo HTTP Post, ciò è dovuto alle ridotte dimensioni del suo Header. La nostra analisi poi, si limita all'invio di un singolo messaggio applicativo, quindi è facile immaginare il guadagno che si può ottenere con un numero maggiore di Application Messages.

8-Conclusioni

Dopo l'attenta analisi eseguita in questo scritto, possiamo affermare con certezza che, al momento, il protocollo MQTT rappresenta un'ottima scelta se usato per implementare comunicazioni M2M. Come notato nel capitolo precedente il protocollo MQTT è in grado di limitare il numero dei byte trasmessi rispetto al protocollo HTTP, usato con metodo POST, in più l'architettura publish/subscribe e le varie tipologie di Type Message sono in grado di fornire alla comunicazione una modularità raramente ottenibile con un'architettura richiesta/risposta. Lo studio, poi, di una comunicazione tra dispositivi IoT (Capitolo 4) ha fatto risaltare, oltre ai pregi detti prima, l'importanza di poter gestire con molta semplicità l'invio di messaggi one-to-many. Tutti risultati che speravamo di ottenere all'inizio della Tesi, sono stati confermati e inseriti in un contesto reale grazie al lavoro di analisi e progettazione svolto.

Bibliografia

[1.a] Androidiani. http://www.androidiani.com/news/steve-wozniak-e-la-bubble-phase-dellinternet-of-things-243195

[2.a] Digitalic http://www.digitalic.it/wp/tecnologia/internet-of-things-entro-il-2018-sara-piu-diffusa-degli-smartphone/97602

[3.a] Systems Integration http://www.systemsintegration.it/domotica-home-automation-iot-bello-ma-come-si-usa-il-termostato/

[4.a] Sgart https://www.sgart.it/IT/elettro/apertura-cencello-iot-controllato-via-wi-fi-esp8266/post

[5.a] jiffyjoseph http://www.jiffyjoseph.com/home/hacking-an-ordinary-washing-machine-to-iot-device

[6.a] Tecnologia360 http://tecnologia360.it/tikimo-scarpe-gps/

[7.a] Pet Pointer https://www.petpointer.ch/it/

[8.a] The High-Tech Society https://thehightechsociety.com/ingestible-medical-sensor/

[9.a] The Diss http://thedissnba.com/2014/10/31/exchanging-blood-for-profits/

[10.a] Enellaktikos http://www.enallaktikos.gr/kg12el_energeia.html

[11.a] Smart Cities World https://smartcitiesworld.net/news/news/smart-traffic-could-save-42-billion-man-hours-annually-653

[12.a] Linked in https://www.linkedin.com/pulse/world-tomorrow-naveen-nellore

[13.a] Instructables http://www.instructables.com/tag/type-id/?sort=none&q=remote+monitoring

[14.a] Linked in https://www.linkedin.com/pulse/how-mqtt-based-ibm-iotf-universe-change-tomorrow-aditya-om

[15.a] Insights Samsung https://insights.samsung.com/2015/06/10/securing-the-internet-of-things-risks-to-benefits/the-risks-and-advantages-of-the-internet-of-things-iot/

[16.a] Jaxenter https://jaxenter.de/smarthome-in-action-mit-openhab-und-mqtt-20108

[17.a] Programming with reason http://programmingwithreason.com/article-mqtt-in-depth.html

[18.a] Mosquitto https://mosquitto.org/

[19.a] Google Play https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client

[20.a] Wireshark https://www.wireshark.org/

[1.b] Key4Biz https://www.key4biz.it/iot-84-miliardi-di-oggetti-connessi-nel-2017-in-aumento-del-31/180486/

[2.b] Standard Oasis MQTT v3.1.1 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html

[3.b] Microsoft Azure https://www.azureiotsuite.com/

[4.b] Amazon Web Service http://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf

[5.b] Eventhelix https://www.eventhelix.com/RealtimeMantra/Networking/HTTP_Post.pdf

[6.b] YouTube https://www.youtube.com/watch?v=79bMMT7RPqY