Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Sistemi integrati per la domotica e per il risparmio energetico tramite micro elaboratori su singola scheda e basso costo

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Umberto Sgueglia

Sessione III
Anno Accademico 2013/2014

Always work hard
on something uncomfortably exciting.
Larry Page

Indice

Introduzione

Lo sviluppo ed il progresso tecnologico hanno sempre portato innovazioni in grado di poter migliorare lo stile di vita quotidiana dei singoli individui. Talvolta, oltre a migliorare le loro vite sono riuscite a rivoluzionarle completamente. Basti pensare ad invenzioni come il televisore, i primi calcolatori passando per i telefoni cellulari ed internet. Queste sono solo alcune delle invenzioni che hanno rivoluzionato la vita di ogni essere umano negli ultimi 50 anni. Oggi stiamo assistendo alla nascita di quella che qualcuno pensa possa essere una nuova pietra miliare per le generazioni future. Stiamo parlando Dell'Internet of Things. In questa tesi verrà mostrato un particolare aspetto dell'IoT che è quello relativo alla domotica. Verranno spiegati i principali aspetti di un sistema domotico, alcuni casi d'uso, e in fine sarà mostrata una sua realizzazione attraverso l'uso del protocollo MQTT.

Capitolo 1

Internet of Things

In telecomunicazioni per internet delle cose, dall'inglese Internet of Things (IoT), si intende l'estensione della rete al mondo degli oggetti. Questa estensione va a rivoluzionare quello che fino ad oggi è stato internet. L'architettura originaria di internet è stata creata quando ancora non era possibile prevedere la nascita di questa esigenza di mettere in comunicazione tra loro un grande numero di elettrodomestici, sensori ed oggetti di vario tipo. Tuttavia la diffusione sul mercato, maggiormente negli ultimi anni, di grandi varietà e tipologie di sensori attuatori e dispositivi capaci di connettersi tra di loro tramite internet, rischia di mettere in difficoltà l'attuale struttura di internet.

1.1 Network per IoT

La necessità di rivoluzionare l'architettura di internet nasce dall'esigenza di avere connesso in rete non più una grande quantità di utenti che comunicano tra loro, ma bensì una gigantesca quantità di macchine che si scambieranno informazioni sotto forma di minuscole frazioni di dati. Questo è esattamente il contrario di quanto avviene nei classici network come internet. I protocolli più importanti di internet come ad esempio il TCP/IP si basano su connessione generalmente affidabile tra mittente e destinatario. Nel caso dell'IoT il concetto è generalmente diverso. L'idea di fondo è quella di una informazio-

ne che viene inoltrata nella rete e successivamente ogni macchina sparsa in questa rete coglie esclusivamente la piccola informazione di cui ha veramente bisogno. La differenza tra le due reti oltre ad essere l'approccio, sono anche le macchine che la sfruttano. Infatti le macchine che attualmente navigano in internet sono per la maggior parte dotate di potenza di calcolo della cpu sempre in continua crescita, così come memoria e altri vari componenti sempre più potenti. L'IoT è pensato in modo completamente opposto. I dispositivi previsti che fanno parte di questa rete dovranno essere dotati di minor potenza di calcolo, dispendio energetico e risorse, possibili. Le motivazioni di queste limitazioni sono fondamentalmente due, la prima è che ci potrebbe essere un dispendio di risorse dal punto di vista energetico eccessivo e in secondo luogo andrebbe a crescere il costo del singolo dispositivo. Sono questi i motivi per cui a tutt'oggi i protocolli più diffusi come il TCP/IP, già pronti e funzionanti, sono per la maggior parte inadatti per l'IoT[1]. Il problema di fondo è la pura e semplice robustezza dei protocolli, che sono progettati per usi intesivi, moli massicce di dati e affidabilità. Tutte cose di cui l'IoT non ha bisogno. Per quanto possa sembrare strano, i dispositivi meno sofisticati sono i più sicuri. Ogni singolo dispositivo IoT che fosse dotato di un sistema operativo e di una memoria di qualche tipo sarebbe a rischio di eventuali errori di configurazione o errori software che crescono esponenzialmente con la sua complessità. Inoltre i sistemi operativi e gli stack protocollari devono essere aggiornati per garantirne la costante sicurezza. Nell'IoT, l'estrema semplicità delle comunicazioni con i dispositivi finali limita i rischi per la sicurezza. La progettazione di una rete alternativa deve infatti garantire che il "danno" che i dispositivi finali potrebbero infliggere al sistema sia minimizzato e quindi la network intelligence decentralizzata dal diretto contatto con essi.

1.2 Protocolli appositi

Uno dei motivi principali quindi dell'inutilità dei protocolli robusti nell'IoT risiede nel divario tra le risorse che questi richiedono e le capacità minime di elaborazione, di memoria e di comunicazione che caratterizzano molti dei più semplici dispositivi IoT. Ha quindi senso trovare una soluzione nuova, da applicare in parallelo ai dispositivi esistenti basati sull'IP, per gestire in modo efficiente la massa immensa di dati generati dalle apparecchiature che non necessitano dell'IP o che sarebbero ostacolate da esso. Sovraccaricare con stack protocollari apparecchi altrimenti semplici come sensori elettrici e macchine per il caffè non farebbe che aumentare nettamente il costo e la complessità dei miliardi di esemplari prodotti. Per superare questi limiti c'è quindi un grande bisogno di cambiare il modo di vedere le cose. Una soluzione possibile è quella di implementare nuovi protocolli caratterizzati dal fatto di essere orientati al ricevente. Con "orientati al ricevente" si intende che la fonte produce il dato e lo spedisce nella rete, se questo arriva ad un ricevente che era in ascolto per la ricezione di dati di quel tipo, allora esso li analizza e li utilizza, altrimenti il funzionamento generale della rete resta inalterato. L'architettura dell'IoT è pensata per rendere partecipe un maggior numero di figure presenti sul mercato, riducendo la quantità di conoscenze e di risorse necessarie ai margini del network. Oltre questo, la rete deve essere altamente tollerante nei confronti di errori e interruzioni nella connessione. L'affidabilità delle informazioni dovrà essere compito dei singoli strumenti connessi con la rete classica, che poi daranno tutte le informazioni necessarie alle enormi schiere di dispositivi disperse nella rete IoT. Nelle normali connessioni WAN (wide area networking) e wireless l'ampiezza o spettro di banda è costosa e limitata, a fronte di una quantità di dati da trasmettere ampia e in continuo aumento. I protocolli classici prevedono numerosissimi controlli e doppi controlli dell'integrità del messaggio per minimizzare costose ritrasmissioni. Nonostante questo però la maggior parte dei dispositivi dell'IoT dovrà basarsi su connessioni wireless e quindi c'è bisogno di rivoluzionare tutto. La quantità dei dati sarà principalmente irrisoria e assolutamente non determi-

nante. Infatti i dispositivi finali saranno progettati per funzionare anche in caso di interruzione della trasmissione o della ricezione, anche per periodi prolungati. E' proprio questa autosufficienza che va ad eliminare la necessità di controlli eccessivi per la ricezione o l'invio dei singoli controlli che quindi risultano inutili. Una volta definite le tipologie di comunicazione della rete bisogna cominciare a pensare al tipo di pacchetto. Infatti questo deve offrire il minimo indispensabile e non di più. Questi piccoli pacchetti, definiti anche *chirp*, ossia "cinguettii", stanno alla base della costruzione di una rete fatta in questo modo. Questi devono differire dai normali pacchetti IP per diversi motivi, ossia, devono incorporare livelli minimi necessari di overhead e non devono comprendere meccanismi di ritrasmissione o riconoscimento. Qualsiasi altra operazione per il mantenimento della rete, ad esempio il routing, deve essere effettuata da altri dispositivi, i dispositivi della rete classica, che aggiungeranno informazioni ai singoli pacchetti ricevuti. Nella rete IoT gli end point interessati potrebbero facilmente diventare un numero molto elevato, anche maggiore della stessa popolazione mondiale. Questi però tenderanno ad essere estremamente economici, autonomi e non particolarmente soggetti al comando e al controllo da parte dell'elemento umano.

1.3 Perché l'IP è inadatto

Il protocollo IP (internet protocol) è lo strumento chiave usato oggi per costruire una interconnessione di reti scalabili ed eterogenee[2]. L'idea di fondo è che IP venga eseguito da tutti i nodi (host e router) di un insieme di reti e che definisca un'infrastruttura che consente a tali nodi e reti di funzionare dal punto di vista logico come una singola rete interconnessa. Quindi il compito principale di IP è l'instradamento tra reti eterogenee. Il Datagramma IP è un elemento fondamentale dell'Internet Protocol. Ogni datagramma porta con sé le informazioni necessarie affinché la rete possa inoltrare il pacchetto fino alla sua destinazione corretta. Una delle più importanti caratteristiche di questo protocollo è che può essere eseguito ovunque. Ovviamente un ruolo

fondamentale nel protocollo IP è assunto dal tipo di pacchetti che possono essere trasportati. Il datagramma IP, come la maggior parte dei pacchetti, consiste di un'intestazione seguita da un certo numero di byte di dati. Alla sua progettazione nessuno immaginava che la rete fosse un giorno cresciuta così drasticamente da contenere miliardi di dispositivi come si prevede per l'IoT da qui a pochi anni. Questo è chiaro se si guarda con un maggiore grado di attenzione l'implementazione di IP. Nella versione attualmente più diffusa, IPv4, lo spazio di indirizzamento è di 2^{32} indirizzi. Il numero totale di indirizzi ottenibile quindi grazie a questo protocollo è di "soli" 4,3 miliardi di elementi. Il problema è che le previsioni di diffusione di dispositivi per l'IoT indicano che ci sarà una crescita molto maggiore di questa cifra. Per superare questo limite è stata introdotta una nuova versione del protocollo IP: IPv6. Che come differenza principale rispetto IPv4 possiede un indirizzamento a 128 bit a fronte dei soli 32 bit di IPv4. Ipotizzando una efficienza dell 100% IPv6 quindi può arrivare a instradare 3.4 x 10^{38} nodi. Una eventuale transizione da IPv4 a IPv6 però, non è poi così semplice. Infatti la rete internet è troppo grande e decentralizzata per poter avere un unico istante di transizione in cui tutti gli host e tutti i router vengono aggiornati da IPv4 a IPv6[2]. Di conseguenza è necessario che IPv6 venga installato in maniera graduale, con una metodologia che consenta agli host e ai router che gestiscono soltanto IPv4 di continuare a funzionare il più a lungo possibile. L'idea ottimale per la transizione è che essa venga fatta molto lentamente e quindi gli host che vengono progettati per l'IPv6 gestiscano sia IPv6 che IPv4, in modo da ottenere un progressivo ricambio generazionale. In realtà IPv6 esiste già, ma non è esattamente quello che serve per l'IoT. Infatti resta un protocollo pesante che necessita di risorse e di elaborazione. Una soluzione accettabile per l'IoT potrebbe però essere una fusione tra l'IPv6 ed altri protocolli citati in precedenza. Ovvero avere nodi all'interno della rete con indirizzi IPv6 laddove è strettamente necessario, e invece avere altri nodi più semplici che fungano solo da end point della connessione.

1.4 Novità nell'IoT

Un primo rapporto tra diretti consumatori e IoT si sta sviluppando particolarmente nell'ultimo anno con la diffusione di oggetti wearable, come orologi e bracciali intelligenti. Grazie ad essi, in particolare sfruttati in coppia con uno smartphone, l'utente è in grado di restare connesso ad internet in modo ancora più semplificato. Il mercato della domotica in Italia è in fase di forte sviluppo, con trend di crescita che secondo Assodomotica, una associazione di soggetti pubblici e privati interessati alla massima divulgazione della cultura di integrazione delle tecnologie usate nelle case, si manterrano intorno al 30% nei prossimi anni. L'attuale tendenza degli utenti è di realizzare impianti domotici in abitazioni nuove o ristrutturate, in Italia si costruiscono circa 300.00 abitazioni l'anno e se ne ristrutturano circa 700.000 con il completo rifacimento dell'impiantistica. Nel 40% dei casi, questo milione di proprietari è molto interessato alla sicurezza, denotando quindi, una maggior attenzione a questo problema di quanto non accada mediamente nel resto dell'Europa.

Secondo uno studio svolto dal sito domotica.it in Italia la domotica è in grande sviluppo[8].

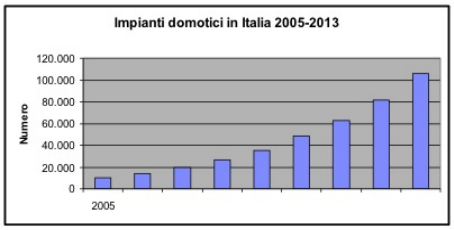


Figura 1.1: Studio di domotica.it

Il 26 agosto 2014, la stampa, ha pubblicato un articolo sul significato reale di IoT[7] e su come questo ancora non sia esattamente alla portata di tutti. In questo articolo si faceva riferimento ad una previsione del mercato dell'IoT fatta dall'acquitygroup nel 2014 che ha rivelato come potrebbe cambiare la percentuale degli acquirenti e utilizzatori di domotica nei prossimi anni.

Secondo questi studi nel prossimo anno questo numero aumenterà di circa il 10%, nei prossimi 5 anni si parla di un 30%, mentre ad un totale accettazione di questa tecnologia si prevede un utilizzo di circa il 70% delle abitazioni.

Capitolo 2

Mqtt

E' il protocollo pensato da IBM per l'IoT nel 1999. L'acronimo sta per Message Queuing Telemetry Transport protocol.

2.1 Introduzione al protocollo

MQTT è un protocollo basato sul modello publish/subscribe progettato per essere aperto, semplice, leggero e facile da implementare. Queste caratteristiche lo rendono perfetto per essere utilizzato in situazioni particolari quali per esempio:

a) Quando la rete ha bisogno di poco bandwidth.

b) Quando il sistema che lo implementa è eseguito su sistemi embedded con limitate capacita di memoria e di CPU

La sua implementazione utilizza tcp, ed è un protocollo machine-to-machine(m2m). L'Organization for the Advancement of Structured information (OASIS) ha dichiarato che il protocollo MQTT(Message Queuing Telemetry Transport) di IBM è lo standard di riferimento per l'IoT. MQTT potrebbe seguire lo stesso destino dell'HTTP, che è divenuto uno standard nella condivisione delle informazioni attraverso il World Wide Web. La pri-

ma versione di MQTT è stata rilasciata da IBM nel 1999. Da allora sono state rilasciate diverse versioni. Attualmente siamo alla versione 3.1

2.2 Formato del messaggio

Ogni messaggio previsto con un formato MQTT ha bisogno di un header fissato e di un payload. In particolare l'header di ogni messaggio è strutturato in due byte come spiegato di seguito[6]:

- **Byte 1** : Contiene il tipo del messaggio e le varie flag (DUP, QoS, Retain) da settare.
- **Byte 2**: Contiene la lunghezza rimanente.

2.2.1 Tipo di messaggio

Le informazioni relative a questa flag sono presenti nel primo byte e precisamente occupano le posizioni dal settimo al quarto bit. Il nome del tipo viene quindi descritto utilizzando 4 bit quindi in modo da descrivere un numero tra 0 e 15. Ed in particolare: con 0 si indica il tipo riservato(reserved), con 1 si indica che il client vuole connettersi al server (connect), con 2 si indica un acknoledgment (connack), con 3 si indica l'operazione di pubblicazione di un messaggio (publish), con 4 si indica l'acknoledgment di una publish (puback), con 5 si vuole indicare una publish ricevuta (pubrec), con 6 una publish rilasciata (pubrel), con 7 si indica una publish completata (pubcomp), con 8 si indica una richiesta di un client che si vuole iscrivere (subscribe), con 9 si indica un aknowledgment di un iscrizione (suback), con 10 si indica invece la volontà di un client di disiscriversi (unsubscribe), con 11 si indica un acknowledgment di una disiscrizione (unsuback), con 12 si indica una richiesta di ping (pingreq), con 13 una risposta ad un ping (pingresponse), con 14 si vuole indicare una disconnessione (disconnect), mentre il 15 è riservato.

2.2.2 Flags

Come detto in precedenza i rimanenti bit del primo byte contengono i campi Dup, che sta per l'invio duplicato, QoS, che sta per la qualità del servizio, e il Retain. In particolare le loro posizioni nel byte sono: il bit 3 per il Dup, bit 2 e 1 per il QoS e il bit 0 per il retain. In particolare la flag di Dup viene settata quando il client o il server cercano di riinviare un publish, pubrel, subscribe o un unsubscribe. Questa flag può essere settata esclusivamente per i messaggi che hanno una qualità del servizio maggiore di 0, ed è richiesto un acknoledgment. Quando il dup è settato l'header comprende un message id. La seconda flag , quella relativa alla qualità del servizio indica il livello di affidabilità del servizio. Ed in particolare: con la QoS settata a 0 si indica la caratteristica di al più 1, settato a 1 si indica almeno 1, settato a 2 esattamente 1, e il QoS settato a 3 è riservato. La flag di retain è utilizzata esclusivamente per i messaggi di publish. Infatti, quando un client invia un messaggio di publish ad un server, se il flag di retain è settato, il server conserverà quel messaggio anche dopo averlo inviato ai correnti iscritti. Nel momento in cui un nuovo cliente si iscrive allo stesso topic, gli viene inviato quel messaggio con il flag di retain settato.

2.2.3 Lunghezza rimanente

Questo dato è memorizzato nel secondo byte. Esso rappresenta il numero di byte rimanenti del messaggio corrente, includendo il payload. La codifica dei messaggi avviene usando un singolo byte per messaggi fino a 127 byte. I messaggi di dimensioni maggiori vengono gestiti come spiegato di seguito. I primi 7 bit di ogni byte tengono il dato, e l'ottavo altri byte seguenti nella rappresentazione. Ad esempio il numero 64 è rappresentato come un singolo byte. Il numero 321 (= 65 + 128*2) è memorizzato come due byte. Il protocollo limita i byte di rappresentazione ad un massimo di 4.

2.2.4 Header variabile

Alcuni comandi MQTT possiedono anche questa componente che è opzionale. Quando presente, essa è compresa tra l'header fisso e il payload. La lunghezza variabile non è parte dell' header variabile. La costituzione dell' header variabile è strutturata come segue:

- **Protocol name**: è presente nel header variabile del messaggio MQTT CONNECT.
- **Protocol version**: è anche esso presente nell'header variabile del messaggio MQTT CONNECT
- **Clean session flag**: esso è rappresentato dal bit 1 del byte della CONNECT. Se è settato a 0, il server deve memorizzare la sottoscrizione del client anche dopo che il client si è posto offline. Se è settato a 1 non dovrà mantenere queste informazioni a seguito di una disconnessione del client.
- **Will flag**: esso è rappresentato dal bit 2 del byte della CONNECT. Quando questo flag è settato il server pubblica un messaggio a nome del client quando, o il server incontra un errore di I/O durante la comunicazione con il client, oppure il client non riesce a comunicare prima dello scadere del tempo di keep alive.
- **Will QoS**: sono i bit 3 e 4 del messaggio CONNECT. Il client specifica il livello di QoS per un Will message che deve essere inviato in caso di una disconnessione del client imprevista. Questo messaggio è definito nel payload della CONNECT.
- **Will retain flag**: è il bit numero 5. Questo bit indica se il server deve mantenere i messaggi inviati con questa flag.
- **Username e Password flag**: sono alla posizione 6 e 7 del byte della CONNECT. Un client può specificare se il messaggio che sta inviando

è formato da un user o una password, e i dati in questione vengono inseriti nel payload della CONNECT.

- **Keep alive timer**: misurato in secondi, definisce il massimo intervallo di tempo tra i messaggi ricevuti dal client. Questo consente al server di capire che la connessione del client è crollata senza dover attendere i lunghi timeout del TCP/IP. Il Client ha la responsabilità di inviare dei messaggi ogni volta per non far scadere questo timer. Allo stesso modo se il client non riceve risposta dal server nello specifico tempo, chiude il canale TCP/IP.

- **Connect return code**: è il valore di ritorno della CONNECT. In particolare specifica: Connessione accettata se il flag è posto a 0, connessione rifiutata causa di versione di protocollo non compatibile se posto a 1, connessione rifiutata causa identificatore rifiutato se il codice di ritorno è posto a 2, server non accessibile se il valore è uguale a 3, nome utente o password errati se invece il valore di ritorno è 4 e infine con il valore 5 il codice di ritorno indica una negata autorizzazione.

- **Topic name**: questo è presente nel header variabile del messaggio MQTT PUBLISH. Questo nome è la chiave che identifica il canale dove il messaggio verrà pubblicato. I client utilizzano questa chiave per sottoscriversi ai canali e ricevere i messaggi.

2.2.5 Payload

I seguenti messaggi MQTT posseggono un payload:

- **Connect**: in questo caso il payload contiene una o più stringhe utf-8. Queste specificano un univoco identificatore per il client, un will topic e messaggio e un Username e una password. Alcuni di essi sono opzionali e dipendono dalle flag settate.

- **Subscribe**: il payload contiene una lista di topic a cui l'utente può iscriversi e il livello di QoS.

- **Suback**: il payload contiene una lista di QoS concessi. Questi sono i livelli di QoS ai quali l'amministratore del sistema concede ai singoli client di connettersi con determinati permessi.

2.3 Command message

2.3.1 Connect

Un client richiede la possibiltà di connettersi ad un topic. In un primo momento viene stabilita una connessione TCP/IP tramite un socket, e in seguito entra in gioco il protocollo MQTT. Nella Connect le flag di Dup, QoS, e retain non sono utilizzate, sono invece utilizzate la lunghezza rimanente e il payload. Nella lunghezza variabile abbiamo la possibilità di settare le flag di User, Password, Clean session, e il timer. Nel payload della CONNECT troviamo una o più stringhe utf-8. Le stringhe se presenti appaiono nel seguente modo:

- **Client identifier**: è una stringa di lunghezza compresa tra 1 e 23 caratteri, identifica univocamente il client. Esso deve essere univo tra tutti i client di un singolo server. Se l'id del client contiene un numero di caratteri maggiore di 23, il server risponde alla connect con un CONNACK che ritorna codice di errore 2: identificatore rifiutato.

- **Will topic**: questa è la stringa che mostra il topic dove dovrà essere inviato il will message. In cui il livello di QoS è definito dal will QoS e il retain dal will retain.

- **Will message**: questo rappresenta il messaggio che viene pubblicato nel will topic se il client si disconnette in maniera inaspettata.

- **Username**: se il relativo bit è settato questo rappresenta l'user del client. E' raccomandato di avere un user uguale o inferiore a 12 caratteri, ma non è strettamente richiesto.

- **Password**: se il bit è settato è l'username è correttamente impostato questa stringa indica la password relativa all'utente del client.

Il server manda un messaggio di CONNACK in risposta alla connect del client. Se il client non riceve una CONNACK dal server chiude il canale TCP/IP. Se un client manda una CONNECT invalida il server procede a chiudere la connessione.

2.3.2 Connack

Questo messaggio è inviato dal server nel seguito della ricezione di una CONNECT. Anche in questo caso le flag di Dup, QoS e retain non vengono utilizzate. Il campo di lunghezza variabile è occupato unicamente dal valore di ritorno da restituire al client. Il payload non è presente.

2.3.3 Publish

Un messaggio di questo tipo è inviato dal client ad un server, quando il client ha necessità che questo messaggio sia consegnato a tutti gli iscritti a quel topic. Per questo ogni messaggio è associato ad un topic. Un messaggio pubblicato su uno specifico topic è inoltrato a tutti i suoi iscritti. In questo caso è possibile settare le flag QoS, Dup e retain. L'header variabile contiene:

- **Topic name**: una stringa utf-encoded che indica il nome del topic dove si vuole pubblicare il messaggio.

- **Message ID**: presente esclusivamente per i messaggi in cui il QoS è settato ad un valore uguale ad 1 o superiore.

Il payload contiene il messaggio da pubblicare. Il response di una PUBLISH dipende dal livello di QoS. Se il QoS è settato a zero, non viene inviato nessun messaggio di ritorno. Se altrimenti il QoS è settato ad uno viene inviato un messaggio di PUBACK a chi ha inviato il messaggio. Se invece il QoS è settato a due viene restituito un messaggio di PUBREC al client.

2.3.4 Puback

Questo è un messaggio di ritorno di una PUBLISH con un QoS settato a 1. In questo caso le flag di Dup, QoS e retain non vengono utilizzate. L'header variabile può contenere l'id del messaggio. Non è presente payload in questo caso. Quando un client riceve un messaggio di questo tipo considera inviato correttamente il messaggio precedente e quindi può eventualmente eliminarlo.

2.3.5 Pubrec

Un messaggio di questo tipo è il response che un server invia al client quando esso aveva precedente inviato un messaggio di PUBLISH con il livello di QoS settato a 2. Analogamente al comando precedente anche in questo caso non sono presenti flag di Dup, QoS, retain ne tantomeno un payload. Questo comando serve unicamente a determinare che il messaggio precedente è stato inviato correttamente.

2.3.6 Pubrel

Questo messaggio è in risposta o da un publisher ad un pubrec da un server, oppure dal server a un messaggio di pubrec da un iscritto, è il terzo messaggio in una connessione in cui è settato il QoS uguale a 2. In questo caso possono essere settati i flag di Dup e di QoS ma non quello di retain. L'header variabile contiene l'id del messaggio a cui verrà fatto l'acknoledgment. Non è previsto un payload per questo comando. Quando un server riceve un messaggio di questo tipo da un publisher, esso rende disponibile il messaggio originale per tutti gli iscritti interessati. Quando un iscritto riceve un messaggio di Pubrel da un server, esso lo rende presente all'applicazione iscritta e manda un messaggio di PUBCOMP al server.

2.3.7 Pubcomp

In questo caso il messaggio può essere o una risposta da un sever che ha ricevuto una PUBREL o la risposta di un iscritto che ha ricevuto dal server una PUBREL. Questo è il quarto e ultimo messaggio in una connessione con un livello di qualità pari a 2. In questo caso le flag del header fisso, ovvero Dup, QoS e retain non vengono utilizzate. Nell'header variabile è presente lo stesso message id che è notificato dalla PUBREL. Il payload per questo comando non è previsto. Quando un client riceve un messaggio di PUBCOMP, esso elimina il messaggio originale perché questo vuol dire che è stato inviato correttamente.

2.3.8 Subscribe

Un messaggio di SUBSCRIBE consente al client di registrarsi ad uno o più topic contemporaneamente. I messaggi pubblicati in questi topic saranno inviati dal server come messaggi PUBLISH ad ogni singolo client in ascolto su quel topic. Il messaggio di subscribe specifica inoltre il livello di QoS con il quale un client vuole ricevere i messaggi del relativo topic a cui si sta iscrivendo. Le flag in aggiunta settate sono quella relativa al livello di QoS, quella relativa al Dup ma non quella relativa al retain. L'header variabile contiene un message id perché il messaggio di subscribe ha un livello di QoS pari a 1. Il payload contiene una lista di topic al quale il client vuole iscriversi, e il livello di QoS con il quale vuole ricevere i relativi messaggi. A seconda del livello di QoS impostato per ogni topic il client riceverà una serie di comandi citati in precedenza.

2.3.9 Suback

Questo tipo di messaggio viene inviato dal server al client per notificare di aver ricevuto ed accettato una sottoscrizione ad un topic. Un messaggio SUBACK contiene una lista di livelli QoS. In questo messaggio non sono previste le flag di QoS, Dup e retain. L'header variabile può contenere l'id

del messaggio che è stato notificato. Il payload di questo comando presenta una lista di livelli QoS. Ogni livello corrisponde ad un topic nel corrispondente messaggio di SUBSCRIBE. L'ordine dei livelli di QoS nel comando SUBACK è equivalente a quello fatto con i vari messaggi di SUBSCRIBE.

2.3.10 Unsubscribe

Un messaggio di UNSUBSCRIBE è inviato da un client ad un server per richiedere l'eliminazione dalla lista di iscritti di quel topic presente nel server. Con questo comando vengono fissate la flag di QoS e la flag di dup ma non viene utilizzata la flag di retain. Un messaggio di UNSUBSCRIBE ha un QoS equivalente a 1 quindi l'header variabile contiene l'id del messaggio. In risposta ad un UNSUBSCRIBE il server invia al client un messaggio di tipo UNSUBACK.

2.3.11 Unsuback

Il comando Unsuback è inviato dal server al client per confermare un UNSUBSCIBE ricevuto. Nessuno dei tre flag dell'header fisso è utilizzato, mentre l'header variabile contiene l'id del messaggio. Non è presente nessun tipo di payload.

2.3.12 Pingreq

Questo è un messaggio del tipo: "sei vivo?". Inviato da un client connesso ad un server. Non è presente un header variabile ne un payload.

2.3.13 Pingresp

E' il messaggio di risposta in seguito ad un PINGREQ e sta ad indicare una cosa del tipo: "si sono vivo". Neanche in questo caso vengono usate ne le flag dell'header fisso, ne quelle di quello variabile e non e neppure presente un payload.

2.3.14 Disconnect

Messaggio inviato dal client al server che indica che sta per chiudere la connessione TCP/IP. Questo comando non consente di settare le flag di QoS, Dup o retain e non possiede un header variabile. In fine non possiede neanche un payload.

Capitolo 3

Realizzazione di un sistema domotico

3.1 caratteristiche di un sistema domotico

Prima di poter realmente implementare un sistema di domotica bisogna chiedersi cosa si intende per domotica. Una definizione esplicativa è quella di Assodomotica: "La domotica o home automation è la materia che si occupa dell'integrazione delle tecnologie e degli impianti nelle abitazioni per realizzare case intelligenti, confortevoli, sicure e di semplice funzione". Tra le caratteristiche troviamo:

- **Vantaggi funzionali**: elementi creati per migliorare la sicurezza, aumentare il comfort, predisporre il sistema centrale per una completa comunicazione con i singoli componenti, ottimizzare i consumi energetici. Questi sono alcuni dei molti vantaggi che un sistema di domotica può apportare ad una casa.

- **Creazione di scenari**: in modo che l'utente possa in maniera semplice ricreare delle condizioni di luminosità o di temperatura da lui particolarmente apprezzate.

- **Interfaccia utente**: in un buon sistema di domotica si deve al più possibile semplificare la vita dell'utente che ha bisogno di accedere al sistema centrale attraverso una semplice interfaccia. Essa può essere composta da consolle lcd o telecomandi vari o ancora touch screen in giro per la casa o altrimenti usando un semplice client web o smartphone.

3.2 Utilizzi

Per comprendere meglio l'utilizzo di un sistema domotico si possono immaginare per esempio i seguenti casi d'uso. Per esempio un sistema domotico può risultare ottimale per la gestione del comfort ambientale, avendo la possibilità di gestire riscaldamento dell'aria e del condizionamento. Un altro ambito estremamente coperto da un sistema domotico può essere il campo della comunicazione e informazione, in questo caso infatti si possono gestire le comunicazioni via telefono o via internet, gestendo per esempio un sistema di segreteria telefonica. Statisticamente però l'uso maggiore della domotica viene ottenuto in ambito di sicurezza, attraverso l'inclusione di protezione antifurto o ancora sicurezza interna, avendo per esempio la possibilità di introdurre allarmi antincendio, anti-allagamento, o ancora inserendo sistemi di tele-soccorso e assistenza per persone anziane o disabili. Un altro ambito in cui poter inserire elementi domotici è quello della gestione dell'illuminazione e della gestione di apparecchi elettrici, qui possiamo trovare la gestione di luci interne ed esterne, lavastoviglie, cucine, forni e quant'altro. Un altra area funzionale gestibile attraverso dei sistemi di domotica è quella audio-video, avendo la possibilità di inserire dei sistemi per la diffusione sonora, come un home theater. Un fattore interessante da considerare in un sistema di domotica è chiaramente il fattore smart. Una funzionalità smart può essere ad esempio quella che, in contratti in cui si prevede una tassazione differente in funzione delle fasce orarie, tiene traccia del consumo e gestisce la diminuzione di carico in certe fasce rispetto che altre. Un'altra funzionalità del genere può essere la gestione intelligente del condizionamento ambientale attraverso

il rilevamento attuale dell'ambiente e l'uso di scenari preimpostati.

3.3 Requisiti di un sistema domotico

Un utente che vuole un sistema domotico può richiedere qualsiasi tipo di requisito, poi sarà compito del progettista approvarlo o meno. I requisiti sono molto personalizzabili, ma ce ne sono alcuni fondamentali a cui è impossibile rinunciare. Tra questi troviamo:

- **Facilità d'uso**: essendo diretto ad un pubblico vasto e possibilmente non professionale deve essere sicuro e non presentare pericoli per chi non è a conoscenza delle sue implementazioni.

- **Continuità**: deve comunque restare attivo in caso di guasti ed essere propenso alla riparazione utilizzando magari strumenti di tele-gestione.

- **Coesistenza**: è la capacità del sistema di domotica di riuscire a coesistere con altri sistemi già presenti. Questo fattore specifica anche la possibilità d'interazione tra i vari elementi di impianti diversi.

- **Aggiornamento**: un sistema domotico deve essere aperto all'inserimento di nuove tecnologie man mano che vengano rese disponibili. Esso infatti deve essere mantenibile e sviluppabile al più possibile.

- **Versatilità**: questo requisito valorizza la capacità del sistema di sapersi adattare alle diverse esigenze e funzioni richieste dall'utente, cercando di eliminare la necessità di modifiche hardware all'impianto, ma consentendo al sistema di riuscirsi ad adattare con l'introduzione di minime modifiche software.

- **Espandibilità**: questo è il requisito che permette al sistema di potersi adattare alle varie tipologie abitative che possono variare in dimensioni, zone ed a varie esigenze applicative.

3.4 Fasi della progettazione

1. Analisi delle esigenze dell'utente
2. Valutazione degli impianti
3. Definizione delle funzionalità
4. Mappa delle interazioni dell'utente
5. Scelta dei componenti

3.4.1 Analisi delle esigenze dell'utente

In questa fase bisogna studiare le caratteristiche del luogo e dell'abitazione e capire quali sono le reali esigenze e i bisogni degli utenti, per comprendere il sistema che occorre progettare. Si tratta di analizzare fattori tipo: se si tratta di una abitazione singola o in condominio, a che tipo di fenomeni climatici è soggetta, se sono possibili interruzioni di alimentazione da parte del distributore elettrico o ad esempio la presenza di persone anziane o disabili.

3.4.2 Valutazione degli impianti

Questa è la fase di analisi del giusto funzionamento degli impianti su cui verrà progettato il sistema di domotica, quali possono essere sistema di climatizzazione, motorizzazioni dei cancelli, porte e finestre e altre apparecchiature che verranno gestite dalla domotica.

3.4.3 Definizione delle funzionalità del sistema

In questa fase vengono scelte le modalità di funzionamento dei vari impianti da cui derivano le specifiche di definizione del sistema. Infatti le scelte come un sistema di sicurezza di antintrusione piuttosto che ambientale o di climatizzazione, vanno ad incidere fortemente su tutta la strutturazione del sistema.

3.4.4 Mappa delle interazioni con l'utente

Una volta definite le funzionalità previste si deve studiare come l'utente deve interagire con esse, questa fase dipende dall'utente finale che può essere di vari tipi. Se l'utente finale ha delle difficoltà motorie bisogna progettare l'interfaccia in modo che riesca ad utilizzare il sistema in modo il più semplice possibile, oppure l'utente potrebbe avere delle difficoltà visive, e allo stesso modo il sistema cambia, e così via.

3.4.5 Scelta dei componenti

La scelta dei componenti non solo segue la fase della scelta delle funzionalità, ma la scelta di questi varia anche in funzione delle modalità di interazioni con l'utente.

3.5 Risparmi energetici

Un sistema domotico è in grado di ricevere informazioni da sensori ed eseguire comandi di tipo on/off. Per questo attraverso l'uso di sistemi del genere è possibile limitare numerosi sprechi comuni quali ad esempio lasciare le luci accese. Un altro risparmio che un sistema domotico può garantire è quello dell'impostazione di scenari quali ad esempio "fuori casa" oppure "in vacanza". In fatti con la selezione di questi ambienti il sistema che regola ad esempio l'accensione dei riscaldamenti può ottimizzare le accensioni e gli spegnimenti. In Italia nel 2009 la quantità di case era pari a circa 28 milioni di unità, di queste una quota pari al 72% è costituita da alloggi di proprietà, il 20% da alloggi in affitto. Le nuove abitazioni in Italia nel 2007 sono state circa 300.000 mentre quelle ristrutturate 700.000. Gli impianti domotici installati in Italia nel 2008 sono stati circa 26.500 e di questi il 90% sono installati in nuove abitazione o ristrutturate ed il 10% in abitazioni esistenti[3]. Gli impianti di sicurezza installati in abitazioni nuove o ristrutturate sono invece

circa il 40% per cui il totale degli impianti di sicurezza nel 2005 è pari a circa 400.000 unità, ma il numero è in costante aumento.

Capitolo 4

Dispositivi

Per la gestione dell'IoT sono presenti sul mercato schede progettate su misura. Esse posseggono componenti fondamentali quali possono essere:

- **microcontrollori**: essi combinano processore, RAM e unità di storage in un solo chip. Sono componenti molto piccole. I microcontrollori hanno funzionalità molto limitate, alcuni modelli utilizzano componenti a 8 bit. In generale la RAM dei microcontrollori si misura in chilobyte. La piattaforma Arduino si basa sulla famiglia di chip per microcontrollori AVR ATmega di Atmel.

- **System-on-chip**: Questa è una via di mezzo tra un personal computer e un microcontrollore. In questa categoria possono rientrare piattaforme come Raspberry Pi. Queste piattaforme lavorano su memorie molto superiori a frequenze molto più elevate.

- **Ram**: Una maggiore quantità di RAM può consentire una maggiore flessibilità degli algoritmi eseguiti. Per eseguire protocolli di cifratura standard servono almeno 4KB.

- **Connessione di rete**: questo è un fattore molto importante per l'IoT. Un collegamento Ethernet è una soluzione immediata ed economica ma prevede il passaggio di cavi fisici. Le operazioni Wi-Fi non hanno il

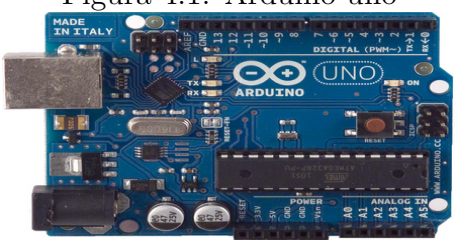
problema dei cavi ma necessitano di una configurazione e gestioni più complesse.

Tra i tanti dispositivi sul mercato i più comuni usati per la domotica sono: Raspberry Pi e Arduino.

4.1 Arduino

Il progetto Arduino nasce a Ivrea nel 2005. Un gruppo di docenti dell'I-DII (Interaction Design Institute Ivrea) per i propri studenti di design allo scopo di realizzare progetti interattivi. Prodotti simili ad Arduino erano già presenti sul mercato, ma erano solitamente prodotti difficili da utilizzare e costosi. Il gruppo decise allora di mettere insieme una scheda poco costosa, nella quale includere una connessione seriale per consentire una programmazione semplice. La decisione iniziale di rendere open source il codice e gli schemi elettronici permise che la scheda Arduino sopravvivesse alla scomparsa di IDII. La chiave del successo di Arduino fu la scelta di prediligere la semplicità alle prestazioni[5]. Sono molti i modelli di Arduino, tra questi un buon compromesso per un sistema di domotica è sicuramente il modello 1. Esso monta un microcontrollore ATmega328 e una presa USB. L'unità di storage è di 32 KB e la RAM di 2KB. Questo modello mette a disposizione 14 pin GPIO digitali e 6 analogici.

Figura 4.1: Arduino uno



4.1.1 Ide di sviluppo

lo sviluppo di Arduino avviene grazie all'impiego di un ide che è possibile scaricare gratuitamente dal sito ufficiale. Questo ide consente di implementare progetti molto complessi, ma il suo uso è studiato per la semplicità. E' inoltre possibile programmare anche senza l'ausilio di questo ide. Infatti grazie al toolset avr-gcc, una raccolta di programmi per compilare ed eseguire il codice, è possibile trasferire direttamente l'eseguibile sul chip.

4.1.2 Caricamento del programma

Dopo aver verificato e compilato correttamente il codice esso deve essere inserito nella memoria flash della scheda. Questa operazione è possibile grazie all'uso della porta usb. Una volta inserito il programma nella memoria basterà riavviare la board e il programma verrà eseguito in automatico.

4.1.3 Sistema operativo

Arduino non esegue di default un sistema operativo. Infatti esso al bootloader non fa altro che caricare il sistema che era stato precedentemente inserito nella memoria flash. Una volta che il programma è stato caricato esso verrà eseguito per tutto il tempo che l'Arduino resterà in funzione, a meno di blocchi imprevisti. In Arduino è possibile caricare un sistema operativo, solitamente distribuzioni RTOS(Real-Time Operating System) come FreeRTOS/DuinOS.

4.1.4 Linguaggio di programmazione

Il linguaggio di programmazione per Arduino è leggermente diverso dal C++, deriva dalla piattaformia Wiring e include librerie per la lettura e scrittura di dati rilevati dai pin I/O di Arduino. Il codice deve comprendere solo due routine:

- **setup()**: routine da eseguire una sola volta, quando si accende la scheda. Viene utilizzata per impostare la modalità dei pin I/O di input e di output.

- **loop()**: questa è la routine da ripetere ciclicamente per tutto il tempo che la scheda Arduino rimane accesa.

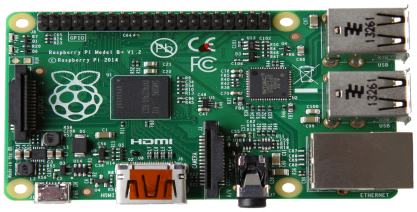
4.1.5 Considerazioni sull'hardware

Arduino mette a dispoizione molti pin GPIO, ogni pin è accompagnato con degli identificativi segnati direttamente sulla scheda. Le funzionalità dei pin cambiano in base al modello prescelto. In generale, sono presenti output di alimentazione a 5V oppure a 3,3V uno o più collegamenti a massa, pin digitali e analogici. L'alimentazione può provenire da Usb, soluzione molto comune in fase di sperimentazione, oppure è possibile alimentare esternamente la scheda. Oltre alle schede standard ve ne sono altre più specifiche che possono essere identificate come estensioni. Queste schede sono dette shield.

4.2 Raspberry Pi

Questo progetto nasce dall'idea di Eben Upton, che voleva realizzare un computer piccolo, poco costoso e progettato per essere programmato e per fare esperimenti. Il progetto prese ispirazione da un tentativo precedente che la BBC aveva fatto, con il "BBC Micro", per migliorare la conoscenza di informatica nel Regno Unito. I Raspberry Pi, considerando i vari modelli, hanno sempre mantenuto un prezzo intorno ai 25 pounds, in pratica un prezzo molto simile a quello di Arduino. La differenza tra i due progetti però è notevole, infatti nel caso di Raspberry Pi abbiamo un computer a tutti gli effetti, con a bordo un vero sistema operativo e dei componenti hardware nettamente superiori a quelli di Arduino.

Figura 4.2: Raspberry Pi B+



4.2.1 Sistema operativo

I sistemi in grado di lavorare con schede Pi sono molti, tra i più utilizzati troviamo:

- **Raspbian**: rilasciata da Raspbian pi foundation, che è una distribuzione basata su debian.
- **Occidentalis**: è la distribuzione Raspbian personalizzata per Adafruit. A differenza di raspbian questa non prevede una iniziale configurazione delle impostazioni di uso da remoto. Essa infatti è già configurata per funzionare con una connessione remota.
- **Xbian**: si tratta di una distribuzione basata su Raspbian per gli utenti che vogliono utilizzare Raspberry Pi come media center.
- **Raspbmc**: è una distribuzione Linux minimale basata su Debian che porta XBMC su Raspberry Pi.
- **OpenELEC**: sta per Open Embedded Linux Entertainment Center. Si tratta di una piccola distribuzione Linux costruita da zero come piattaforma per trasformare un computer in un completo media center XBMC.
- **QtonPi**: nasce esclusivamente per combinare il framework Qt 5 e Raspberry Pi. L'obiettivo è quello di sviluppare i fattori per gli sviluppatori su piattaforma Qt 5. E' un framework ampiamente utilizzato per lo sviluppo di software applicativo con un'interfaccia utente grafica e anche per quello di programmi a riga di comando.

A differenza di Arduino, ha bisogno di una fase di configurazione prima di essere utilizzato. Infatti prima di ogni altra cosa va installato il sistema operativo su scheda sd. In seguito ci sono le configurazioni che si potrebbero voler fare al sistema operativo[4].

4.2.2 Linguaggio di programmazione

Ancora una volta Raspberry Pi e Arduino sono nettamente diversi. Prima di poter fare un programma su Raspberry Pi bisogna selezionare che linguaggio di programmazione usare e in che ambiente si vuole programmare. Il consiglio della Raspberry Pi Foundation è quello di utilizzare python come linguaggio per l'apprendimento. In effetti la "Pi" deriva proprio da Python.

4.2.3 Considerazioni sull'hardware

La scheda Raspberry Pi B+ possiede 40 pin. Di essi però nessuno è analogico. Sono inclusi output a 5V oppure a 3,3 V, mentre i ping GPIO ammettono solo tensioni di 3,3V. La scheda Pi non è protetta dalle sovratensioni, perciò è molto facile rischiare di rovinare tutto alimentandolo con una tensione sbagliata.

Capitolo 5

Domitica

Domitica è un progetto di un sistema di domotica implementato attraverso l'uso di una scheda Raspberry Pi. Il modello utilizzato è il Raspberry Pi B+. Esso possiede 40 pin di controllo. Essendo quindi limitato il numero di pin utilizzabili, per ogni stanza o punto di interesse viene impiegato un Raspberry Pi che gestisce un numero definito di elementi, quali essi possono essere relè, sensori di temperatura, di luce e molti altri. Questo progetto si basa sull'utilizzo quindi di vari Raspberry Pi che comunicano tra di loro e con dei client android. Gli elementi principali sono: Raspberry Pi, Client android, Mosquitto ed Mqtt;

5.1 Funzionamento Generale

5.1.1 Mosquitto e Mqtt

Per l'implementazione della comunicazione tra i Raspberry Pi e i client è stato usato il protocollo mqtt introdotto al capitolo 2. Lo scambio di messaggi viene gestito da Mosquitto, un broker message mqtt.

5.1.2 Raspberry Pi

L'unica configurazione prevista per ogni Raspberry Pi è quella del file /etc/hostname in cui viene impostato il nome della stanza o del reparto che quel Raspberry Pi andrà a gestire. Per esempio il Raspberry Pi che si occuperà della cucina avrà segnato in quel file una stringa fatta così: "domotica-cucina". Tra tutti i Raspberry Pi non c'è un master ma sono visti tutti come delle eguali unità che gestiscono una data parte, fatta eccezione per la prima connessione. Infatti quando un client si connette per la prima volta farà riferimento ad un Raspberry Pi che sicuramente si trova nella rete, ad esempio il Raspberry Pi "domotica-salotto". Nel momento in cui i Raspberry Pi si accendono fanno una scansione della rete cercando elementi con un nome di dominio del loro tipo, così facendo andranno a compilare un file di configurazione che si trova in ogni singolo Raspberry Pi, in cui saranno segnati tutti gli elementi della domotica presenti nella rete con i loro relativi indirizzi. Oltre alla configurazione della rete, ogni Raspberry Pi contiene al suo interno anche un altro file di configurazione, che indica la composizione della stanza o punto di interesse relativo. Ovvero le informazioni relative ai nomi degli interruttori a cui si fa riferimento, e i sensori di cui quella stanza è attualmente fornita.

5.1.3 Client Android

Come detto in precedenza nel momento in cui un client android avvia l'applicazione viene instaurato un primo contatto con un Raspberry Pi principale che servirà esclusivamente a restituire al client i dati necessari per la costruzione della propria interfaccia. I dati in questione sono appunto le informazioni necessarie dei vari Raspberry Pi che si trovano nella rete con i loro indirizzi, in modo da poter fare delle richieste separate ad ogni Raspberry Pi.

5.2 Dati tecnici

5.2.1 Raspberry Pi

La prima operazione effettuata da ogni Raspberry Pi all'avvio è la configurazione. La configurazione viene effettuata attraverso il listener che sarà attivo per tutta la sessione del programma. Esso viene impostato attraverso la funzione:

```
set_listener(char * nome)
```

la prima volta che viene chiamata questa funzione, l'argomento passato è la stringa ".", che rappresenta il topic di configurazione di ogni Raspberry Pi. Questa funzione svolgerà due compiti: quello di leggere le attuali configurazioni, e quello di impostare il loop infinito. Per la configurazione abbiamo il seguente frammento:

```
read_configuration();
read_interface();
set_connection();
```

con le prime due funzioni vengono lette, se presenti, eventuali configurazioni precedenti. Con le informazioni relative le precedenti configurazioni vengono riempiti due array che contengono uno le stringhe identificative delle connessioni e il secondo le stringhe che identificano la struttura della singola stanza. Attraverso la funzione set_connection(), viene fatta la scansione della rete in cui si cercano hostname con un nome compatibile con quello del sistema. Una volta trovati si verificata la loro presenza all'interno dell'array contenente le connessioni, nel caso in cui un indirizzo non dovesse trovarsi in questo array, allora esso viene inserito nell'array e scritto sul file di configurazione della rete. Dopo questa fase di configurazione viene istanziata la libreria libmosquitto-dev utilizzata per implementare e comunicare con il broker.

```
mosquitto_lib_init();

mosq = mosquitto_new(NULL,clean_session,name);

mosquitto_connect(mosq,host,port,keepalive);

mosquitto_connect_callback_set(mosq,my_connect_callback);
mosquitto_message_callback_set(mosq,my_message_callback);

mosquitto_loop_forever(mosq,-1,1);
```

Dopo aver eseguito la configurazione è necessario impostare il loop con il quale il Raspberry Pi resterà attivo. In un primo momento si avvia la libreria attraverso la mosquitto_lib_init(), poi si genera una struttura di tipo struct mosquitto, attraverso la funzione mosquitto_new. Dopo aver instanziato la struttura si procede con la connessione. In seguito si settano le funzioni di callback relative ad una connessione avvenuta ed a un messaggio ricevuto. Con queste funzioni si andranno a specificare gli eventi da scatenare in queste specifiche occasioni. Dopo aver settato le funzioni di callback si imposta il loop attraverso la funzione mosquitto_loop_forever. Una volta impostata la connessione, attraverso la funzione mosquitto_loop_forever (mosq,-1,1) il Raspberry Pi si pone in ascolto. La comunicazione tra client Android e Raspberry Pi come consono al protocollo mqtt citato in precedenza avviene attraverso l'invio e la ricezione di semplici stringhe. Per questo una volta settato il listener sul Raspberry Pi i messaggi a cui esso risponde sono i seguenti:

- **Android:*idconnessione***: per idconnessione si intende un id univoco per ogni connessione che viene precedentemente creato dal client e poi inviato. A questo messaggio il Raspberry Pi risponde creando un nuovo topic con questo nome, in modo tale da poter scambiare messaggi privati esclusivamente con il client in questione.

- **/home/*nomestanza***: questo messaggio viene seguito dal nome di una particolare stanza, questa funzionalità consente al client di conoscere l'indirizzo di una stanza in particolare.

- **Interfaccia**: a questa stringa il Raspberry Pi risponde, prima effettuando un lettura sul file di configurazione dell'interfaccia e poi concatenando tutti i nomi delle interfacce in una sola stringa da restituire al client.

- **rooms-android**: allo stesso modo del messaggio precedente il Raspberry Pi restituisce una stringa con i nomi di tutte le stanze/connessioni che si trovano nella configurazione attuale.

- **on/*nomeinterruttore***: alla ricezione di questo comando il Raspberry Pi cerca nel file locale di configurazione una interfaccia con il nome specifico. Nel file di configurazione si trova oltre al nome il relativo pin sulla porta GPIO del Raspberry Pi. Quindi una volta ottenuto il pin, può essere acceso con una funzione di libreria.

- **off/*nomeinterruttore***: allo stesso modo della funzione precedente questa volta il segnale inviato sarà di spegnere il pin.

5.2.2 Android:*idconnessione*

Quando il Raspberry Pi riceve questo comando viene creato un nuovo thread in questo modo:

```
pthread_create(&listner,NULL,set_listner,topic_name);
```

in cui viene creato un thread che richiama la funzione che viene chiamata all'inizio per creare il primo listener, che è configurato su un topic standard costituito dalla stringa.

5.2.3 Interfaccia

Quando il Raspberry Pi riceve questo comando vuol dire che il client ha richiesto l'implementazione della sua interfaccia locale. Attraverso la funzione

```
get_android_interface(interfaccia);
```

il Raspberry Pi ricerca all'interno del suo file di configurazione dell'interfaccia tutte le informazioni. Mentre legge queste informazioni si preoccupa di popolare una array che in seguito sarà usato per verificare la presenza delle interfacce.

5.2.4 Rooms-android

In modo speculare all'interfaccia c'è la possibilità che il client richieda la strutturazione della rete, con le singole stanze e i loro relativi indirizzi. Quando il Raspberry Pi riceve questo messaggio viene chiamata la funzione:

```
get_android_rooms(rooms);
```

allo stesso modo del dato precedente il Raspberry Pi ricerca all'interno del file di configurazione della rete e riempie un array speculare in cui inserire i nomi di tutte le stanze.

5.2.5 /home/*nomestanza*

Questo è il messaggio che il client invia al Raspberry Pi quando vuole ricevere l'indirizzo di una stanza. Per *nomestanza* si intende la stanza di cui ho bisogno di ottenere l'indirizzo. Una volta ricevuto questo messaggio il Raspberry Pi attraverso la funzione:

```
get_address_from_room(stanza,addr);
```

ottiene l'indirizzo della stanza attraverso una ricerca sull'array precedentemente creato e restituisce il nome della stanza al client.

5.2.6 on/*nomeinterruttore*

In modo simmetrico questa è la richiesta del client di accendere un interruttore. In questo caso il client non si aspetta però nessun messaggio di ritorno. Quando il Raspberry Pi riceve questo messaggio ricerca nell'array precedentemente creato relativo alle interfacce il nome di quella interfaccia attraverso la funzione:

```
get_command_from_name(tmp_interruttore,command);
```

successivamente viene ricavato il pin relativo all'interruttore dal file di configurazione dell'interfaccia. Una volta fatto questo esso accende il pin relativo.

5.2.7 off/*nomeinterruttore*

In modo completamente identico alla funzione precedentemente spiegata, questa volta una volta ricavato il pin relativo all'interruttore esso viene spento.

5.2.8 Client Android

Nel momento in cui parte l'applicazione il client android invia al Raspberry Pi un messaggio del tipo "android:*idconnessione*", dove *idconnessione* era stato precedentemente da lui creato, sul topic di configurazione ".". Una volta fatto questo viene quindi creato un nuovo topic per una comunicazione privata tra client e server. A questo punto il server, ovvero il Raspberry Pi, invia al client le informazioni relative alle stanze, e in questo modo il client costruisce la sua interfaccia.

Nel momento in cui il client seleziona una stanza viene inviato un messaggio di tipo "/home/*nomestanza*", quindi viene restituito l'indirizzo della

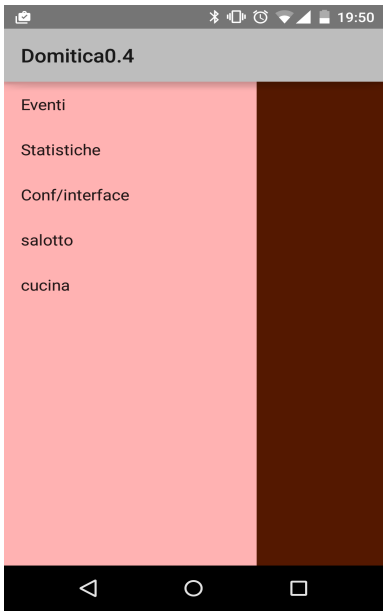


Figura 5.1: Viene riempito il menu laterale attraverso i dati ricevuti dal messaggio “rooms-android”

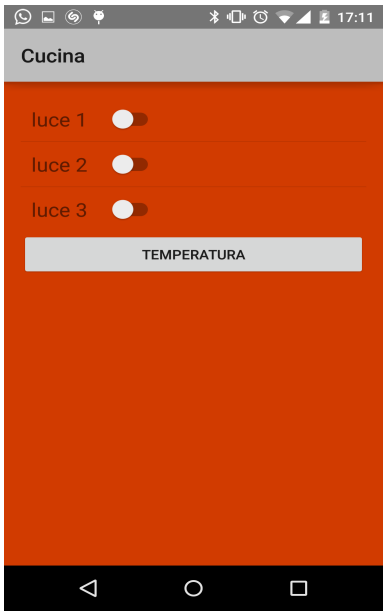


Figura 5.2: Viene costruito il layout di ogni singola stanza attraverso i dati ricevuti dal messaggio “interfaccia’

stanza richiesta al client che può connettersi ad essa. Una volta connesso il client richiede l'interfaccia attraverso il comando "interfaccia" e in questo modo costruisce l'interfaccia. Con uno switch il client potrà selezionare l'interruttore da accendere o spegnere ed effettuare operazioni di questo tipo su di esso.

Conclusioni

Dagli studi effettuati evince che l'IoT è in pieno sviluppo e sta per esplodere in tutto il mondo. Nelle nostre vite oramai oggetti intelligenti sono già da molto presenti, essi sono anche connessi tramite internet, ma questo non basta per essere definito IoT. Infatti l'attuale conoscenza di questi oggetti da parte degli utenti e l'attuale strutturazione di internet non permette ancora di rendere totale giustizia all'IoT. Il protocollo MQTT risulta utilizzabile e funzionante, ma basandosi sul protocollo IP risulta ancora essere troppo oneroso dal punto di vista dell'utilizzo di risorse. Quando saranno chiare le strutture dell'IoT e si avrà un grado di compatibilità tra i vari sistemi tale che anche i prezzi possano rimanere relativamente bassi allora potremmo davvero dire di trovarci di fronte all'IoT.

Bibliografia

[1] Francis daCosta , "*Rethinking the IoT a scalable approach to connecting everything*", Appress, USA, 2014

[2] Larry Petersn e Bruce Davie , "*Computer networks: a system approach* ", Elsevir, 360 Park Avenue South New York, USA, 2012

[3] Giuseppe Gustavo Quaranta, "*La domotica per l'efficenza energetica delle abitazioni*", Maggioli editore, Santarcangelo di Romagna, 2013

[4] Pier Calderan, "*Raspberry Pi - Guida al computer più compatto del mondo*", Apogeo, Milano, 2013

[5] Adrian McEwen e Hakim Cassimally, "*Designing IoT*". John Wiley & Sons, USA, 2013

[6] Oasis standard (Ottobre 2014), *MQTT Version 3.1.1* , http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[7] Antonino Caffo (Gennaio 2015)" *Cos è l Internet delle cose?*" http://www.lastampa.it/2014/08/26/tecnologia/internet-degli-oggetti-l-delle-persone-non-sa-cosa-sia-nZwZW48lypEmHcK92EGyZP/pagina.html

[8] Paolo Mongiovì (Febbraio 2015) "*Il mercato della domotica in italia: stato e prospettive*", http://www.domotica.it/2011/07/il-mercato-della-domotica-in-italia-stato-e-prospettive/.