Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Elettronica, Informatica e delle Telecomunicazioni

Protocolli applicativi per il mondo Internet Of Things

Relatore:
Prof. FRANCO CALLEGATI

Presentata da:
DIEGO SOZIO

III Sessione
Anno Accademico
2017/2018

Ceci n'est pas une pipe.
René Magritte

Indice

Introduzione

Nel mondo odierno, grazie ad internet e il suo impressionante sviluppo, avvenuto negli ultimi anni, vengono forniti all'utilizzatore finale strumenti che solo all'inizio del ventunesimo secolo erano inimmaginabili. L'utente finale oggi é ormai fortemente integrato nel mondo digitale e lo sará sempre piú vista la futura evoluzione portata dall'Internet of Things. Risulta spontaneo quindi chiedersi com'é possibile realizzare una forte interazione tra i vari sistemi eterogenei e fornire un servizio che sia il piú naturale e semplice possibile tanto da realizzare una interazione quasi senza latenza. Questa valutazione ha valenza sia che ragioniamo in termini di servizi non fondamentali magari accessori(per esempio l'applicazione messanger di Facebook), sia che spostiamo l'attenzione su un ambito industriale, in cui risulta essere fondamentale fornire strumenti che permettono al processo produttivo di realizzare una comunicazione sicura e garantita funzionante in ogni possibile casistica possibile. Per affrontare questa problematica sono stati realizzati standard per la comunicazione specializzati per specifici ambiti, vista la grande eterogeneità delle applicazioni IoT. In questo elaborato di tesi saranno trattati e analizzati tre standard tra i piú utilizzati, nello specifico OPC-UA, MQTT e LONWORKS.

Capitolo 1

Sommario

Negli ultimi anni grazie all'abbattimento dei costi, riduzione dell'hardware e sviluppo di nuovi protocolli è osservabile l'incremento del numero di dispositivi con capacità di accedere a internet, tanto che si stima il raggiungimento di almeno tre dispositivi connessi a persona nel breve periodo. Questa casistica non prende però in esame solo i classici dispositivi con capacità di connessione ad internet (cablata o wireless) ma anche i dispositivi nascenti che rispondono al nome e fanno parte della categoria IoT (Internet Of Things) stiamo quindi assistendo ad una delle più grandi rivoluzioni del mondo dell'informazione che ha come obbiettivo finale quello di fornire all'utente una serie di nuovi servizi e informazioni. Questa rivoluzione non coinvolge però solo l'utente finale ma anche la rete internet come la conosciamo che in virtù di ciò sta progredendo più intensamente oggi piuttosto che negli anni passati. L'architettura originaria di internet, come fù concepita da Tim Berners-Lee nel 1996, non prevedeva un probabile incremento dei dispositivi così alto e questa casistica ad oggi rischia di mettere in discussione quella che è la corrente infrastruttura del WWW. Si è venuta quindi a creare la necessità di sviluppare una struttura internet che sia in grado di gestire un alto numero di dispositivi che scambiano informazioni tra loro oltre a permettere la comunicazione tra i singoli individui. Detto ciò nasce da subito un primo problema che riguarda il come questi dispositivi riescano a comunicare tra loro ? Per dare una risposta a questo primario quesito dobbiamo da prima metterci nell'ottica che la comunicazione di rete non è visualizzabile come una black-box ma piuttosto una comunicazione per essere definita tale deve rispondere a determinati e ben definiti passi fondamentali. Come risulta da subito chiaro negli anni molte entità, sia organizzazioni internazionali che singoli sviluppatori, hanno sviluppato e in seguito messo in atto nuovi meccanismi di comunicazione portando ad oggi ad avere una ampia scelta di protocolli a disposizione. Risulta quindi chiaro che da una prima occhiata tutti questi protocolli non sono pensati e implementati per poter interoperare tra loro, visto che si basano su concetti, idee differenti, quindi come risolviamo il problema di una differente concezione della comunicazione ? La risposta a questa seconda domanda è cercare di realizzare N standard riconosciuti da un ente accreditatore, l'Organizzazione Internazionale

per la Normazione (ISO), che permettano quindi di coprire una parte importante di quelle che sono le comunicazioni di rete così da portare i produttori a produrre dispositivi di marchi differenti ma che integrano la stessa metodologia di comunicazione così da creare interoperabilità tra gli stessi. L'obiettivo di questo elaborato di tesi è proprio andare ad analizzare e studiare tre protocolli, tra i più usati e spiegarne le varie funzionalità. Ora che abbiamo risposto ad alcuni iniziali quesiti andiamo a contestualizzare quello che è lo scenario applicativo dei protocolli che andremo ad analizzare, possiamo fare una prima categorizzazione dei protocolli in funzione delle singole peculiarità e obiettivi prefissati cioè una comunicazione sicura e affidabile piuttosto che una comunicazione su banda ridotta e con forti perdite di pacchetto. Come risulta chiaro questa prima categorizzazione ci permette di operare con i singoli protocolli in particolari settori, esempio se dobbiamo realizzare un sistema che permetta l'esecuzione di un operazione chirurgica in remoto, dobbiamo scegliere un protocollo che per natura ci garantisca una comunicazione efficace e con ridottissima perdita di pacchetti ma che allo stesso tempo abbia una latenza che permetta la definizione, da parte dell'utilizzatore, di comunicazione molto vicina alla real time, diversamente in altri ambiti possiamo permetterci l'utilizzo di un protocollo meno affidabile che però ha tra le sue peculiarità la capacità di funzionare su hardware a basse prestazioni e di conseguenza basso impatto energetico.

1.0.1 Il Mondo IoT

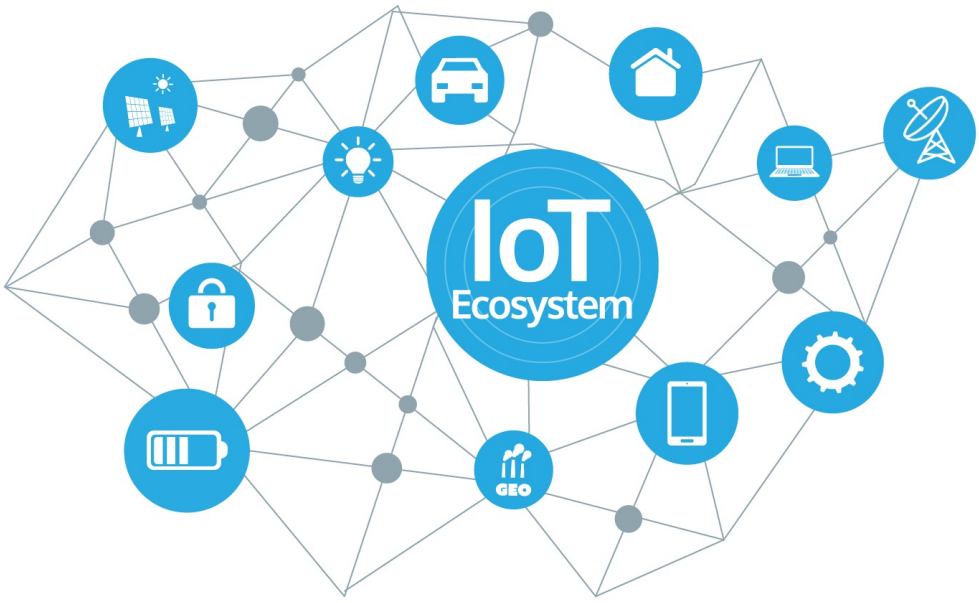

Fig.1.1

Analizziamo ora cosa si intende comunemente parlando in linguaggio natuale di IoT, quando si parla di internet delle cose si fa riferimento a qualsiasi tipo di oggetto intelligente, quindi si può parlare di un computer, sensore ma anche di un'automobile come di una macchina da caffé. Tutti questi oggetti, associati al mondo IoT, hanno la possibilità di comunicare, tramite wireless, grazie alla presenza di chip intelligenti installati all'interno di essi. Grazie a questa astrazione possiamo quindi immaginare, non più una rete internet formata da soli computer, cellulari o tablet ma una rete globale in cui qualsiasi oggetto, anche il più semplice risulta connesso in ogni momento e in ogni parte del globo. Passando alla definizione tecnica di Iot definiamo una rete di oggetti interconnessi con la capacità di comunicare tra loro in due modalità:

- **M2M** Machine to Machine
- **M2H** Machine to Human

I dispositivi che fanno parte della rete di oggetti sono chiamati *smart objects* o *smart things* che a differenza dei normali dispositivi possiedono la capacità di poter interagire all'interno del sistema di comunicazione in cui sono inseriti con quindi un ruolo attivo. Questi dispositivi sono composti da specifiche caratteristiche:

1. Sono oggetti fisici, caratterizzati da un costo
2. Hanno risorse hardware molto ridotte
3. Sono associati ad un indirizzo univoco ID e un nome in linguaggio naturale, quindi individuabili in rete e iniettivamente individuano altri dispositivi
4. Possono influenzare la realà se dotati di attuatori (motori, robot, pistoni)

La concretizzazione del concetto IoT è stata resa possibile grazie all'introduzione di tecnologie abilitanti come **WSN(Wireless Sensor Networks)** utilizzata in particolar modo per operazioni inerenti il sensing. I nodi di una WSN sono rappresentati da sensori ambientali con lo scopo di rilevare determinati dati e sono applicati in vari ambiti come militare, medico, investigativo. Un'altra tecnologia abilitante per l'IoT è rappresentata dall'**RFID(Radio Frequency IDentification)** nativamente utilizzato per il processo di identificazione e a seguire anche per operazioni di sensing. un sistema RFID è costituito da *tag*, *reader* e un sistema di *back-end* che permette di associare ad ogni ID l'oggetto fisico corrispondente e eventuali informazioni connesse all'oggetto stesso. Questa tecnologia è molto utilizzata nell'ambito della geolocalizzazione e tracciamento di colli. Un'ultima tecnologia interessante è **NFC(Near Field Communication)**, questa nasce partendo dai sistemi RFID e la principale differenza tra le due sta nella distanza che è possibile coprire, in questo caso si attesta nell'intorno di pochi centimetri, (indicativamente 10cm). NFC è fortemente utilizzato nella condivisione di informazioni tra i vari dispositivi. L'eterogeneità di dispositivi e di applicazioni, in un sistema IoT,

rende necessaria la presenza di un software detto **Middleware** che fa da collante tra i dispositivi e le applicazioni presenti in essi.

1.0.2 Comunicazione IoT

Il motivo principe della non applicabilità dei protocolli classici nell'ambito dell'IoT è dipeso dall'eccessiva robustezza di questi ultimi che inevitabilmente porta ad un eccessiva quantità di risorse minime, si vede quindi necessario l'utilizzo di nuove tecnologie che non vadano a sovraccaricare lo stack protocollare rendendo inutilmente complessi apparecchi semplici come sensori o macchine del caffè. Una probabile soluzione a questa problematica è lo sviluppo di protocolli che si basano sull'idea "orientati a ricevere" con questa definizione andiamo a specificare il macro-comportamento osservabile dell'oggetto che se riceve il dato atteso allora questo lo analizza, diversamente il sistema resta inalterato. L'utilizzo di questa tipologia di interazione rappresenta la concretizzazione di quella che è l'idea base dell'IoT cioè la gestione di numerosi dispositivi collegati in rete che però nel loro insieme non vanno a sovraccaricare l'intero apparato portandolo ad un collasso. Le classiche connessioni Wan e Wireless richiedono alta ampiezza di banda e per definizione alto costo, inoltre eseguono numerosi controlli inerenti l'integrità del messaggio con lo scopo di ridurre al minimo la necessità di ritrasmissioni. Diamentralmente i dispositivi IoT dovranno basarsi sulle mere connessioni wireless, la quantità dei dati da inviare sarà estremamente ridotta rispetto alle trasmissioni classiche ed inoltre i dispositivi saranno progettati per funzionare anche in caso di interruzione della trasmissione. In virtù di proprio questa autosufficienza per i dispositivi IoT non è più necessario eseguire test di integrità e di conseguenza vanno anche ridefinite quelle che sono le informazioni che vengono inviate nei pacchetti trasmessi che devono fornire solo il minimo indispensabile. Si configura necessario più che mai oggi la necessità di sviluppare e standardizzare questi nuovi prodotti così da poter creare da subito un ecosistema pronto all'applicazione e quindi scongiurare, nel lungo periodo, il collasso della rete internet conosciuta fino ad oggi. I protocolli che andremo a trattare hanno differenti applicazioni, alcuni diretti al mondo business mentre altri derivano direttamente da attività di ricerca; OPC-UA rappresenta ad oggi lo standard de facto per quanto concerne la trasmissione di dati nell'ambito industriale, LONWORKS viene ampiamente utilizzato invece nella gestione di reti elettriche, sia nazionali che internazionali mentre MQTT rappresenta il protocollo di riferimento per quanto concerne la messaggistica web-based e trova applicazione nella chat Messanger del colosso Facebook.

Capitolo 2

Header Protocollare

Prima di affrontare lo studio dei protocolli risulta essere utile descrivere quello che è lo standard de facto a livello di header per i singoli protocolli. Iniziando la trattazione sopra anticipata, possiamo dare vita ad un analisi strutturata per protocollo.

2.0.1 Header OPC-UA

Analizzando in prima istanza OPC-UA, possiamo verificare che per questo protocollo è disponibile un header fisso, variabile in funzione della richiesta, se Request o Response, che si formano come segue**(Fig.2.1)**:

Proprietà pubbliche	Descrizione
NodeId AuthenticationToken	Identificatore utilizzato per assegnare la chiamata del servizio alla sessione precedentemente stabilita tra client e server.
DateTime Timestamp	Istante di tempo in cui il client ha inviato la richiesta.
uint RequestHandle	Identificativo assegnato alla chiamata del servizio, definito dal client.
uint ReturnDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice *status code*.
String AuditEntryId	Identifica il client o l'user che ha iniziato l'azione; usato in caso di auditing.
uint TimeoutHint	Timeout settato per la chiamata del servizio lato client nell' UA Stack; quando termina, il server annulla la chiamata ricevuta.

Fig.2.1

Mentre per response abbiamo un header come segue **(Fig.2.2)**::

Proprietà pubbliche	Descrizione
DateTime Timestamp	Istante di tempo in cui il server ha inviato la risposta.
uint RequestHandle	Identificativo assegnato alla chiamata di servizio definito dal client.
StatusCode ServiceResult	Classe definita dallo standard per descrivere il risultato della chiamata del servizio. Contiene un uint a 32 bit in cui i 16 più significativi rappresentano il valore numerico dell'errore ed i
	rimanenti contengono informazioni aggiuntive. In particolare, i 2 bit più significativi si riferiscono all'attendibilità del risultato (00=Good, 01=Uncertain, 10=Bad, 11=not used).
DiagnosticInfo ServiceDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice status code.

Fig.2.2

2.0.2 Header MQTT

Il protocollo MQTT prevede un Header che è formato da due componenti, abbiamo un **header fisso** che presenta uno schema standard, come riportato **(Fig.2.2)**:

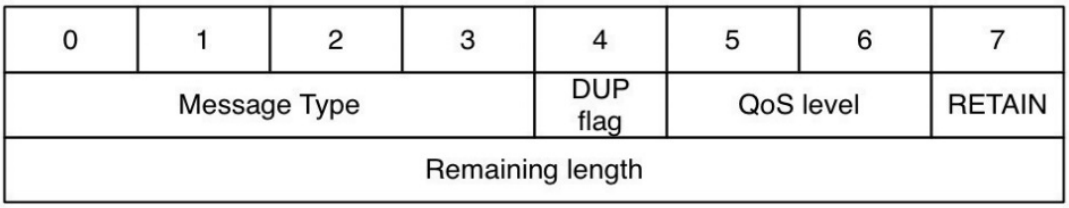

Fig.2.2

i campi associati sono :

1. **Message Tipe:** 4 bit
    - *Reserved (0) : riservato*
    - *CONNECT (1) : un client richiede la connessione ad un server*
    - *CONNACK (2) : ACK sulla CONNECT*
    - *PUBLISH (3) : pubblica un messaggio*
    - *PUBACK (4) : ACK sulla PUBLISH del messaggio*
    - *PUBREC (5) : messaggio PUBLISH ricevuto*
    - *PUBREL (6) : messaggio PUBLISH rilasciato*
    - *PUBCOMP (7) : messaggio di PUBLISH completato*
    - *SUBSCRIBE (8) : richiesta di SUBSCRIBE dal client*
    - *SUBACK (9) : ACK della richiesta di SUBSCRIBE*

- *UNSUBSCRIBE (10) : richiesta di UNSUBSCRIBE dal client*
- *UNSUBACK (11) : ACK della richiesta di UNSUBSCRIBE*
- *PINGREQ (12) : richiesta di PING*
- *PINGRESP (13) : risposta al PING*
- *DISCONNECT (14) : indica che il client si vuole disconnettere*
- *RESERVED (15) : riservato*

2. **DUP :** Flag utilizzato quando il client o il server vuole ritrasmettere un messaggio Publlish, Pubrel, Subscribe o Unsubscribe. Si applica quando il valore di QoS ¿ 0, se impostato a 1 si ha nell'header un Message ID.

3. **QoS level:** : indica il livello di delivery dei messaggi:

   - *0 At Most Once :*Invio senza riconoscere la consegna
   - *1 At Least Once :*Riconoscere la consegna
   - *2 Exactly once :*Consegna assicurata
   - *3 riservato*

4. **RETAIN :** Flag utilizzato solo per i messaggi ti tipo Publish.

5. **Remaining length:** Rappresenta il numero di byte che rimangono all'interno del messaggio corrente.

   Mentre per quanto riguarda **header variabile**, esso si forma come segue. L'header varibile può essere formato da differenti campi:

   - *Protocol name:* Nome del protocollo di un messaggio MQTT CONNECT codificato in UTF
   - *Protocol version (8 bit):* versione del protocollo di un messaggio MQTT CONNECT

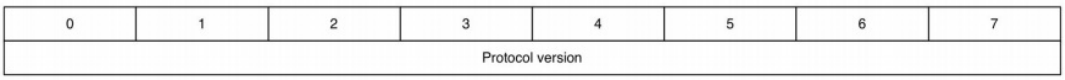

Fig.2.3

- *Flag per i messaggi CONNECT*

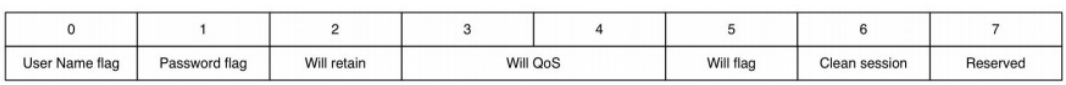

Fig.2.4

- *Keep Alive Timer (16 bit):* Si trova nell'header variabile di un messaggio CONNECT

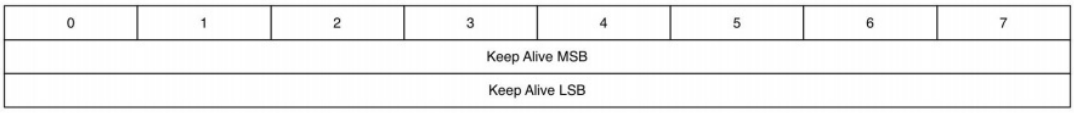

Fig.2.5

- *Connect Return Code:* Codice che viene restituito nella variabile header di un messaggio CONNACK

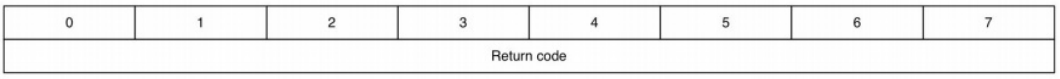

Fig.2.6

2.0.3 Header LonWorks

Per quanto riguarda l'ultimo protocollo che si andrà a trattare presenta un header di tipologia fissa. I nodi LonTalk comunicano tra loro inviando dei datagrammi contenenti le informazioni da trasportare, datagrammi riportati come segue in figura

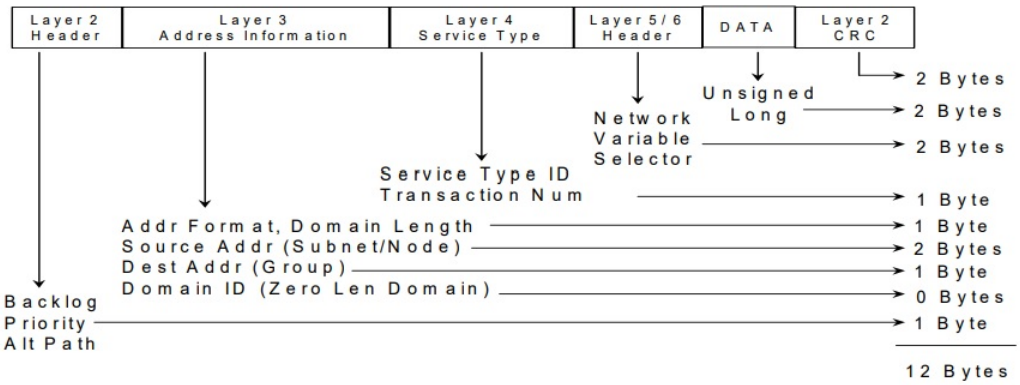

Fig.2.7

Il campo dati può avere una lunghezza massima pari a 228 byte, mentre il campo ID dominio può contenere 0, 1, 3 o 6 byte. In generale la lunghezza è pari a 10 byte, la gestione dei frame viene eseguita dalla CPU di accesso multimediale e dalla CPU di rete e l'applicazione deve solo fornire il contenuto del campo DATI, contenuto referenziato all'interno il programma applicativo come variabile di rete o campo di messaggio.

Capitolo 3

OPC-UA

Questo primo protocollo deriva della definizione dello standard OPC Data Access, definizione ad opera della OPC Foundation, che fu concepito per definire interfacce client-server per l'accesso ai dati di processo, permettendo così una semplice e completa integrazione dei vari dispositivi con funzioni di automazione prodotti da differenti produttori.Questa prima soluzione però, specialmente in ambito industriale non permise di usufruire integralmente dei vantaggi offerti dalla tecnologia OPC-DA, principalmente a causa delle limitazioni dei meccanismi offerti per l'accesso remoto.Il protocollo OPC-DA fu il pregressore dello standard OPC Unified Architecture (OPC-UA) questo standard nasce con l'intento di creare un sostituto degli standard già conosciuti andando però a preservare le varie funzionalità e garantendo alta efficienza di funzionamento. Questo nuovo protocollo soddisfa la necessità di interfacce platform-independent e permette la creazione di modelli estendibili con la funzione di descrivere sistemi complessi.

3.1 Standard Structure

Per garantire i vari requisiti che si prefissa, lo standard è strutturato a strati **(Fig.3.1)** dove le principali componenti equivalgono al protocollo di trasporto e al modello dei dati. Nel protocollo di trasporto vengono definiti due differenti casi d'uso :

1. un protocollo TCP binario per la comunicazione internet ad alta performace
2. un protocollo Web Services per la comunicazione internet Firewall-Friendly

Il modello dei dati definisce invece le regole e gli elementi minimi per fornire un valido modello informativo.

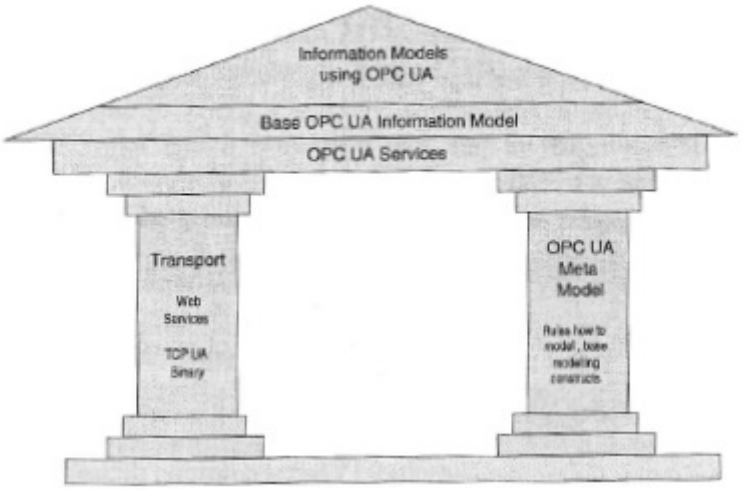

Fig.3.1

Nella definizione strutturata di OPC-UA i servizi OPC (OPC-UA services) costituiscono interfacce tra server e client, definiti distintamente come fornitori di modelli infomativi e "consumatori" di tali modelli, la comunicazione tra queste due entità avviene tramite i meccanismi di trasporto per lo scambio dei dati. Uno dei principi base del protocollo OPC-UA dichiara che un client può accedere alla più piccola porzione di dati di un sistema complesso senza che questo sia a conoscenza di come è strutturato l'intero sistema informativo. La suddivisione del modello informativo è strutturata in 4 sezioni **(Fig.3.2)**

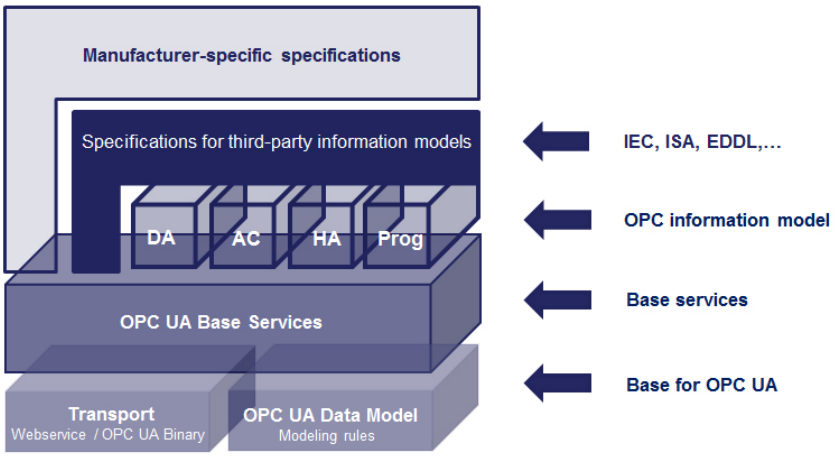

Fig.3.2

1. **DA:** definisce estensioni automation-data-specific, modelli per la descrizione di dati analogici e digitali.

2. **AC:** *(Alarm e Conditions)* rappresenta un modello avanzato per la gestione degli allarmi legati allo stato dei processi.

3. **HA:** *(Historical Access)* definisce i meccanismi per accedere alla cronologia dei dati e degli eventi.

4. **PROG:** *(Programs)* specifica i metodi utili ad avviare, monitorare e manipolare l'esecuzione dei programmi.

Un' applicazione OPC-UA è un sistema che ha la funzione di esporre dati e al contempo contiene sia le sue funzionalità specifiche che il loro mapping a OPC-UA attraverso OPC-UA Stack **(Fig.3.3)** (definisce meccanismi di comunicazione) e OPC-UA SDK(Software Development Kit). Analizzando l'OPC-UA Stack si ottiene che implementa diversi meccanismi di trasporto ed è diviso in tre strati come segue :

- **MESSAGE SERIALIZATION:** definisce i metodi per serializzare i dati scambiati in modalità binaria.
- **MESSAGE SECURITY:** specifica i criteri di protezione dei messaggi secondo le regole dettate dalla sicurezza dei Web Services.
- **MESSAGE TRANSPORT:** definisce il protocollo di rete utilizzato, che può essere UA TCP o HTTP e SOAP per i Web Service.

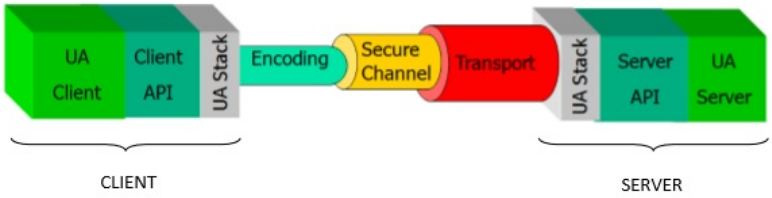

Fig.3.3

3.2 Information Modelling

Le specifiche di base di OPC-UA forniscono soltanto l'infrastruttura della model information, sta poi ai singoli vendor la modellazione dell'informazione che porta però a una forte frammentazione delle soluzioni con un conseguente accesso difficoltoso ai dati dai singoli client, per evitare questa problematica OPC-UA offre la possibilità di definire delle estensioni che fanno capo ad un information model di base definitio dalla OPC Foundation. Ogni produttore quindi può creare le proprie soluzioni applicabili ai propri dispositivi, quindi i client sono in grado di accedere a tali informazioni ugualmente com'è possibile per il produttore perché entrambi usano lo stesso model information di base. I principi base sanciti per OPC-UA sono:

- *Utilizzo di tecniche object-oriented, inclusi gerarchie di tipi e ereditarietà*
- *Modalità di accesso ai tipi identica a quella delle istanze*
- *Esporre le informazioni in vari modi sfruttando reti connesse a nodi*
- *Estendibilità delle gerarchie dei tipi e dei tipi di collegamento tra i nodi*
- *Nessuna limitazione sulla modellazione dell'informazione*
- *Information Modelling situato sempre lato server*

I concetti base dell'information modelling in OPC-UA sono i *Nodes*(nodi) e le *References*(collegamenti); questi due elementi vengono raggruppati nell'*Address Space* che definisce il set di informazioni esposte lato Server OPC-UA e accessibile da qualunque Client OPC-UA.

3.2.1 Nodes

I nodi appartengono a differenti *NodeClass* in base al loro obiettivo specifico ed inoltre contengono *Attributes* che vengono usati per descrivere i Nodes, in base alla NodeClass un nodo può avere un diverso set di attributi, fanno eccezione alcuni attributi comuni a tutti i nodi. L'accesso da parte di un Client al valore degli Attributes è sempre possibile tramite i servizi OPC-UA Reader, Write, Query e Subscription/MonitoredItem. I valori che ogni singolo attributo deve assumere, vengono assegnati quando il nodo viene istanziato nell' AddressSpace. Viene inoltre utilizzato un indicatore mandatory/opzional asscociato ad ogni singolo Attributo che indica se tale Attributo debba essere istanziato o meno.

3.2.2 References

Per definizione una Reference descrive la relazione tra due nodi è identificata univocamente dal nodo sorgente, dal nodo target, dalla Reference Type e dalla sua direzione. Il collegamento tra Reference e Node è univoco, si deduce quindi che una Refence può puntare a un solo Node e un Node può essere puntato da una sola Reference, per ottenere quindi questa associazione una Reference contiene:

- NodeId del nodo a cui punta
- OPC-UA Server del nodo puntato
- il Reference Type, che rappresenta la semantica della Reference
- la direzione della Reference in questione

Il nodo che contiene la Reference è detto **SourceNode** mentre il Node a cui punta è detto **TargetNode**, quest'ultimo può appartene allo stesso AddressSpace o ad un altro AddressSpace di un altro OPC-UA Server. Una Reference viene definita come istanza di un particolare NodeClasse definito come ReferenceTypeNode.

3.2.3 Base NodeClass

Come già dichiarato in OPC-UA esistono i NodeClasses che hanno la funzione di definizione dei Nodes e per l'istanza dei References, tutti i NodeClasses sono derivati da un Base NodeClass che può assumere vari tipi :

1. Variable
2. VaraibleType
3. Object
4. ObjectType
5. ReferenceType
6. DataType
7. Method
8. View

il Base NodeClass presenta un determinato numero di attributi e non prevedere References, tutti gli attributi della Base NodeClass vengono ereditati da tutti i NodeClasses e comuni a tutti i Nodi (**Fig.3.4**)

Attribute	DataType	Descrizione
NodeId	NodeId	Identifica univocamente un nodo in un server OPC-UA, per poterlo correttamente specificare nell'ambito dei servizi OPC-UA.
NodeClass	NodeClass	Enumerazione che identifica la NodeClass di appartenenza di un nodo, come ad esempio Object o Method.
BrowseName	QualifiedName	Identifica il nodo quando il server viene visitato col servizio Browse. Non è localizzato, ossia il suo valore non dipende dal valore di LocaleId, ossia dai settaggi relativi ad una determinata lingua di appartenenza.
DisplayName	LocalizedText	Contiene il nome del nodo che dovrebbe essere utilizzato nella user interface. Per questo motivo, è localizzato, cioè dipende dal valore di LocaleId, ossia dal setting linguistico di un determinato paese di appartenenza. Un Server potrebbe fornire diversi valori per il DisplayName, a seconda della lingua e del settaggio LocaleId.
Description	LocalizedText	Attributo opzionale che contiene una descrizione testuale localizzata del nodo.
WriteMask	UInt32	E' opzionale e specifica quali attributi del nodo sono modificabili.
UserWriteMask	UInt32	E' opzionale e specifica quali attributi del nodo sono modificabili dall'utente attualmente collegato al server.

Fig.3.4

3.2.4 ReferenceType NodeClass

Un Reference è definito come un istanza di un ReferenceType Node, quest'ultimo è visibile nell'AddressSpace e segue una organizzazione di tipo gerarchica basata ugualmente sulla definizione di ReferenceType Node. Come per i Nodes, il RefereceType Node è definito in termini di Attributes e References, come riportato nella tabella illustrativa che segue. (**Fig.3.5**)

Attribute	DataType	Descrizione
Base NodeClass Attributes		Sono tutti gli attributi ereditati dal Base NodeClass, descritti in precedenza.
IsAbstract	Boolean	Se TRUE, specifica che il ReferecenceType è utilizzato solo per scopi organizzativi per una migliore definizione della gerarchia dei ReferenceType.
Symmetric	Boolean	Indica se la Reference è simmetrica, cioè se è valida in entrambe le direzioni.
InverseName	LocalizedText	Attributo opzionale che specifica la semantica della Reference nella direzione inversa (per la direzione diretta si usano BrowseName e DisplayName). Può essere applicato solo a References asimmetriche e deve essere obbligatoriamente presente in caso di ReferenceType non astratto.

Fig.3.5

3.2.5 Object NodeClass

Gli Objects sono fondamentali in quanto utilizzati in OPC-UA per rappresentare: sistemi, componenti di sistemi, oggetti reali e oggetti software. Gli Object sono definiti usando l'Object NodeClass e contengono sempre Variables, Methods e altri Objects. Un Object può essere un EventNotifier a cui i client possono registrarsi per essere avvisati sugli Events che rappresentano specifiche situazioni che possono accadere improvvisamente in un sistema, quali ad esempio cambi di configurazione ed errori di sistema. Sono ricevuti dai Client tramite opportune notifiche chiamate Event Notifications. Gli Events non sono direttamente visibili nell'Address Space, ma sono accessibili solo tramite Object e View, si deduce quindi che Object e View sono gli unici Nodes utilizzabili per sottoscriversi ad un Evento la cui sottoscrizione avviene tramite il settaggio di un Attribute dell'Object o View, chiamato EventNotifier Attribute. L'Object NodeClass possiede sia Attributi sia References. Per quanto riguarda gli Attributi, esso eredita tutti i Base NodeClass Attributes e in più possiede l'attributo EventNotifier (**Fig.3.6**).

Attribute	DataType	Descrizione
Base NodeClass Attributes		
EventNotifier	Byte	Rappresenta una maschera di bit che specifica se un Object può essere usato per sottoscriversi ad eventi e se, in caso affermativo, la cronologia di questi eventi può essere accessibile o modificabile. Tra i Bit ve ne è uno da ricordare: SubscribeToEvents. Questo bit, se settato, indica che l'Object può essere usato per sottoscriversi ad eventi.

Fig.3.6

3.2.6 ObjectType NodeClass

Un'importante caratteristica di OPC-UA è quella di fornire informazioni sui tipi non solo a livello di tipo di dato ma anche a livello di oggetto. Ciò consente di recuperare ad esempio l'informazione su un tipo specifico di sensore che misura una temperatura. Nei precedenti standard OPC per fare questo era necessario utilizzare convenzioni specifiche del produttore; in OPC-UA si può rendere disponibile questa informazione definendo un tipo per i sensori di temperatura e creando oggetti di quel tipo. Gli ObjectTypes vengono utilizzati per fornire la descrizione completa di Objects inoltre gli ObjectTypes sono definiti utilizzando l'ObjectType NodeClass.

3.2.7 Variable NodeClass

Le Variables sono utilizzate per rappresentare valori semplici e complessi, appartengono a determinati Variable Types. Una Variable può essere usata, ad esempio, per rappresentare la temperatura misurata da un sensore. Le Variables possono essere definite come Properties o come DataVariables di Nodes contenuti nell'AddressSpace. Le Variable sono definite utilizzando il NodeClass Variable, quest'ultimo presenta Attributi come da tabella seguente. (**Fig.3.7**).

Attribute	DataType	Descrizione
Base NodeClass Attributes		Attributi ereditati dal Base NodeClass.
Value	*Il relativo tipo di dato è definito dall'attributo DataType specificato di seguito*	Rappresenta il valore più recente della Variable che il Server possiede per essa. Il suo tipo di dato è specificato dal dall'attributo DataType.
DataType	NodeId	I DataTypes sono rappresentati nell'Address Space come Nodes. Questo attributo contiene l'id del Node che rappresenta il tipo della Variable.
ValueRank	Int32	Specifica se il Value è un array e ne specifica il numero di dimensioni (non il numero di elementi per ogni dimensione).
ArrayDimensions	UInt32[]	Attributo opzionale che, in caso di array, specifica il numero di elementi per ciascuna dimensione.
AccessLevel	Byte	Maschera di bit che indica se l'attributo Value è leggibile/scrivibile e se il contenuto rappresenta il valore corrente o un dato storico.
UserAccessLevel	Byte	Identico a AccessLevel, ma riferito ai permessi dell'user. Indica se l'attributo Value è leggibile/scrivibile tenendo conto dei permessi dell'utente.
MinimumSamplingInterval	Duration	Attributo opzionale che indica quanto frequentemente il server può rilevare i cambiamenti dell'attributo Value. Per dati non direttamente accessibili dal server, ad es. la temperatura rilevata da un sensore, il server deve periodicamente effettuare un polling al dispositivo fisico e quindi questo sampling interval dovrà essere per forza di una certa consistenza. Un valore 0 di questo attributo significa che il Server monitora il cambiamento di valore continuamente.
Historizing	Boolean	Specifica se attualmente il server memorizza la cronologia dei cambiamenti della variabile.

Fig.3.7

Ogni Server può definire i propri DataType, espressione dei nodi presenti nell'Address-Space, accessibili dai Client. Lo standard OPC-UA prevede quattro tipi di DataType:

1. **Built in**: definiti in OPC-UA e con capacità estendibili, comprendono tipi di base come int32, Boolean, Double e anche tipi propri dello standard (NodeId, LocalizedText, QualifiedName).

2. **Simple**: sono sottotipi dei DataType built in, ogni Information Model può definire i propri.

3. **Enumeration**: è un insieme di valori nominali gestiti in maniera eguale ai Data-Type built in.

4. **Structured**: rappresentano dati strutturati e sono il tipo di dato ridefinibile più potente, come per i Simple, ogni Information Model può definire i propri.

OPC-UA definisce due tipi di Variable: *DataVariables* e *Properties*, entrambe utilizzano il NodeClass Variable presentando però alcune limitazioni, di seguito descritte.

- **DataVariables:** sono utilizzati per rappresentare i dati di un Object, esse sono strutturate in maniera gerarchica, cioè possono avere sottovariabili contenenti parte dei dati e proprietà che le descrivono.
- **Properties:** utilizzate per descrivere le caratteristiche di un nodo, non descrivibili tramite le Attributes del nodo.

3.2.8 VariableType NodeClass

Come definito per gli ObjectType, i VaribleType descrivono la semantica di una Variable oppure permettono di restringere il dominio dell'attribuo Value. I VariableType vengono definiti utilizzando il VariableType NodeClass.

3.2.9 Views

Un Server OPC-UA può offrire un ampio insieme di Nodes a cui un Client è interessato ad un suo sottoinsieme definibile per diversi Client nell'AddressSpace. Ogni Client accede ad una *Views* che restringe l'insieme dei Nodes ad un sottosistema che soddisfa i requisiti del Client. Ogni Node appartente ad una Views può contenere solo un sottoinseme delle sue References, definite in fase di impostazione della View. Una View è rappresentata nell'AddressSpace come un nodo che fornisce un punto d'ingresso ai dati che si vogliono mostrare. Tutti i nodi che fanno parte di una Views sono accessibili gerarchicamente e hanno come punto d'ingresso il Nodo di partenza stesso. Come descritto per gli Object, anche per la Views è possibile settare il bit SubscribeToEvent dell'attributo EventNotifer, così da rendere possibile la sottoscrizione a un Events.

3.2.10 Methods NodeClass

I *Method* rappresentano funzioni richiamabili che vengono invocate tramite l'ausilio del servizio Call del protocollo OPC-UA. Un Method è esposto all'esterno soltanto tramite la sua signature senza imposizioni da parte del protocollo su come questo deve essere implementato. Gli argomenti di input e output sono specificati da particolari Varibles dette *Properties* e i Methods sono definiti usando I Method NodeClass

3.3 Servizi

I servizi che offre OPC-UA richiedono un pattern di comunicazione che deriva direttamente da quelli utilizzati per i sistemi Web Service, in cui il servizio si articola secondo una comunicazione di tipo richiesta (*Request*) e riposta (*Response*). Per invocare un Service un client invia un messaggio di *Request* al server che dopo aver elaborato la richiesta invia un messaggio di *Response* al client. Questo tipo di interazione non è istituita in real-time ma richiede una tempistica utile alla sua attuazione che quindi porta alla definizione della comunicazione stessa di tipo asincrona; quindi dopo aver mandato il messaggio di Request l'applicazione client può continuare la propria esecuzione in attesa del messaggio di Response.(**Fig.3.8**).

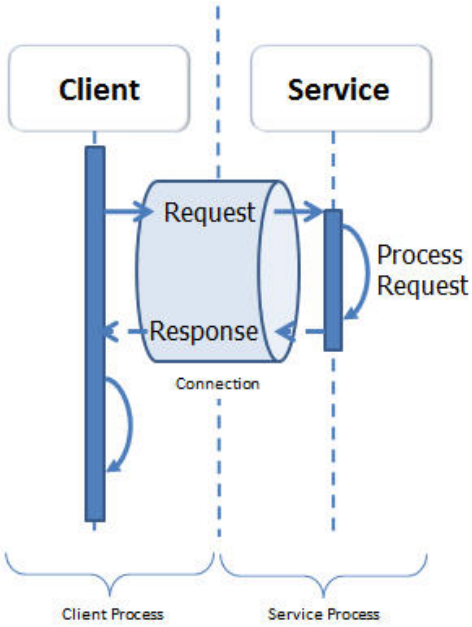

Fig.3.8

3.3.1 Timeout

In OPC-UA la comunicazione è concepita per poter funzionare tra diversi sistemi, eterogenei tra loro, connessi in una rete; quindi per come è strutturata la comunicazione sono possibili interruzioni di servizio in qualsiasi momento che devono essere rilevate e gestite, per questa motivazione ogni chiamata è strutturata e contiene come parametri dei **Timeout** definiti dal client che servono proprio a rilevare l'assenza della comunicazione così che in seguito questa possa essere gestita correttamente.

3.3.2 Request Response Headers

In OPC-UA i messaggi sia di Request (**Fig.3.9**) che di Response (**Fig.3.10**) relativi a un qualsiasi servizio contengono il medesimo Header.

Proprietà pubbliche	Descrizione
NodeId AuthenticationToken	Identificatore utilizzato per assegnare la chiamata del servizio alla sessione precedentemente stabilita tra client e server.
DateTime Timestamp	Istante di tempo in cui il client ha inviato la richiesta.
uint RequestHandle	Identificativo assegnato alla chiamata del servizio, definito dal client.
uint ReturnDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice *status code*.
String AuditEntryId	Identifica il client o l'user che ha iniziato l'azione; usato in caso di auditing.
uint TimeoutHint	Timeout settato per la chiamata del servizio lato client nell' UA Stack; quando termina, il server annulla la chiamata ricevuta.

Fig.3.9

Proprietà pubbliche	Descrizione
DateTime Timestamp	Istante di tempo in cui il server ha inviato la risposta.
uint RequestHandle	Identificativo assegnato alla chiamata di servizio definito dal client.
StatusCode ServiceResult	Classe definita dallo standard per descrivere il risultato della chiamata del servizio. Contiene un uint a 32 bit in cui i 16 più significativi rappresentano il valore numerico dell'errore ed i
	rimanenti contengono informazioni aggiuntive. In particolare, i 2 bit più significativi si riferiscono all'attendibilità del risultato (00=Good, 01=Uncertain, 10=Bad, 11=not used).
DiagnosticInfo ServiceDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice status code.

Fig.3.10

3.3.3 Discovery Services Set

Il Discovery Services Set è l'insieme dei servizi utilizzati dai vari client per individuare gli endpoints esposti da un Server OPC-UA e per leggere le specifiche di sicurezza implementati in essi; queste informazioni contenute negli endpoints saranno poi utilizzate dal Client per creare un *SecureChannel* con il Server. Ogni Server OPC-UA deve esporre necessariamente almeno un Discovery Endpoint, attraverso cui il client accederà per leggere le configurazioni di sicurezza contentute in essi, per determinare queste informazioni si usa il servizio GetEndpoints. L'accesso da parte del client agli EndPoint avviene senza che sia necessario stabilire una sessione inoltre la lettura delle relative informazioni di sicurezza contenute negli Endpoint avviene in chiaro senza che ci sia un meccanismo

di sicurezza. Di seguito è ripartata la sequenza di funzionamento per l'accesso ad un Endpoint (**Fig.3.11**)

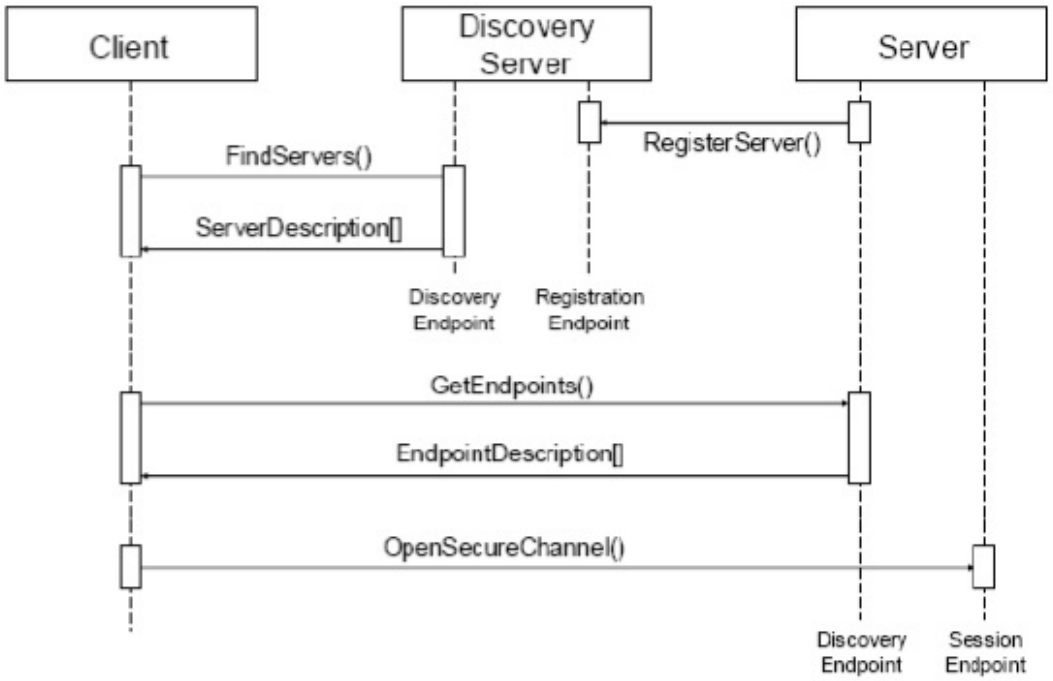

Fig.3.11

3.3.4 FindServers

Il servizio è implementato sia dai Discovery Server, sia dai generici server OPC UA; nel primo caso viene restituita la lista dei server registrati mentre nel secondo caso il server restituisce se stesso, questa caratteristica assicura il corretto funzionamento anche nel caso di un unico nodo server in una rete dove non é presente alcun Discovery Server. La lista dei server ritornati dalla funzione FindServers contiene, per ogni server, l'elenco degli URL che il client può utilizzare per invocare il servizio GetEndpoint. FindServer non richiede meccanismi di sicurezza ne una sessione attiva, perché la creazione di quest'ultima è subordinata alla conoscenza delle informazioni restituite da questo servizio.

3.3.5 GetEndpoints

Questo servizio ha la funzione di restituire gli endpoint disponibili nel server. In fase di invocazione va specificato l'URL del Discovery Endpoint del Server, che a sua volta deve supportare il servizio GetEndpoints. Le informazioni che vengono in seguito restituite sono necessarie per stabilire una SecureChannel che client e server. Non essendo previsti

meccanismi di controllo e sicurezza questo servizio è vulnerabile ad attacchi di tipo DOS(Denial Of Service) cioè attacchi finalizzati alla negazione del servizio. Il protocollo in analisi, OPC-UA, segue una Security Policy(contiene tutti gli algoritmi sviluppati nel protocollo per per funzioni di sicurezza) che viene identificata dall'URI. Prendendo in analisi l'URI associato al sito ufficiale di questo protocollo.

- https://opcfoundation.org/UA/SecurityPolicy/Basic128Rsa15

si evince che il suddetto è associato a una policy che prevede l'algoritmo AES con chiave pari a 128 bit per quanto riguarda la cifratura mentre viene utilizzato l'algoritmo RSA 1.5 per quanto riguarda tutte le attività che usano crittografie asimetriche. Quando un client chiede l'elenco degli endpoints disponibili in un server, il Client può ridurre l'elenco dei disponibili nel server specificando dei criteri di filtraggio basati su

1. *Locale Ids*: specifica il settaggio legato alla lingua che il client desidera ottenere.
2. *Trasport Profile URIs*: indica le caratteristiche dei protocolli di trasporto che il client desidera ottenere.

Nel caso il server non sia in possesso di endpoints come specificati nel filtraggio, questo ritorna un elenco vuoto.

3.3.6 RegisterServer

Questo servizio permette la registrazione di un server presso un Discovery Server esso può essere usato da un server o da una applicazione di configurazione che non abbia funzioni di client perché sia possibile una connessione un server deve prima stabilire un SecureChannel con il Discovery Server, questa richiesta permette di avere la certezza che solo server trusted possano essere registrati nei Discovery Servers. A tal fine, l'amministratore del Server dovrà fornire al Server un Endpoint Description per il server Discovery come parte del processo di configurazione. Il processo di registrazione da parte del Server consiste nella trasmissione al Discovery Server il proprio ServerUri (global unique identifier dell'instanza del Server) e i discoveryUrls (gli URLs dei Discovery Endpoint), che verranno successivamente chiesti dai client tramite il servizio GetEndpoints. Un'altra informazione che il server deve fornire in fase di registrazione è il nome di se stesso (servernames[], vettore di stringhe)in tutte le lingue supportate. La registrazione può essere realizzata in due diverse modalità in base alla tipologia di applicazioni Server:

1. quelle che vengono lanciate manualmente (ad esempio da un amministratore di sistema).
2. quelle che vengono avviate automaticamente quando un client tenta di connettersi.

3.4 Sicurezza

Le applicazioni di tipo OPC-UA possono essere eseguite in molti ambienti differenti, proprio per la definizione del protocollo, cioè fornire la possibilità di creare dispositivi eterogenei che struttano OPC-UA come protocollo comunicativo, ne consegue che OPC-UA può essere implementato per differenti livelli della *automation pyramid* (**Fig.3.12**)

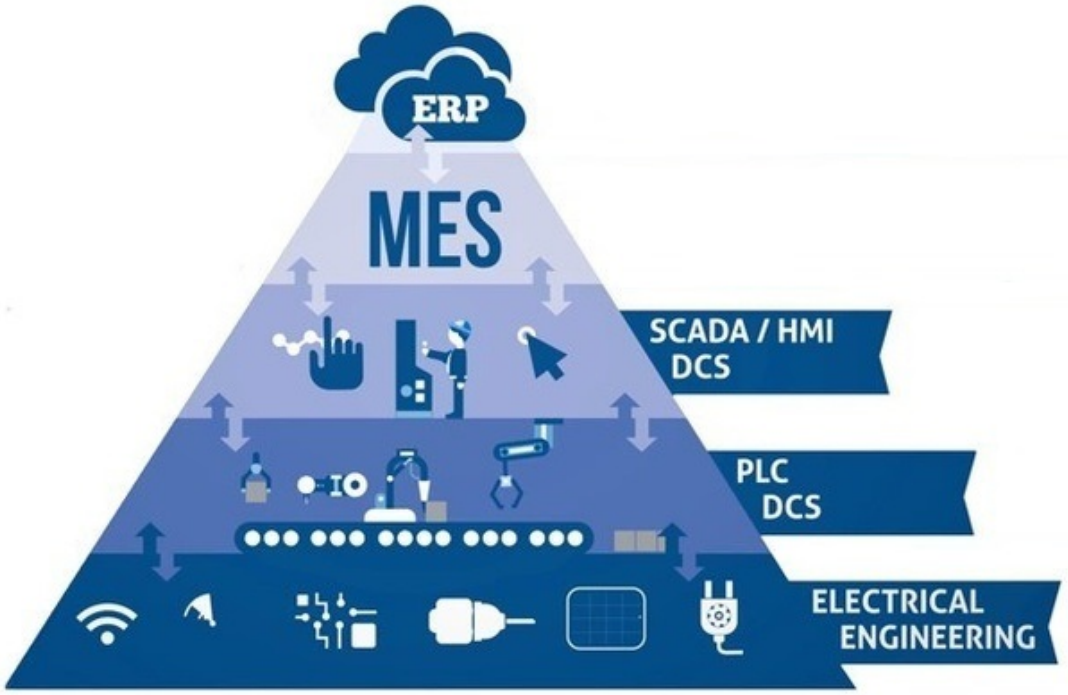

Fig.3.12

essendo una struttura a piramide, il protocollo deve fornire un criterio si sicurezza che sia il più flessibile possibile così da essere adattabile ai vari livelli della piramide in cui può variare la percezione delle performance piuttosto che della sicurezza.

3.4.1 Architettura

L'architettura di un sistema OPC-UA presenta una divisione a livelli :

- Application Layer
- Communication Layer
- Transport Layer

L'Application Layer viene usato per la trasmissione di informazioni tra client e server che hanno stabilito una sessione (OPC-UA Session) questa viene stabilita su una Secure

Channel (situata nel Communication Layer) che ha lo scopo di rendere sicuro lo scambio dei dati di una sessione. Il Transport Layer è il livello responsabile della trasmissione e ricezione dei dati, tramite l'utilizzo di una socket a cui vengono applicati meccanismi di gestione dell'errore atti a prevenire protezione del sistema evitando negazione del servizio(Denial of Service).

3.4.2 Secure Channel

La generazione di una Secure Channel in OPC-UA avviene tra due entità che sono OPC-UA Server e OPC-UA Client, perché vi sia una connessione le parti devono rispettare una serie di passi fondamentali (**Fig.3.13**):

1. Si compone nell'atto del settaggio delle opzioni di configurazione della connessione, questa prima fase può essere tralasciata se l'applicazione risulta preconfigurata, nel caso si cada nella prima occorrenza allora il client deve inviare una GetEndpoints request al Discovery Endpoint del server a cui desidera connettersi per ottenere la descrizione degli Endpoints esistenti, inclusa la configurazione di sicurezza. A seguito della ricezione delle informazioni, il client seleziona un Endpoint con una configurazione di sicurezza supportata e valida l'Application Instance Certificate del server, questa interazione avviene verificandone la validità presso la Validation Authority associata.

2. Ha come pre-condizione l'attendibilità del certificato, se questa condizione è confermata, allora viene inviata una richiesta di OpenSecureChannel in accordo alla Security Police e al Security Mode del Session Endpoint selezionato. Il Security Mode può prendere forma in tre stati differenti sul piano della sicurezza nella comunicazione:

   (a) None: i messaggi non hanno politiche di sicurezza.

   (b) Sign: i messaggi saranno firmati con la chiave associata al certificato del client OPC-UA.

   (c) SignAndEncrypt: i messaggi oltre ad essere firmati, come nella casistica Sign, sono cifrati con la chiave pubblica presente nel certificato del server.

   Mentere la Security Police definisce quale algoritmo utilizzare per la firma e cifratura dei messaggi.

3. Il server, a seguito della ricezione di un messaggio, valida il certificato del client con un'apposita richiesta alla relativa Validation Autority. Assodata l'attendibilità del messaggio questo viene decifrato tramite l'utilizzo della chiave privata del server e verificata la firma tramite l'uso della chiave pubblica nel certificato del client. Il server invia quindi una response che utilizza egualmente i meccanismi di sicurezza utilizzati dal client.

Il Secure Channel possiede un certo lifetime finito, al termine del quale sarà necessario ripercorrere le prime due fasi sopra descritte per ottenere un nuovo canale di comunicazione sicuro.

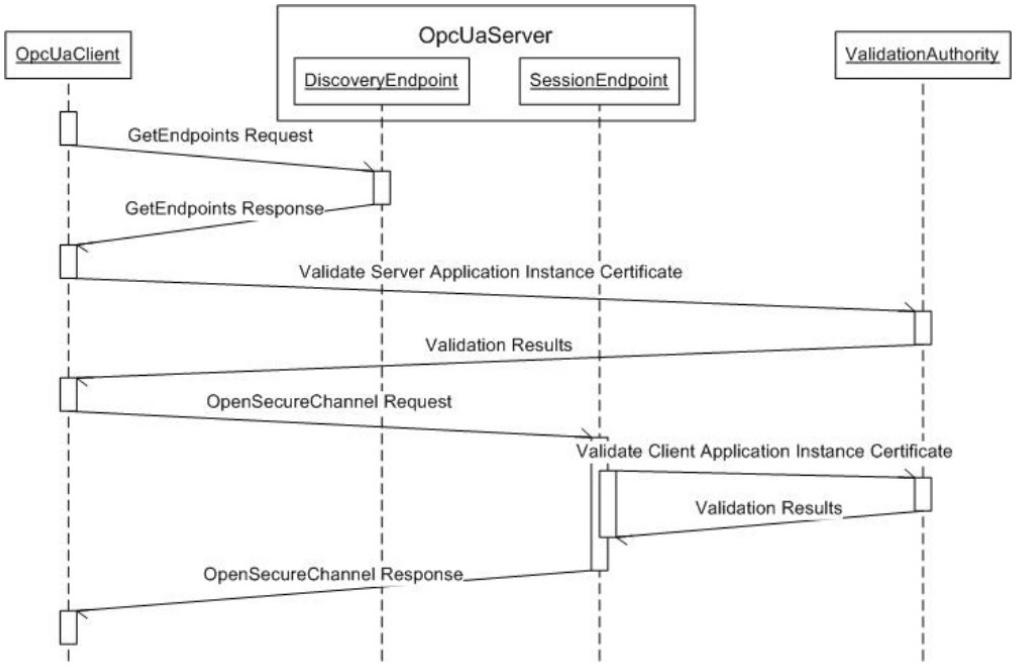

Fig.3.13

3.4.3 Session

Una Session si instaura al seguito della creazione di una Secure Channel, viene inviata una richiesta CreateSession al server che risponde fornendo i propri Software Certificates per comunicare le proprie funzionalità e per dimostrare il possesso del certificato utilizzato nella creazione del sottostante Secure Channel. La creazione di una Session deve essere necessariamente seguita dalla sua attivazione, questa funzionalità si reifica inviado al server una richiesta di tipo ActivateSession che include internamente le credenziali dell'user corrente e i Software Certificates del client. Le credenziali possono essere rappresentate da un certificato X.509(standard ITU-T fornisce standardizzazione per i certificati a chiave pubblica, certificati di attributo ed un certification path validation algorithm) oppure dalla coppia username-password. A seguito della validazione, da parte del server, delle credenziali e dei certificati software, la Session è instaurata e utilizzabile da parte del client che può accedere ai dati del server.

3.4.4 SecureChannel Service Set

Il SecureChannel Service Set rappresenta l'insieme dei servizi forniti in OPC-UA utili alla gestione di una SecureChannel, essenzialmente esistono due servizi:

1. OpenSecureChannel
2. CloseSecureChannel

Capitolo 4

MQTT

Il secondo protocollo che andremo ad illustrare è MQTT(Message Queue Telemetry Transport), protocollo di standard ISO sviluppato da IBM nel 1999 è un protocollo light-weight pensato per le comunicazioni su reti poco affidabili, con bassa ampiezza di banda e scarsa affidabilità in termini di consegna dei pacchetti, MQTT si basa sul concetto di publish-subscribe, quindi a seguito di topic specifici definibili on-the-fly, i vari client registrati, possono comunicare tra loro pubblicando informazioni sui vari topic e dualmente ricevere aggiornamenti. Tutta la comunicazione viene sostenuta da un broker, il quale ricopre la funzione di mediatore, filtrando e distribuendo i messaggi ai vari sottoscrittori sulla base del topic di appartenenza del messaggio stesso. MQTT, in virtù delle specifiche sopra dette è stato definito dall' OASIS come lo standard di riferimento per la comunicazione per l'IoT(Internet of things) e M2M(machine-to-machine).

4.1 Topic

Un concetto importante nel protocollo MQTT è quello di "topic". Un broker MQTT può essere pensato come amministratore di un numero imprecisato di code, ciascuna contraddistinta da un nome chiamato appunto topic (argomento). Questo permette ai client sottoscrittori che si connetteranno al broker di essere informati solo di alcuni messaggi, appartenenti a quei topic che hanno selezionato al momento della sottoscrizione. Un client pubblicatore dovrà associare un topic ad ogni messaggio, così il broker capirà se accodare il messaggio in qualche coda, se la coda per quel topic già esiste, o creare una nuova coda per un topic non usato in precedenza. Le code di un broker MQTT non sono semplici code, ma code gerarchiche. Un topic è identificato da una chiave gerarchica questo permette ad un sottoscrittore di configurarsi come ascoltatore di una categoria di messaggi, cioè messaggi provenienti da più di una coda, contraddistinta dagli identificatori più a destra. Questo modello gerarchico conferisce ad MQTT una estrema flessibilità e generalità di utilizzo.

4.2 Standard Structure

MQTT (**Fig.4.1**) è un protollo basato sulla comunicazione a modello publish-subscribe con il fine di essere aperto, semplice e di facile implementazione, queste caratteristiche lo rendono molto funzionale in situazioni particolari, come ad esempio:

1. Quando la rete che ha necessità di poca larghezza di banda
2. Quando il sistema che lo implementa ha poca potenza di calcolo (sistema embedded)

Questo protocollo basa l'implementazione su tcp e come HTTP, che lo è per il Word Wide Web, aspira ad essere il nuovo standard de facto per il mondo dell'IoT.

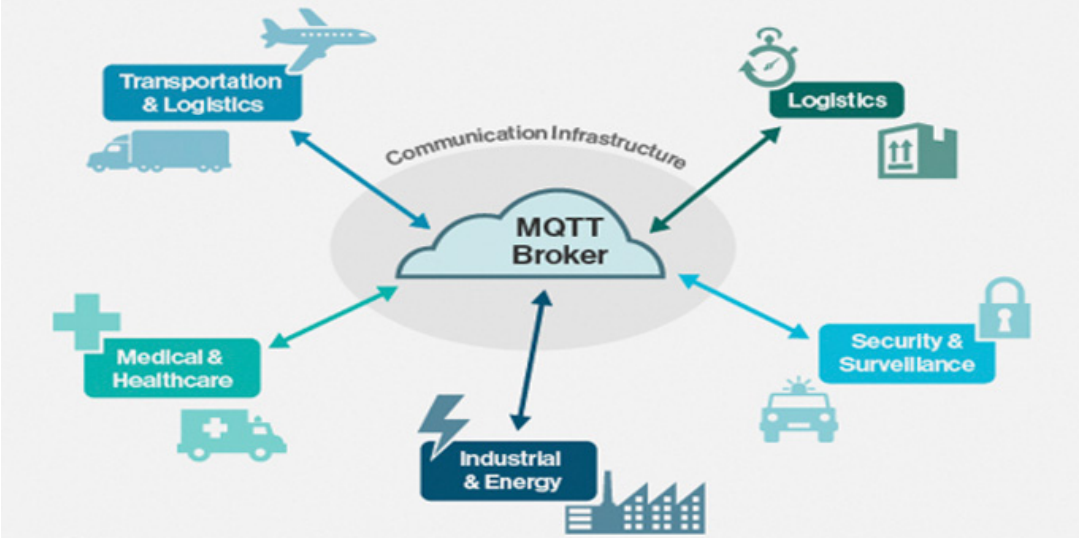

Fig.4.1

4.3 Information Modelling

Il protocollo MQTT quindi prevede che il suo funzionamento sia basato su comunicazione publish-subscribe con l'utilizzo del protocollo TCP/IP, ogni messaggio previsto con un formato supportato da MQTT ha bisogno di un header fissato e di un payload, in particolare l'header è strutturato in due byte:

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

4.3.1 Tipi Di Messaggio

Le informazioni che riguardano i tipi di messaggi sono presenti nel primo byte e occupano le posizioni comprese dal settimo al quarto bit, quindi il nome viene definito tramite l'utilizzo di 4 bit che permette quindi di descrivere un numero compreso nell'intervallo tra 0 e 15, in particolare rappresentano rispettivamente:

- 0: il tipo riservato (reserved)
- 1: il client vuole connettersi al server (connect)
- 2: indica un acknolegement (connack)
- 3: indica operazioni di pubblicazione di un messaggio (publish)
- 4: indica l'acknolegement di una publish (puback)
- 5: indica la publish ricevuta (pubrec)
- 6: indica una publish rilasciata (pubrel)
- 7: indica una publish completata (pubcomp)
- 8: indica una richiesta di un client che si vuole iscrivere (subscribe)
- 9: indica un acknolegement di un iscrizione (suback)
- 10: indica la volontà di un client di disiscriversi (unsubscribe)
- 11: indica l'acknolegement di una disiscrizione (unsuback)
- 12: indica una richiesta di ping (pingreq)
- 13: indica una risposta ad un ping (pingresponse)
- 14: indica una disconnessione (disconnect)
- 15: indica un bit riservato

4.3.2 Flags

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 la flag di Dup viene settata quando il client o il server cercano di riinviare un publish, pubrel, subscribe o un unsubcribe, questo flag è settabile solo per messaggi che hanno un QoS maggiore di 0 e prevede l'invio di un acknoledgment. La seconda flag, relativa alla qualità del servizio (QoS), indica il livello di affidabilità del servizio, può assumere valori come segue:

- **QoS 0:** indica la caratteristica di al più 1 (**Fig.4.2**)

Fig.4.2

- **QoS 1:** indica la caratteristica di almeno 1 (**Fig.4.3**)

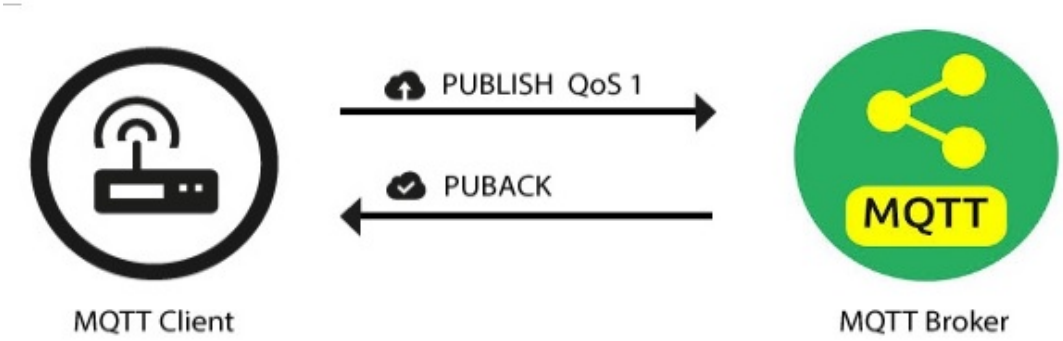

Fig.4.3

- **QoS 2:** indica la caratteristica di esattamente 1 (**Fig.4.4**)

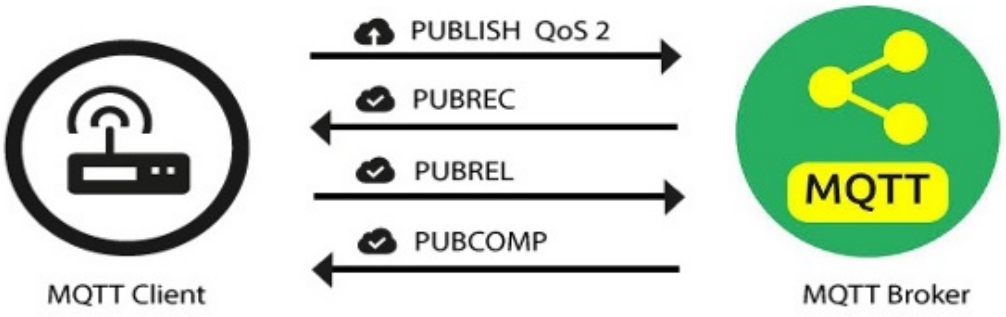

Fig.4.4

- **QoS 3:** indica la caratteristica di riservato

La terza flag, Retain, è usata esclusivamente per i messaggi di publish, 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 l'invio ai client iscritti. Nel momento in cui un nuovo client si iscrive allo stesso topic, gli viene inviato quel messaggio a con il flag di retain settato. Il secondo byte rappresenta il numero di byte rimanenti del messaggio corrente, includendo il payload. La codifica dei messaggi avviene utilizzando un singolo byte per messaggi fino a 127 byte di cui i primi 7 bit di ogni byte tengono il dato e l'ottavo altri byte presenti nella rappresentazione. Il protocollo limita i byte di rappresentazione ad un massimo di 4.

4.3.3 Header Variabili

Aluni comandi MQTT prevedono 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 costruzione dell'header variabile è 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:** è rappresentato del bit 1 del byte della CONNECT, se settato a 0, il server deve memorizzare la sottoscrizione del client anche se il client è offline, se invece settato a 1 non dovrà mantenere queste informazioni a seguito della disconnessione del client.
- **Will Flag:** è rappresentato dal bit 2 del byte della CONNECT. Se questo flag è settato il server pubblica un messaggio a nome del client quando 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 numero 5. Questo bit indica se il server deve mantenere i messaggi inviati con questa flag.
- **Username Password Flag:** sono alla posizione 6 e 7 del byte della CONNECT. Un client può specificare se il messaggio che sta inviando è formale de 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, 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, ugualmente se il client non riceve risposta dal server nello specifico tempo, chiude il canale TCP/IP.

- **Connect Return Code:** rappresenta il valore di ritorno della CONNECT, specificando in particolare, connessione accettata se il flag è posto a 0, connessione rifiutata a causa di versioni non compatibili se posto a 1, connessione rifiutata causa identificatore rifiutato se il codice di ritorno è posto a 2, sever 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:** è presente nell'header variabile del messaggio MQTT PUBLISH, rappresenta il canale dove il messaggio verrà pubblicato. I client utilizzano questa chiave per sottoscriversi ai canali e ricevere messaggi.

4.3.4 Payload

MQTT prevede alcuni messaggi che posseggono un payload, indicati come segue:

- **Connect:** il payload contiene una o più stringhe utf-8, queste specificano un univoco identificatore per il client, un Will Topic, messaggio, username e password se settati.

- **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.

4.4 Messaggio Di Comando

Analizziamo ora i tipi di messaggi elencati precedentemente, il formato dei vari messaggi è come riportato in figura (**Fig.4.5**).

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							

Fig.4.5

4.4.1 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 si farà riferimento al protocollo MQTT. Nella Connect le flag di Dup, QoS e Retain non sono utilizzate, mentre sono 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, che appaiono come segue:

- **Client Identifer:** stringa di lunghezza compresa tra 1 e 23 caratteri, identifica univocamente il client. 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 che rappresenta identificatore rifiutato.
- **Will Topic:** stringa che mostra il topic dove dovrà essere inviato il will message.Il livello di QoS è definito dal will QoS e il retain dal will retain.
- **Will Message:** 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.
- **Password:** se il bit è settato e l'username è impostato questa stringa indica la password relativa all'utente del client.

Se il server riceve una CONNECT valida risponde inviando al client una CONNACK, a seguito della non ricezione della CONNACK il client chiude il canale di comunicazione TCP/IP.

4.4.2 Connack

Questo messaggio viene inviato dal server a seguito della ricezione di una CONNECT da parte di un client, ugualmente come nella CONNECT i flags di Dup, QoS e Retain non vengono utilizzati, come non è presente il payload e il campo di lunghezza variabile non viene completamente utilizzato.

4.4.3 Publish

Questo messaggio è inviato dal client al server, a seguito della necessità del client di consegnare questo messaggio a tutti gli iscritti a quel particolare topic, ne consegue quindi che ogni messaggio associato a un topic e questo messaggio è conseguentemente inoltrato a tutti i suoi iscritti. In questa casistica è possibile settare Dup, QoS e Retain. L'header variabile contiene:

- **Topic Name:** una stringa 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 o superiore a 1.

La risposta attesa a seguito della ricezione di una Publish è in funzione del QoS settato, illustrata in seguito la casistica:

- **QoS 0:** se settato a 0 non viene inviato messaggio di ritorno.
- **QoS 1:** se settato a 1 viene inviato un messaggio di PUBACK a chi ha inviato il messaggio.
- **QoS 2:** se settato a 2 viene inviato un messaggio di PUBREC al client.

4.4.4 Puback

Questo tipo di messaggio ha occorrenza se la richiesta effettuata dal client ha QoS pari a 1. In questa casistica le flag Dup, QoS e Ratain non vengono utilizzate, non è presente payload e l'header variabile può contenere l'id del messaggio. La ricezione di questo tipo di messaggio, permette di definire, da parte del client, il corretto invio del messaggio e conseguentemente la corretta ricezione dello stesso da parte del server.

4.4.5 Pubrec

Questo tipo di messaggio rappresenta la risposta che un server invia al client a seguito dell'invio, da parte del client stesso, di un messaggio Publish con QoS settato a 2. Come definito per Puback, anche in questo caso, non sono presenti i flag di Dup, QoS, Ratain, non è prestito un payload. Questo tipo di messaggio ha la sola funzione di determinare che il messaggio precedente è stato inviato correttamente.

4.4.6 Pubrel

Questo messaggio può avere occorrenza in due casi:

1. Messaggio in risposta da un publisher ad un pubrec da un server
2. Messaggio in risposta dal server a un messaggio di pubrec da un iscritto

In questo caso possono essere settati i flag di Dup, QoS ma non di Retain, non è previsto payload mentre l'header variabile contiene l'id del messaggio a cui verrà fatto l'acknoledgement. Il funzionamento prevede che, quando un server riceve un messaggio di questo tipo da un publisher esso rende disponibile il messaggio originale per tutti gli iscritti interessati, mentre quando un iscritto riceve un messaggio di Pubrel da un server, esso lo notifica all'applicazione iscritta e manda un messaggio di PUBCOMP al server.

4.4.7 Pubcomp

Anche in questo caso, come visto per Pubrel, le casistiche in cui può comparire questo messaggio sono:

1. Una risposta da parte del server che ha ricevuto una Pubrel.

2. La risposta da parte di un iscritto che ha ricevuto dal server una Pubrel.

In questo caso i flag Dup, QoS e Retain non vengono utilizzate, non è previsto il payload e nell'header variabile è presente lo stesso message id che è notificato dalla Pubrel. Il funzionamento definisce che quando un client riceve un messaggio di Pubcomp esso elimina il messaggio originale, considerandolo come inviato correttamente.

4.4.8 Subscribe

Questo messaggio permette al client di registrarsi ad uno o più client contemporaneamente, i messaggi pubblicati in questi topic saranno inviati dal server, tramite un messaggio Publish, a tutti i client registrati su questo particolare topic. Con il messaggio di subscribe, oltre a registrasi sul singolo topic, viene anche definito il livello QoS con il quale un client vuole ricevere i messaggi del relativo topic. Analizzando questo messaggio, possiamo verificare che i flag settati sono quindi Dup, QoS ma non Retain, l'header variabile contiene un message id in virtù del messaggio di subscribe che prevede un QoS pari a 1, infine il payload contiene una lista di topic a cui il client vuole iscriversi e il relativo livello di QoS per ogni topic.

4.4.9 Suback

Questo tipo di messaggio rappresenta la risposta e accettazione, da parte del server, alla richiesta effettuata dal client di sottoscriversi ad un particolare topic. Un pacchetto SUBACK contiene un elenco di codici di ritorno che specificano il livello QoS massimo che è stato concesso in ciascuna Sottoscrizione richiesta dal SUBSCRIBE. In questo messaggio non sono previste nessuna delle tre flags Dup, QoS e Retain. L'header variabile può contenere l'id del messaggio che è stato notificato, il payload invece presenta una lista di QoS relativi ai singoli topic.

4.4.10 Unsubscribe

Un messaggio di UNSUBSCRIBE è inviato da un client ad un server per richiedere l'eliminazione dalla lista di iscritti di quel particolare topic presente nel server. Con questo messaggio vengono settate i flag Dup, QoS ma non la flag di Retain. Questa tipologia di messaggio ha un QoS pari a 1 e quindi l'header variabile contiene l'id del

messaggio. Il server, a seguito della ricezione di un UNSUBSCRIBE invia al client un messaggio di tipo UNSUBAK.

4.4.11 Unsuback

Il comando UNSUBACK è inviato dal server al client per dare conferma della ricezione di un messaggio di UNSUBSCRIBE. In questo messaggio nessuno dei tre flags è utilizzato e l'header variabile contiene come nel caso precedente l'id del messaggio, inoltre non è previsto nessun payload.

4.4.12 Pingreq

Questo messaggio è inviato da un client a un server, può essere utilizzato per tre occorrenze:

1. Indicare al server che il client è vivo.
2. Richiedere una risposta al server per confermare che è vivo.
3. Verificare che la connessione di rete sia attiva.

Questo pacchetto è utilizzato nell'elaborazione del keep alive, inoltre questo messaggio non prevede header variabile ne un payload

4.4.13 Pingresp

Messaggio di risposta in seguito a un PINGREQ e sta ad indicare che il server è vivo e quindi funzionante. Come nel caso precedente non vengono usate flag dell'header fisso e variabile, inoltre non è presente neanche il payload.

4.4.14 Disconnect

Il pacchetto DISCONNECT rappresenta il pacchetto del controllo finale inviato dal client al server che indica la chiusura del canale di connessione TCP/IP tra i due. Come per il pacchetto sopra definito, non prevede nessun flag, header varibile e payload.

Capitolo 5

LonWorks

Il terzo standard, che questa tesi si prefigge di analizzare, risponde al nome di LonWorks tecnologia proprietaria sviluppata dalla Echelon Corporation. La tecnologia LonWorks rappresenta una tecnologia di comunicazione digitale su bus con lo scopo di garantire prestazioni, affidabilità, flessibilità e una facile installazione o manutenzione di sistemi di automazione ad intelligenza distribuita. La tecnologia LonWorks fù originariamente sviluppata per dispositivi di rete o nodi che potessero comunicare utilizzando diversi tipi di connessioni fisiche come doppino, onde convogliate, fibra ottica,trasmissioni radio e TCP/IP. Il protocollo storicamente denominato LonTalk, implementa tutti gli strati della pila ISO/OSI ed oggi è riconosciuto come standard internazionale definibile con il nome ISO/IEC 14908. Inizialmente LonWorks venne impiegato per la creazione di automazione negli edifici, in particolare in sistemi di riscaldamento, condizionamento e illuminazione, a seguire però ha avuto anche affermazione in ambito industriale e di domotica (IoT). In virtù delle sue caratteristiche LonWorks, negli ultimi anni, sta avendo sempre un maggior impiego in sistemi di gestione energetica, con lo scopo di ridurre i consumi energetici e allo stesso tempo abbassare i costi e la frequenza delle manutenzioni.

5.1 Standard Structure

L'obiettivo che insegue questa piattaforma è la fornitura di un sistema integrato, ottimamente progettato e in fine economico che permetta la creazione di sistemi intelligenti. Al fine di raggiungere gli obiettivi prefissi LonWorks è sviluppato attorno al concetto di **Neuron** per l'assonanza con un sistema centrale "neurone", il nucleo Neuron è disponibile anche singolarmente e risponde al nome di **NeuronChip**. Allo scopo di ridurre ulteriormente i costi del dispositivo Echelon fornisce anche nuclei Neuron combinati in grado ci comunicare tra loro tramite ricetrasmettitori, chiamati **Smart Transceivers**. Avendo espresso la volontà di simplicità, il nucleo Neuron ricopre gli strati 2 e 6 dello stack ISO/OSI (**Fig.5.1**) inoltre Smart Transceiver va a ricoprire lo strato 1, ne con-

segue che quindi il produttore deve fornire solo la programmazione inerente allo strato applicativo e la relativa integrazione con la rete.

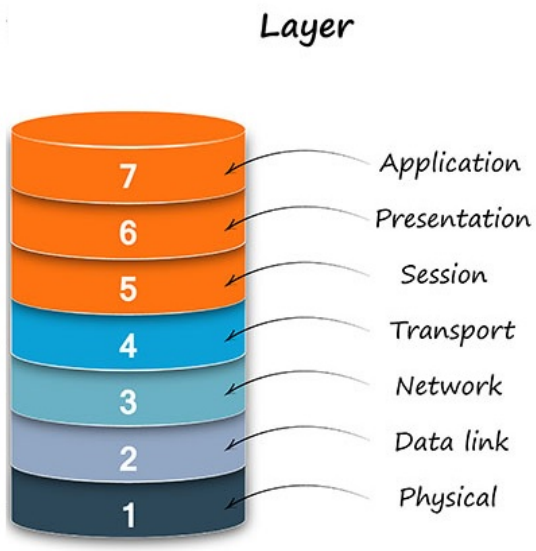

Fig.5.1

5.1.1 Neuron

Il nucleo Neuron è un componente a semiconduttore specificamente progettato per fornire intelligenza e funzionalità di rete ai dispositivi di controllo a basso costo. Il nucleo Neuron comprende fino a quattro processori che forniscono entrambe le funzionalità di elaborazione di comunicazione e delle applicazioni. Due processori eseguono lo strato 2 a 6 attuazione del 14.908-1 protocollo ISO / IEC e il terzo esegue strato 7 e il codice dell'applicazione. A seguito dell'aggiornamento a LonWorks 2.0 i nuclei Neuron aggiungono un quarto processore per l'elaborazione degli interrupt, il nucleo Neuron è un sistema on-a-chip con più processori, memoria e sottosistemi I/O. Al momento della creazione del Neuron Chip ad ogni core viene assegnato un codice univoco a livello globale di 48bit chiamato **NeuronID**, viene completata la produzione con l'installazione del **Neuron Firmware** associato al Neuron core.

5.1.2 Routers

Nascono allo scopo di fornire varie metodologie di comunicazione per tutte le esigenze possibili. I router possono essere utilizzati per controllare le sezioni di traffico della rete

e parti della rete dal traffico in un'altra sezione, aumentando il throughput totale e la capacità della rete. Strumenti di rete permettono la configurazione automatica dei router basati su topologia di rete, rendendo l'installazione degli stessi facile per gli installatori e trasparenti ai dispositivi. I Router permettono che una singola rete peer-to-peer supporti decine di migliaia di dispositivi. Un router ha due lati, ciascuno con un ricetrasmettitore appropriato per i due canali a cui è collegato il router, questi sono completamente trasparenti per l'operazione logica della rete, ma non necessariamente trasmette tutti i pacchetti, se configurato da uno strumento di integrazione di rete, i router intelligenti vengono a conoscere la configurazione del sistema per bloccare i pacchetti che non hanno destinatari sul lato opposto. Utilizzando un altro tipo di router, chiamato IP-852,è possibile estendersi su grandi distanze su reti wide-area come Internet, inoltre risulta funzionale anche nel funzionamento su diversi tipi di canali come Internet, intranet o una rete privata virtuale (VPN), possiamo stilare quindi una lista di possibili router messi a disposizione:

- i.LON SmartServer
- i.LON 600
- MPR-50 Multi-Port Router
- LonPoint Router

5.1.3 Interfaccia di Rete

Possiamo definirla come un modulo che viene utilizzato per collegare una serie di elaboratori elettronici. L'interfaccia di rete stessa non esegue un'applicazione ma fornisce uno strato 2 o strati 2 - 5 del Network, più un ricetrasmettitore strato 1, e il firmware per lo scambio di pacchetti con il computer collegato livello 2 o strato 5. Tra le interfacce più comuni possiamo identificare

- U10/U20 USB Network Interface
- i.LON SmartServer
- i.LON 600

5.1.4 Smart Servers

Uno Smart Server è un dispositivo programmabile che combina un controller con un server Web per l'accesso locale o remoto, Echelon definisce un server intelligente a basso costo che risponde al nome di **i.LON SmartServer**, questo ha il compito di collegare tra loro LonWorks, Modbus e M-Bus che fanno parte dei dispositivi per reti IP aziendali

o Internet. Smart Server è dotato di un server Web integrato che consente l'accesso Web a tutti i dati gestiti e controllati, così come le applicazioni integrate per la pianificazione, la registrazione e la traduzione dei dati. Inoltre comprende anche un'applicazione Web legante che permette di collegare vari domini LonWorks nonché bridging da dispositivi Modbus e M-Bus a domini LonWorks. Smart Server fornisce un'interfaccia SOAP / XML Web service per l'uso da pagine Web personalizzate e per l'integrazione con le applicazioni aziendali.

5.2 Gestione della Rete

Esiste una categorizzazione di quelle che sono le rete associate al protocollo LonWorks possiamo definire principalmente due tipologie di reti:

1. *Managed Networks*
2. *Self Installed Networks*

Una Menaged Networks è una rete in cui il Network Management Server viene utilizzato per stabilire l'installazione di rete, Il Network Management Server può essere parte di un sistema operativo di rete o può essere parte di un server Internet, come lo SmartServer. Viene data la possibilità all'utente di comunicare con il server tramite uno strumento che viene denominato *Network Management tool*, il suo utilizzo è fondamentale per apportare modifiche alla configurazione della rete. Il Network Management Server è consapevole della topologia di rete e può configurare i dispositivi per ottimizzare le prestazioni entro i limiti della topologia. L'alternativa a una rete gestita è una rete di Self Installed Networks dove non esiste uno strumento centrale o server che gestisce l'intera configurazione di rete ma ogni dispositivo contiene codice che sostituisce parti del funzionamento del Network Management Server con la conseguenza di non avere la necessità di uno strumento utile a stabilire comunicazione di rete o modificare la configurazione della rete. L'installazione della rete prevede un numero di passi specifici:

1. Assegnazione di indirizzi logici ai dispositivi
2. Esecuzione del binding tra le variabili di rete per la creazione di una connessione logica tra i device
3. Configurazione dei vari parametri di rete per le caratteristiche e le prestazioni.

L'installazione della rete può risultare complessa ma in ogni caso risulta nascosta dalle soluzioni di gestione di rete che fanno parte di LonWorks. Lo strumento di gestione della rete assegna automaticamente gli indirizzi logici, lega variabili di rete in funzione delle connessioni stabilite e configura i parametri di controllo della rete stessa. Possiamo dividere il processo di installazione di rete in due casistiche:

1. *ad-hoc:* i dispositivi vengono prima connessi alla rete e mandati a regime, i dati vengono automaticamente installati o scaricati dalla rete tramite uno strumento integrato.

2. *engineered:* le informazioni sono stoccate in database tramite l'utilizzo di uno strumento di rete integrato e scaricate, all'occorrenza, durante la fase di installazione.

Non esistite uno strumento centralizzato o un server che gestisce l'intera configurazione di rete è previsto che ogni dispositivo contenga parti di codice specifico per la gestione della rete e in particolare che sostituisca un server centralizzato. Con un conseguente alleggerimento della rete stessa che non prevede uno strumento di configurazione.

5.2.1 ISI

Le reti dette auto-installanti seguono quello che è il modello ISI (Interoperable Self-Installation) che ha come vantaggio oltre al basso costo, la forte adattabilità a reti semplici e con semplici, connessioni. Ogni componente è responsabile per la propria configurazione e non si basa su un server per determinare la stessa, detta questa caratteristica risulta quindi necessario l'utilizzo di uno standard comune affinchè i dispositivi si configurino in maniera compatibile tra loro, nel mondo LonWorks, questo standard corrisponde al nome di ISI. Questo protocollo permette la comunicazioni tra numerosi dispositivi con un limite massimo a 200 unità, se sono presenti un numero superiore di unità allora è necessario passare all'utilizzo di reti gestite, che quindi suddividono la rete totale in sottoreti di massi 200 unità a loro volta.

5.2.2 NOS

Per quando riguarda le reti gestite è possibile fare uso di un sistema operativo a livello di rete NOS (Network Operating System) per fornire un insieme di servizi a livello di rete per quanto riguarda il monitoraggio, il controllo di supervisione, installazione e configurazione. Il sistema operativo NOS fornisce quindi uno strumento di facile utilizzo per la manutenzione e la gestione della rete, inoltre prevede anche funzionalità aggiuntive per applicazioni HMI e SCADA fornendo la possibilià di gestione remota tramite IP internet. Il sistema NOS è stato progettato per fornire sincronizzazione tra più strumenti e applicazioni, per questo deve integrare internamente una completa interoperabilità con gli standard dei vari produttori per fornire una installazione facile. Va specificato che il sistema NOS viene utilizzato per la messa a regime degli elementi della rete ma una volta che questi ultimi risultano operativi e funzionanti allora il NOS ha terminato le sue funzionalità visto che non risulta essere fondamentale per la comunicazione tra i vari componenti che viene rimandata ad altri elementi, questo vantaggio deriva direttamente dalla natura del protocollo LonWorks che basa la comunicazione sul protocollo peer-to-peer(P2P) diversamente da altri procolli che usano il sistema gateway-to-gateway

che necessita della gestione rimandata al NOS non essendo in grado di separare le loro funzionalità dalla loro progettazione hardware.

5.2.3 LNS

Allo scopo di fornire interoperabilità LonWorks fornisce un solo NOS che specificatamente prende il nome di **LNS Netwok Operating System** che ha lo scopo di fornire una piattaforma standard per supportare applicazioni interoperabili LonWorks. LNS utilizza il modello client/server così da garantire la presenza di più attori allo stesso tempo attivi sulla rete, Echelon ventila altamente l'utilizzo del componente LNS ai vendor, così da evitare costi inerenti allo sviluppo di componentistica personalizzata e incoraggiare l'utiizzo di componente software già collaudato e pronto all'uso. In definitiva il componente LNS ha il compito di semplificare quella che è la visione della rete e nascondere sottostanti meccanismi di comunicazione tra componenti software e dispositivi fisici.

5.3 Strumenti di Rete

Identifichiamo con strumenti di rete applicazioni software che hanno tra le loro capacità strumenti per la progettazione della rete, configurazione, monitoraggio, controllo, diagnostica e manutenzione della stessa,questi applicativi vengono sviluppati superiormente al sistema operativo. Possiamo fare una categorizzazione delle possibili applicazioni rintracciabili:

- **Strumenti di integrazione di rete :** Forniscono le funzioni necessarie per progettare, configurare e manutenere una rete.
- **Strumenti di diagnostica :** Forniscono strumenti per analizzare, osservare e diagnosticare il traffico di rete.
- **Strumenti di sviluppo HMI :** Forniscono funzionalità utili a creare una interfaccia uomo-macchina
- **Strumenti di I/O dei server :** Forniscono funzionalità di middleware per rendere utilizzabili applicazioni non espressamente sviluppare per il protocollo LonWorks.

Echelon, sempre con il focus di fornire uno standard unificato, ha reso disponibile ulteriori due strumenti che forniscono intelligenza a dispositivi prodotti da vendor.

5.3.1 LonMaker Integration Tool

Fig.5.2

Il primo pacchetto che andiamo ad osservare fornisce strumenti per la progettazione, documentazione, installazione e manutenzione multi-vendor. Questo strumento deriva direttamente da LNS e combina una architettura client-server con una semplice interfaccia utente, ne risulta quindi uno strumento altamente specializzato ma con una user interface molto friendly per l'utilizzatore che riduce la complessità necessaria per operare nel mondo delle reti. La progettazione avviene tremite l'utilizzo di uno strumento grafico che permette la generazione della rappresentazione grafica di quella che è la nostra rete, inoltre à possibile definire nuovi tipi di elementi che possono essere sia singolo che accorpati in gruppi funzionali. l'aggiunta di nuovi elementi si attua tramite il solo trascinamento dell'elemento stesso e all'occorrenza ogni elemento della rete può essere definito un super-nodo. Ogni dispositivo è associato ad un univoco Id Neuron che permette di identificare il singolo dispositivo in tutta la rete progettata, così da ridurre drasticamente le tempistiche di gestione della stessa anche se di grande complessità. Lo strumento LonMaker è un singolo strumento espandibile che copre l'intero ciclo di vita della rete per semplificare le operazioni di installazione.

5.3.2 LonScanner Protocol Analyzer

Fig.5.3

Rappresenta un secondo pacchetto software che viene fornito da Echelon, questo software fornisce la capacità di osservazione, analisi e diagnostica del comportamento di una rete LonWorks. L'analizzatore di protocollo può essere utilizzato per raccogliere timestamp. I pacchetti vengono salvati in file di log che possono essere visualizzati e analizzati in seguito, in tempo reale come vengono raccolti dall'analizzatore di protocollo. I registri possono essere visualizzati in forma sintetica con un pacchetto per linea così da avere una analisi rapida o in forma espansa con un pacchetto per finestra per un'analisi più dettagliata. Fornisce inoltre le descrizioni testuali di ogni messaggio e la descrizione del servizio di messaggi CNP utilizzato per trasmetterla, eliminando la necessit per l'utente di interpretare manualmente il codice binario CNP per riduce conseguentemente il tempo e lo sforzo necessario per diagnosticare problemi di rete. L'utente a sua discrezione può specificare i filtri di acquisizione per limitare i pacchetti raccolti. Il software inoltre integra uno strumento che va a rappresentare le statistiche di traffico, in maniera dettagliata, relative al comportamento della rete. Le varie statistiche includono un counter dei pacchetti, un counter dei pacchetti di errore e la rete di carico, il tutto viene rappresentato a display per avere una più facile lettura delle statistiche in tempo reale.

Capitolo 6

Conclusioni

Dallo sviluppo di questa tesi abbiamo potuto apprendere lo stato dell'arte di quelli che sono i principali protocolli utilizzabili nell'emergente mondo IoT. La comprensione di questi protocolli è fondamentale per comprendere meglio il mondo IoT che promette una espansione enorme e quindi una modifica futura sostanziale di quella che è la nostra visione di internet e dei servizi ad esso collegati. Grazie alla prevista introduzione massiva di dispositivi "intelligenti" possiamo solo immaginare come varierà la qualità della vita nell'arco del medio e lungo termine. Analizzando i vari trend tecnologici possiamo osservare un massivo investimento dei "big" della tecnologia nel mondo IoT con l'immissione sul mercato di dispositivi, fino ad oggi completamente "analogici", dotati di capacità computazionali e relativo I/O. Se accoppiamo questa classe di prodotti analogici, oltre a dotarli di capacità computazionale, con un parco sensoristica completo e quel tanto che basta di intelligenza ecco che non risulta troppo immaginario il mondo creato da Tesla con le sue creature avveniristiche. Auto fino ad oggi completamente passive che da soli pochi anni hanno appreso la capacità di guidare senza l'ausilio dell'uomo e tramite l'implementazione di algoritmi ad hoc capaci perfino di essere predittive nel tempo. Lo scenario che ci stiamo preparando a vivere sembra derivare da un romanzo di metà novecento ma tutta questa innovazione può portare ad utilizzo nocivo perchè come può essere rassicurante, per esempio, il controllo remoto di un velivolo piuttosto che di una nave ecco questa casistica può creare scenari pericolosi se nelle mani di mal intenzionati. Risulta quindi chiaro che la standardizzazione e il controllo di queste tecnologie avveniristiche deve essere gestito sin dal livello delle reti perchè è proprio questo il livello più delicato, il livello che deve garantire sicurezza e non vulnerabilità. Ad oggi possiamo stimare il numero di oggetti detti "intelligenti" che è pari a 40 milioni di elementi, con una curva di crescita estremamente ripida che porterà nel breve/medio periodo ad essere completamente circondati da dispositivi IoT. Secondo gli analisti quindi il classico internet visto come una "enciclopedia" mondiale andrà a mutare divenendo uno strumento con cui si potrà migliorare e ottimizzare l'esistenza dell'uomo in generale e magari demandare tutti quei compiti altamente pericolosi, oggi affidati a personale umano. In definitiva per quan-

to riguarda la parte centrale della presente tesi, possiamo affermare che i tre protocolli risultano essere allineati in termini puramente tecnologici mentre per quanto concerne il parco applicativo sono evidenti differenze in funzione delle performance richieste e quindi della capacità computazionale dei singoli dispositivi. Risulta quindi naturale la scelta di MQTT per una infrastruttura costituita da dispositivi a bassa capacità computazionale mentre in ambito business è preferibile optare per OPC-UA piuttosto che LonWorks, entrambi capaci di garantire ottime prestazioni ma con differenti header file quindi non comunicabili tra loro.

Capitolo 7

Bibliografia

1. Oasis standard (Aprile 2018), MQTT Version 3.1.1 , http://docs.oasis-open.org /mqtt/mqtt/v3.1.1/os /mqtt-v3.1.1-os.html
2. LonMark Italia, https://www.lonmark.it/tecnologia-lon/
3. Echelon standard, LonWork, https://www.echelon.com/
4. OPC-UA Foundation, https://opcfoundation.org/about/opc-technologies/opc-ua/
5. OPC-UA Foundation, https://opcfoundation.org/wp-content/uploads /2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf
6. Mauro Bellini (Novembre 2016), https://www.internet4things.it/iot-library/internet-of-things-gli-ambiti-applicativi-in-italia/

Capitolo 8

Ringraziamenti

Giunto al termine del presente elaborato di tesi, vorrei cogliere l'occasione per ringraziare in prima istanza il professor Callegati Franco, per la sua disponibilità e il suo aiuto prezioso. Ringrazio i miei genitori, Lina e Filippo, mia sorella Ilenia per avermi sempre supportato e spronato nell'affrontare questo percorso, anche nei momenti più difficili. Ringrazio tutti i colleghi e i compagni di corso con cui ho condiviso ansie e gioie, mentre un particolare ringraziamento va a Anna per avermi sempre spronato. In ultimo ma non per minor importanza, ringrazio Laura per essermi sempre vicina, che siano gioie o dolori. Ringrazio, in definitiva, tutti coloro che hanno partecipato attivamente o meno a questo percorso.