ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA
SEDE DI CESENA

SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Bluetooth per TuCSoN mobile: panorama tecnologico ed esperimenti

Tesi di Laurea in Sistemi Distribuiti

Relatore:
Chiar.mo prof.
Andrea Omicini
Correlatore
Dott. Ing.
Stefano Mariani

Presentata da:
Svetlozar Orlovski

Anno Accademico 2014/2015

Sessione III

Alla mia famiglia che mi è stata sempre vicina
e mi ha dato appoggio nelle difficoltà di questo percorso.
A tutti i miei professori che mi hanno trasmesso la loro passione.

Introduzione

A giorno d'oggi si sente parlare sempre più spesso di Internet Of Things (IoT) ed effettivamente siamo circondati da dispositivi in grado di svolgere i task per i quali sono stati progettati sia nella maniera classica, cioè controllati direttamente dall'utente, ma anche in autonomia, visto che sono controllabili remotamente grazie alla loro connettività. Questo ci permette di rendere il loro funzionamento più intelligente. Questo comportamento intelligente si ottiene non soltanto collegando i dispositivi tra loro per poterli controllare, ma soprattutto si è in grado di coordinarli nel modo opportuno per ottenere il funzionamento organizzato dei singoli componenti senza l'intervento da parte dell'utente. TuCSoN è un middleware di coordinazione per agenti, ed il suo utilizzo potrebbe permetterci di ottenere questo risultato. Purtroppo però allo stato attuale, non tutti i dispositivi sono in grado di interagire con TuCSoN. Sia perché non tutti utilizzano Java, ma soprattutto perché non tutti sono collegati direttamente a una rete TCP/IP. Nel mondo IoT spesso i dispositivi sono dotati di tecnologie di connettività diverse tipo Bluetooth, ZigBee per citare due delle più utilizzate. Vista la natura minimalista dei dispositivi, anche quando sono dotati della possibilità di interagire direttamente con una rete TCP/IP il linguaggio con il quale vengono programmati non è Java.

Questa tesi si è posta come obiettivo di analizzare quali sono i limiti e le possibilità di estensione di TuCSoN aggiungendo la possibilità di utilizzare Bluetooth. In particolare si è pensato di implementare la comunicazione con Bluetooth anche per le comunicazioni nell'infrastruttura di TuCSoN, seguendo lo stesso modello di quello già implementato rendendo indipendente la comunicazione dal protocollo di collegamento utilizzato. Analizzando in dettaglio i limiti tecnologici, ci si è resi conto delle difficoltà di un approccio simile visti anche i limiti attuali della tecnologia Bluetooth e la tesi è stata orientata più verso un analisi dei problemi e la ricerca di una strada alternativa per permettere la possibilità di interagire con queste tecnologie.

Indice

1 Tecnologie....14
1.1 Sistemi Distribuiti....14
1.2 TuCSoN....15
1.2.1 Agenti....16
1.2.2 Centri di tuple....16
1.2.3 Nodi TuCSoN....17
1.2.4 Comunicazione....17
1.2.4.1 Naming....17
1.2.4.2 Operazioni del modello base di TuCSoN....19
1.3 TCP/IP....21
1.3.1 Livello fisico....23
1.3.2 Livello collegamento dati....24
1.3.3 Livello rete....24
1.3.4 Livello di trasporto....27
1.3.5 Livello applicativo....28
1.3.6 Ricapitolando....28
1.4 Bluetooth....29
1.4.1 Origini di Bluetooth....29
1.4.2 Caratteristiche e evoluzione....30
1.4.3 Reti Bluetooth....32
1.4.4 Struttura dello stack Bluetooth e comparazione con TCP....35
2 Analisi delle tecnologie....41
2.1 Analisi implementazione attuale in TCP/IP....41
2.2 Bluetooth e API Java....42
2.2.1 BlueCove....45
2.2.2 Avetana....46
2.2.3 Rocco....47
2.3 Bluetooth e Android....47
3 Integrazione TuCSoN e Bluetooth....49
3.1 Problema di indirizzamento e Naming....49

3.2 Topologia e Routing nelle reti Bluetooth.....51
3.3 Disponibilità delle API Android su architettura x86.....53
4 Conclusioni.....58
5 Sviluppi futuri e approfondimenti.....60
5.1 Raspberry Pi.....60
5.2 Routing Ad-Hoc.....60
5.3 Layer di presentazione.....61

Parte I

Tecnologie utilizzate

1 Tecnologie

In questo capitolo farò una panoramica delle tecnologie analizzate durante lo svolgimento della tesi.

1.1 Sistemi Distribuiti

Ci sono molte definizioni diverse dei sistemi distribuiti a seconda della prospettiva dalla quale viene utilizzato. Dal punto di vista dell'utilizzatore, un sistema distribuito è un insieme di computer che appaiono ai loro utilizzatori come un singolo sistema. Dal punto di vista ingegneristico, i sistemi distribuiti sono un insieme di entità computazionali indipendenti progettate in modo che il loro funzionamento simuli un singolo sistema.

Un diagramma semplificato di un'applicazione distribuita:

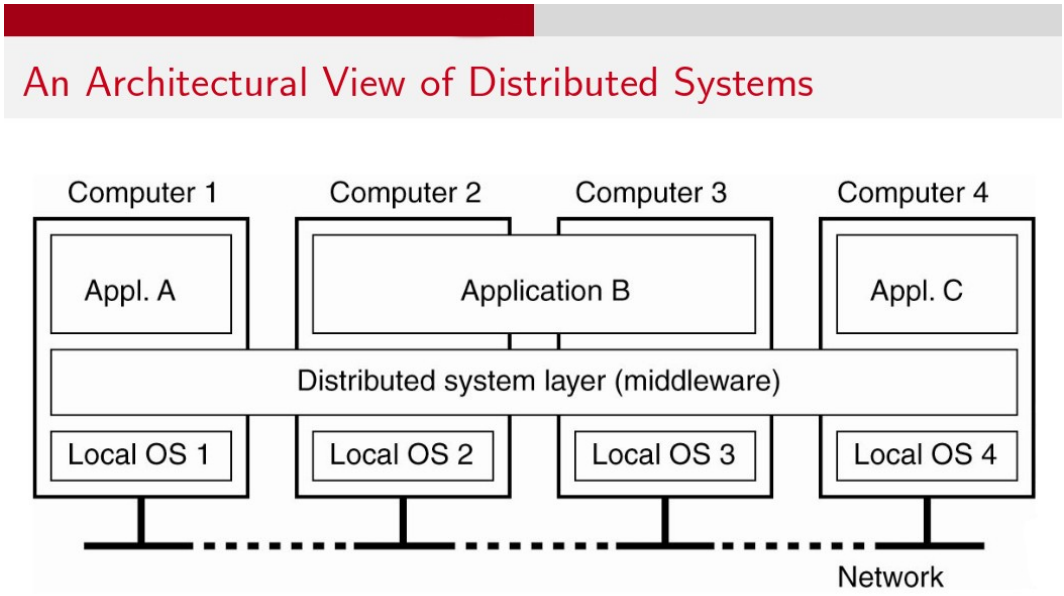

Figura 1.1: struttura sistema distribuito

L'applicazione in se è composta da un insieme di elementi indipendenti i quali “lavorano” in modo coordinato per eseguire un determinato task. I vari

componenti sono distribuiti su diverse macchine fisiche in rete e necessitano di coordinazione e comunicazione tra di loro per poter conseguire i task dell'applicazione distribuita.

Il componente più importante di un sistema distribuito è il middleware di coordinazione/comunicazione il quale sfrutta le risorse del sistema operativo sottostante per permettere lo scambio di messaggi/dati tra le varie entità che compongono il sistema distribuito. Grazie a questo scambio di messaggi/dati è possibile realizzare la coordinazione delle entità autonome che compongono il sistema. Come si vedrà più avanti TuCSoN è proprio un middleware di coordinazione/comunicazione. Per permettere la comunicazione e la coordinazione il middleware incapsula molte delle complessità legate alle comunicazioni via reti. Inoltre incapsula in se la gestione dei problemi legati al fatto che il sistema non è monolitico ma distribuito. Alcuni dei problemi riguardano la fault tolerance e l'eterogeneità dei componenti da coordinare.

1.2 TuCSoN

In questo capitolo sarà illustrato cosa è TuCSoN e il funzionamento del modello base, soprattutto dal punto di vista delle comunicazioni. Aspetti avanzati non sono interessati dal lavoro svolto in quanto il loro comportamento non dipende dal modo un cui è strutturata la comunicazione.

TuCSoN è un'infrastruttura (middleware) rilasciata con licenza GPL che si pone l'obiettivo di agevolare la comunicazione, l'organizzazione e la coordinazione di entità computazionali autonome (agenti) distribuite che fanno parte di un sistema multiagente. L'infrastruttura concettuale di TuCSoN prevede tre entità: agenti, centri di tuple e nodi. TuCSoN è scritto in Java che gli permette di essere un middleware multipiattaforma (Windows, Linux oppure qualsiasi piattaforma per la quale esiste una JVM compatibile con Java SE 7). Inoltre è stata sviluppata un'applicazione Android (la quale include vari tool di supporto) per poter sfruttare TuCSoN da una applicazione nativa Android. Lo scambio di messaggi tra le entità in TuCSoN avviene tramite Tuple che vengono manipolate

tramite le operazioni disponibili in TuCSoN.

1.2.1 Agenti

Gli Agenti sono autonomi, attivi e indipendenti, distribuiti nella rete. Sono le entità base da coordinare nel sistema. Per coordinarsi e scambiarsi informazioni, si interfacciano con vari centri di tuple che possono essere distribuiti su diversi nodi TuCSoN. Gli agenti sono entità proattive, la comunicazione con i centri di tuple avviene attraverso diverse operazioni disponibili nel linguaggio di coordinazione TuCSoN che verrà discusso più avanti. Comunque per effettuare la comunicazione viene sfruttata una connessione via rete TCP/IP.

1.2.2 Centri di tuple

I centri di tuple sono delle entità passive/reattive contenute nei nodi TuCSoN le quali contengono al loro interno le tuple di coordinazione/informazione le quali possono essere consultate/modificate dagli agenti e dal centro di tuple stesso. Il comportamento dei centri di tuple non è solo passivo, cioè non si limita solo ad offrire uno spazio condiviso nel quale gli agenti possono comunicare/coordinarsi inserendo oppure leggendo tuple tramite le operazioni definite nel linguaggio TuCSoN, ma grazie al linguaggio ReSpecT è possibile definire, nel centro di tuple, un insieme di REACTION, cioè delle reazioni con le quali un centro di tuple potrebbe eseguire operazioni computazionali programmabili sia a priori (in fase di creazione del centro di tuple) che durante l'esecuzione (inserendo o cancellando REACTION da parte degli agenti). Questo rende i centri di tuple entità programmabili e reattive dando la possibilità di eseguire operazioni computazionali anche complesse in modalità trasparente per quanto riguarda gli agenti.

1.2.3 Nodi TuCSoN

I nodi sono l'astrazione base di TuCSoN. Come detto prima, contengono al loro interno i centri di tuple e possono anche contenere agenti. Sono strettamente legate a una macchina fisica. Cioè risiedono su un server (oppure dispositivo Android).

1.2.4 Comunicazione

Come già accennato, la comunicazione eseguita dalle operazioni TuCSoN avviene attraverso la rete TCP/IP. Nel capitolo seguente ci sarà un'analisi più dettagliata sia del TCP/IP che della implementazione della comunicazione in TuCSoN. In questa sezione verranno illustrate le basi della comunicazione con l'astrazione scelta da TuCSoN.

1.2.4.1 Naming

Per identificare le varie entità in un sistema distribuito, ognuna deve avere un suo identificativo univoco. Essendo distribuiti nella rete questo aggiunge un ulteriore livello di complessità. Non è sufficiente la coppia IP/porta TCP del layer di rete sottostante per identificare le entità in quanto sullo stesso indirizzo di rete potrebbero esserci istanziati diversi centri di tuple, nodi o agenti.

Gli identificativi delle entità TuCSoN sono strutturati in questo modo.

I nodi TuCSoN sono identificati con questa sintassi:

netid : portno

- **Netid** è l'indirizzo IP della macchina sulla quale è istanziato il nodo
- **Portno** è la porta TCP sulla quale il nodo è in ascolto

Questa sintassi ci permette di identificare univocamente il nodo visto che sullo stesso indirizzo IP (macchina) potrebbero essere istanziati diversi nodi, ma su porte TCP differenti. La porta di default utilizzata da TuCSoN è 20504. È sempre presente un nodo con un centro di tuple di default su questa porta.

I centri di tuple sono identificati con questa sintassi:

tname @ netid : portno

- **Tname** rappresenta il nome del centro di tuple, univoco sul nodo sul quale è istanziato
- **Netid** è l'indirizzo IP del nodo sul quale è presente il centro di tuple desiderato
- **Portno** è il numero di porta TCP sul quale il nodo che ha al suo interno il centro di tuple con quel nome è in ascolto

La combinazione di questi tre parametri permette di identificare univocamente un centro di tuple e anche il nodo sul quale è istanziato. E' possibile creare diverse combinazioni di diversi centri di tuple istanziati sullo stesso nodo, ma con nomi diversi, oppure avere vari centri di tuple con lo stesso nome su nodi diversi.

In fine, gli agenti in TuCSoN, dovrebbero essere riconoscibili e distinguibili univocamente. Per l'identificazione degli agenti la sintassi utilizzata è la seguente:

aname : uuid

- **Aname** è il nome dell'agente definito da lui stesso (dall'utente quando lo crea)
- **UUID** è un identificativo univoco generato dal nodo TuCSoN e

restituito all'agente con invocazione della prima operazione

Nelle operazioni successive l'agente dovrà usare quel identificativo per poter eseguire le operazioni. (nel modello esteso di TuCSoN in realtà esiste l'ACC che rappresenta un interfaccia che incapsula l'UUID e definisce regole per la sicurezza in modo che il comportamento di ogni agente sia limitato soltanto alle operazioni a lui assegnate/permesse).

1.2.4.2 Operazioni del modello base di TuCSoN

In questo paragrafo elencherò le operazioni di coordinazione del modello base di TuCSoN. La sintassi utilizzata per invocare le operazioni da parte degli agenti è la seguente:

tname @ netid : portno ? op

Dove la prima parte identifica il nodo e il centro di tuple sul quale l'operazione OP deve essere eseguita. Le operazioni previste dal modello base sono 9.

- **out(Tuple) –** scrive la tupla desiderata nel centro di tuple di destinazione. Quando l'operazione ha successo la tupla scritta viene restituita per confermare la operazione. In caso di errore viene restituita una tupla vuota
- **rd(TupleTemplate) –** cerca nel centro di tuple una tupla che combacia con il template (TupleTemplate). Se la tupla esiste, viene restituita, se nessuna tupla tra quelle nel centro di tuple corrisponde al template, l'operazione viene sospesa finché non compare una tupla che rispetta il template
- **in(TupleTemplate)** - cerca nel centro di tuple una tupla che combacia con il template (TupleTemplate). Se la tupla esiste, viene rimossa dal centro di tuple e restituita per notificare il successo dell'operazione. Se nessuna

tupla nel centro di tuple corrisponde al template, l'operazione viene sospesa finché non compare una tupla che rispetta il template

- **rdp(TupleTemplate)** - cerca nel centro di tuple una tupla che combacia con il template (TupleTemplate). Se la tupla esiste, viene restituita, se nessuna tupla nel centro di tuple corrisponde al template, viene restituito il template e l'operazione viene considerata conclusa
- **inp(TupleTemplate)** - cerca nel centro di tuple una tupla che combacia con il template (TupleTemplate). Se la tupla esiste, viene rimossa dal centro di tuple e restituita per notificare il successo dell'operazione. Se nessuna tupla nel centro di tuple corrisponde al template viene restituito il template e l'operazione viene considerata conclusa
- **no(TupleTemplate)** - cerca nel centro di tuple una tupla che combacia con il template (TupleTemplate). Se la tupla non esiste, viene restituita la TupleTemplate e l'operazione viene conclusa. Se nel centro di tuple esiste una o più tuple che combaciano con il TupleTemplate, l'operazione viene sospesa finché quelle tuple esistono
- **nop(TupleTemplate)** - cerca nel centro di tuple una tupla che combacia con il template (TupleTemplate). Se la tupla non esiste, viene restituita la TupleTemplate e l'operazione viene conclusa con successo. Se nel centro di tuple esistono tuple che combaciano con il TupleTemplate, l'operazione fallisce e la tupla che combacia il template viene restituita per notificare l'esito.
- **get –** Legge tutte le tuple di un centro di tuple e le restituisce come lista di tuple. Se non ci sono tuple, viene restituita una lista vuota
- **set(Tuples) –** Scrive la lista di tuple nel centro di tuple. Quando l'esecuzione finisce, la lista viene ritornata per confermare l'esito

1.3 TCP/IP

In questo capitolo verranno illustrati la struttura dello stack TCP/IP e menzionate alcune delle sue fondamenta per agevolare la comprensione della analisi nei successivi capitoli e delle varie problematiche emerse in questa tesi. Visto che nel capitolo seguente sarà analizzata la struttura e le funzionalità dello stack Bluetooth, nell'analisi di entrambi gli stack si farà riferimento allo stack concettuale ISO/OSI per avere una base di confronto.

TCP/IP o Transmission Control Protocol è lo stack di protocolli di rete sul quale si basa internet e le reti LAN. E' organizzato a livelli. Ciascuno dei livelli incapsula meccanismi per la risoluzione di diversi problemi legati alla trasmissione dei dati nelle reti sottostanti.

La struttura dei livelli di TCP confrontata con quella del modello ISO/OSI

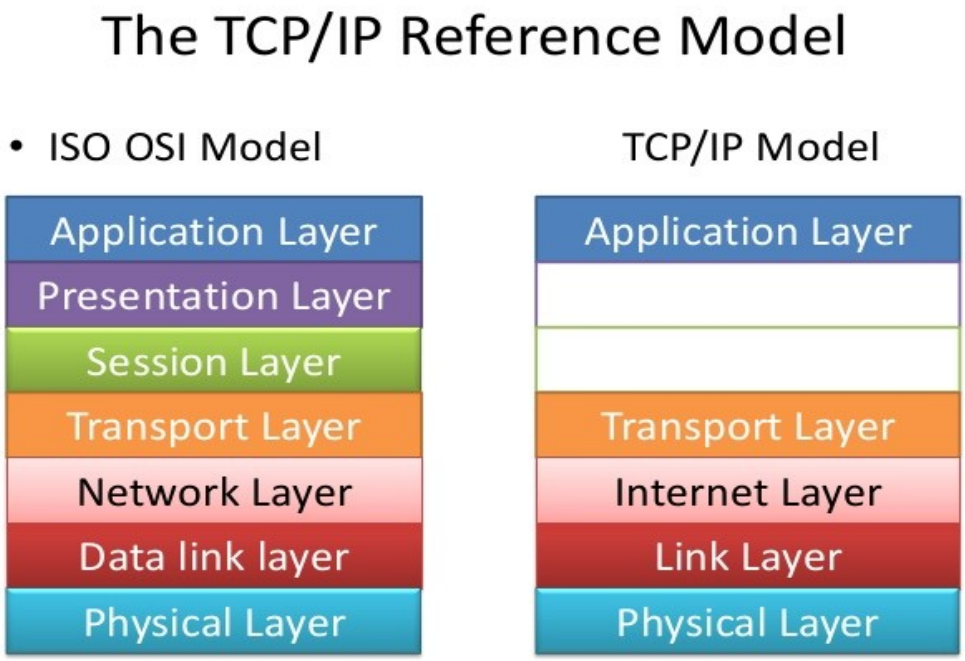

Figura 1.3: Struttura TCP e OSI a confronto

Ogni livello incapsula una parte dei problemi legati alla trasmissione e offre un servizio ai livelli superiori. I vari livelli possono utilizzare i servizi forniti soltanto dal livello sottostante.

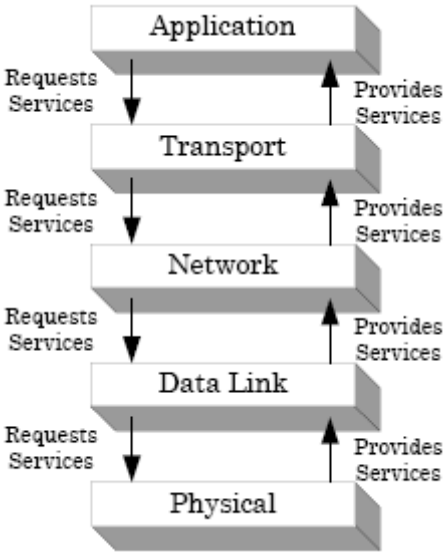

Figura 1.2: Struttura delle comunicazioni nello stack TCP/IP

Come si vede dal immagine 1.2 non tutti i livelli sono implementati dentro allo stack. Alcuni dei livelli OSI infatti vengono implementati dalle applicazioni che utilizzano TCP/IP.

1.3.1 Livello fisico

Il livello più basso dello stack TCP/IP è il layer fisico che corrisponde al livello fisico nel modello OSI. Questo livello si occupa della rappresentazione fisica dei dati nei mezzi trasmissivi e della trasmissione/ricezione dei dati nel mezzo, della risoluzione delle collisioni e in generale, di tutte le specifiche necessarie per la trasmissione/ricezione (tensioni, modulazioni, frequenze ecc). Durante la trasmissione dei dati in internet da un endpoint ad un altro, i dati

attraversano un insieme di apparati intermedi che possono avere mezzi trasmissivi differenti. Per esempio, durante la navigazione web da un dispositivo mobile, la prima trasmissione tra l'apparato e il router/accesspoint avviene attraverso le onde radio della wifi (802.11) successivamente il router/accesspoint invierà gli stessi dati, ma ad un altro dispositivo, eventualmente attraverso mezzo differente. Ad esempio via cavo (Ethernet) al router/modem che a sua volta invierà i dati via altro mezzo trasmissivo (onde,cavo,fibra) al ISP (Internet Service Provider) il quale continuerà la ritrasmissione nella sua struttura in mezzi diversi anche in fibra ottica dove per esempio gli stessi dati inizialmente trasmessi con onde radio sono rappresentati dalla luce. La definizione di tutti gli standard necessari per la trasmissione dei datagram insieme formano il layer fisico.

Il perché delle ritrasmissioni dei datagram e dei meccanismi utilizzati per decidere quale è l'apparato successivo al quale trasmettere i dati diventerà più chiaro con la trattazione degli strati più alti dello stack.

L'implementazione di questa parte dello stack non fa parte del sistema operativo, ma fa parte direttamente dal firmware del dispositivo utilizzato per la trasmissione. Inoltre in questo layer di comunicazione sono inclusi alcuni meccanismi per il controllo della integrità dei dati che dipendono dal mezzo. Il riferimento a dove si trova concretamente questo layer è importante per l'analisi successiva dello stack Bluetooth. In pratica il sistema operativo utilizza i driver del device per passargli i dati senza poter controllare tutti gli aspetti della trasmissione (modulazione, voltaggio, lunghezza dei datagram), a meno che il device non preveda la possibilità di modificare alcuni dei parametri .

1.3.2 Livello collegamento dati

Il secondo layer del TCP/IP è il Network Interface il quale corrisponde a DataLink del modello ISO/OSI. Questo livello è il primo ad essere implementato dentro al sistema operativo e non nel firmware del dispositivo hardware utilizzato. Si occupa della trasmissione dei dati chiamati Frame. In TCP/IP, in questo layer è introdotta anche la prima forma di indirizzamento attraverso l'indirizzo MAC

(Media Access Control) che serve per identificare i vari dispositivi i quali possono accedere al mezzo di trasmissione. Il MAC address è una sequenza di 48 bit assegnata dal produttore del hardware, anche se è possibile impostare un indirizzo diverso con il quale presentarsi in rete, la quale identifica il dispositivo in modo univoco.

È importante notare che con questo layer, concettualmente è possibile realizzare le comunicazioni dentro una rete che condivide lo stesso mezzo trasmissivo come veniva fatto in alcuni protocolli non più utilizzati.

1.3.3 Livello rete

Il livello successivo è IP (Internet Protocol), il quale corrisponde al Network Layer del modello OSI. È il livello più importante dello stack di comunicazione ed il livello nel quale vengono gestite la maggior parte delle problematiche legate alla interconnessione di reti. Essendoci tanti argomenti, molti dei quali sono aspetti generali legati alle reti in se, in questa tesi verranno trattati soltanto a quelli necessari alla analisi delle problematiche legate alla tesi. Essendo aspetti molto complessi e articolati, non possono essere dati per scontati, ma il modo in qui verranno trattati in questa tesi avrà come obbiettivo non di spiegarli in modo approfondito, ma soltanto di far capire in grandi linee il loro funzionamento e soprattutto far capire al lettore quali sono i problemi concettuali compresi in questo layer.

Anche questo livello dello stack è implementato nel sistema operativo. Introduce un altro tipo di indirizzamento virtuale da non confondere col MAC address del layer fisico. Gli indirizzi del livello IP sono appunto indirizzi IP. Gli indirizzi sono nati a 32 bit (4 byte) (IPv4 anche se in fase di aggiornamento a 128 bit in IPv6). Per semplificare la scrittura degli indirizzi, e renderla facilmente comprensibile, gli indirizzi vengono scritti utilizzando 4 numeri da 0 a 254 separati (ogni uno dei quali rappresenta il valore di uno dei 4 byte che compongono l'indirizzo) da un punto (per esempio 137.204.71.146). Gli indirizzi sono suddivisi in pubblici e privati. Ci sono 3 reti dedicate agli indirizzi privati

che possono essere utilizzate liberamente nelle proprie reti (10.0.0.0/8, 172.16.0.0/16 e 192.168.0.0/24). Oltre l'indirizzo IP, ogni macchina ha una netmask, che è un numero da 32 bit il quale serve per identificare la grandezza della rete. Brevemente spiegato, la netmask è una maschera di bit, da affiancare al indirizzo IP per identificare se un dato indirizzo IP appartiene alla rete locale e possiamo raggiungerlo direttamente, oppure no. È composto da due parti, la prima parte contiene degli 1 i quali delimitano il prefisso della rete, la seconda parte è composta soltanto di 0 i quali identificano la parte di rete direttamente raggiungibile. Questo ci porta al meccanismo di instradamento (routing) utilizzato per trasmettere i pacchetti nella rete. Il motivo della netmask verrà chiarito sotto. Prima di affrontare il meccanismo di routing in IP, vediamo come è fatta la tabella di routing sulla quale si basa l'instradamento IP.

Kernel IP routing table

Destination	*Gateway*	*Genmask*	*Flags*	*Metric*	*Ref*	*Use*	*Iface*
192.168.20.0	*0.0.0.0*	*255.255.255.0*	*UH*	*0*	*0*	*0*	*eth0*
192.168.70.0	*0.0.0.0*	*255.255.255.0*	*UH*	*0*	*0*	*0*	*eth1*
10.100.0.0	*0.0.0.0*	*255.255.0.0*	*UH*	*0*	*0*	*0*	*eth2*
0.0.0.0	*192.168.20.254*	*0.0.0.0*					*eth0*

Figura 1.3.3: *Tabella di routing Linux*

Ogni volta che un nodo della rete deve inviare dati ad un altro nodo, deve prima sapere dove inviarli. Per capire se il destinatario è direttamente raggiungibile il nodo esegue un AND con l'indirizzo di destinazione e la netmask. Questa operazione permette di individuare il suo prefisso di rete. Se il prefisso coincide con il prefisso di qualche rete locale direttamente raggiungibile, il pacchetto viene trasmesso direttamente. Se no, viene trasmesso a un gateway il quale dovrà a sua volta spedirlo più vicino possibile (passandolo dentro una delle reti al quale lui è connesso) al destinatario. La ritrasmissione avviene finche il pacchetto non raggiunge il destinatario (oppure finché il TTL non diventi 0). Per fare un esempio collegato alla tabella di un host Linux posta sopra come esempio,

pensiamo a un pacchetto che dovrebbe essere spedito a destinazione 192.168.20.6. Facendo l'AND tra l'indirizzo è la netmask vediamo che questo indirizzo appartiene alla rete collegata al scheda di rete (NIC) eth0 e può essere spedito direttamente al destinatario. Se si dovesse trasmettere un pacchetto con indirizzo di destinazione 173.194.112.47 facendo AND con le proprie netmask, l'indirizzo non coinciderebbe con nessuna delle reti direttamente connesse. In questo caso il pacchetto verrà trasmesso al gateway che nell'esempio della tabella sopra è 192.168.20.254 sempre appartenente alla rete eth0. Il gateway a sua volta dovrà fare lo stesso lavoro con la propria tabella di routing per indirizzare il pacchetto verso la rete di destinazione.

Ricapitolando, se il nodo di invio non è connesso direttamente al nodo di destinazione (non è nella stessa rete), invierà il dato al gateway che conosce, il quale è più vicino al nodo di destinazione, che poi stabilirà a sua volta come instradare il pacchetto dati alla destinazione corretta. Ogni nodo ha quindi bisogno di tenere traccia delle rotte necessarie all'instradamento dei pacchetti di dati, e per questo si utilizza una tabella di routing. Ovviamente la tabella varia nelle sue dimensioni e complessità da nodo a nodo. La tabella di routing riportata sopra come esempio è la tabella di un nodo Linux collegato a 3 reti LAN (eth0,eth1,eth2) e in una delle reti (eth0) è presente un gateway (significa che è connesso a reti che noi non conosciamo) il quale permette di instradare i pacchetti nelle reti non accessibili da noi direttamente.

Le tabelle di routing negli host normali sono statiche, ma nei router connessi a molte reti le tabelle sono dinamiche generate con dei protocolli (ICMP e IGMP) con i quali i router si coordinano per creare e aggiornare le proprie tabelle di routing con i migliori percorsi.

A parte permettere di collegare più reti, altri aspetti gestiti da questo layer sono il controllo di flusso, la gestione delle congestioni. Da notare che non ci sono connessioni dirette (il protocollo è conectionless), ma semplicemente l'invio di pacchetti di dati. Senza garanzie sulla qualità o su i tempi di consegna dei pacchetti.

1.3.4 Livello di trasporto

Infine il TCP il quale è uno dei diversi protocolli di trasporto dello stack TCP/IP. Corrisponde al Transport Layer di ISO/OSI. TCP è il protocollo che sta più in alto nello stack TCP/IP ed anche questo fa parte del sistema operativo.

Questo protocollo offre alle applicazioni la possibilità di accedere alla rete, incapsulando tutta la complessità della rete, sottostante permettendo di creare un collegamento virtuale tra due endpoint. Il layer gestisce il flusso di trasmissione dei dati, correggendo gli eventuali errori o ritrasmettendo i pacchetti IP persi, così garantendo l'integrità dei dati. A parte la ritrasmissione, per garantire l'integrità dei dati utilizza meccanismi di controllo su i dati per garantire che non sono stati alterati a causa delle ritrasmissioni nella rete.

A livello TCP, non abbiamo degli indirizzi, ma porte (65535 porte) le quali servono sostanzialmente per distinguere le applicazioni software che accedono alla rete. In pratica, per accedere alla rete un software ha bisogno di “agganciarsi” a una delle porte e “collegarsi” alla porta del software con il quale vorrebbe scambiare dati. Per fare ciò l'unica cosa che l'applicazione deve fare è richiedere il collegamento fornendo la coppia IP e porta TCP per identificare la macchina (IP) e l'applicazione (TCP). Il collegamento come sappiamo è virtuale, cioè i dati viaggiano in pacchetti i quali vengono trasmessi separati, anche su percorsi diversi, ma questo layer ha il compito di riassemblarli nell'ordine opportuno.

1.3.5 Livello applicativo

Come indicato nella figura di confronto tra i protocolli, in ISO/OSI ci sono altri tre livelli. L'implementazione dei restanti livelli dello stack ISO/OSI nei sistemi moderni è lasciato alle applicazioni. E' importante citare il layer di presentazione (Presentation Layer) il quale viene trattato successivamente nella tesi.

Il layer di presentazione si occupa di rendere i dati delle applicazioni su due endpoint che comunicano comprensibili tra loro. Visto che la presentazione

locale dei dati potrebbe essere differente (bigendian/littlendian, 32/64bit, sistemi operativi differenti ecc). In pratica è un traduttore. Per capire meglio il concetto pensiamo al caso nel quale una delle due applicazioni vorrebbero scambiarsi un dato di tipo String (Ogetto Java). Il ricevente potrebbe non “capire” cosa sono i dati inviati dal mittente perché magari rappresenta localmente il dato String diversamente (JVM differente oppure linguaggio di programmazione differente) o addirittura potrebbe non conoscere l'astrazione oggetto. Cioè nell'applicazione serve un meccanismo di “traduzione”, un accordo su come “confezionare/interpretare” i dati trasmessi/ricevuti tra le entità che comunicano.

1.4 Bluetooth

In questo capitolo sarà illustrato come è nata e come si è evoluta la tecnologia Bluetooth concentrandomi soprattutto sugli aspetti coinvolti nel raggiungimento degli obiettivi della tesi. Inoltre, nella seconda parte, analizzerò come è strutturato lo stack di comunicazione illustrando le differenze tra reti Bluetooth e reti TCP/IP. Userò sempre lo stack ISO/OSI come base per fare i confronti tra TCP/IP e Bluetooth.

1.4.1 Origini di Bluetooth

Bluetooth (BT) è una tecnologia sviluppata da Ericson nel 1994 per lo scambio di dati senza fili a distanze non superiori a 10 metri. Inizialmente nata come per alternativa senza fili alla RS-232 (collegamento seriale per la trasmissione di dati). Attualmente è utilizzato come standard per la trasmissione dati per reti senza fili conosciute come WPAN - Wireless Personal Area Network (Rete personale senza fili). Anche se la tecnologia è stata inizialmente sviluppata da Ericson, è stata formalizzata dalla Bluetooth Special Group of Interest (SIG). La SIG è nata nel maggio del 1999 come un associazione formata da SonyEricson, IBM, Toshiba, Nokia e altri con lo scopo di creare uno standard

comune, ma anche con lo scopo di certificare la compatibilità dei vari prodotti.

1.4.2 Caratteristiche e evoluzione

La tecnologia Bluetooth è stata sviluppata con gli obiettivi di avere dei dispositivi con bassi costi di produzione, bassi consumi e comunicazione a corto raggio. Il raggio varia a seconda della classe di dispositivo che viene utilizzato da 1 metro per la classe C (non più in produzione) a 10 metri per la classe B fino a 100 metri per la classe A.

Bluetooth utilizza la banda ISM di frequenze libere a 2.45GHz. Come sappiamo in questa banda potrebbero esserci molte interferenze con altre tecnologie senza fili (soprattutto WiFi canali 1 e 6 ma anche LTE canali 40 e 41). Per ridurre le interferenze il protocollo divide la banda in 79 oppure 40 canali (Bluetooth standard/smart vedi dopo) e commuta tra i canali fino a 1600 volte al secondo (frequency hopping) ottimizzando il rapporto segnale/rumore ed evitando la parte dello spettro con interferenze. Nel tempo sono state sviluppate varie versioni di Bluetooth con velocità e caratteristiche diverse. Vedi tabella:

Versione	Bitrate	Rilasciato	Rilasciato Novità introdotte
1.2	1 Mbit/s	2002	-
2.0 e 2.1	3 Mbit/s	2004/2007	EDR, GFSK, SSP
3.0 + HS	24 Mbit/s	2009	AMP, L2CAP-EM
4.0 +LE	24 Mbit/s – 100 Kbit/s	2010	Single e Dual mode

Tabella 1.4.2: Tabella comparativa versioni Bluetooth e novità

Alcune delle novità introdotte nella versione 2 e 2.1 sono EDR (Enhanced Data Rate) per aumentare la velocità di trasmissione, GFSK una modulazione con un duty cycle più corto per ridurre il consumo di energia. PSK (Phase Shift Keyng) per aumentare la sicurezza delle trasmissioni. SSP (Secure Simple

Pairing) un meccanismo per rendere il pairing (accoppiamento di due dispositivi) più facile e sicuro.

Alcune delle novità maggiori nella versione 3 sono AMP (Alternative MAC/PHY) che permette di sfruttare la radio 802.11 (WiFi) in sinergia con la radio Bluetooth per le trasmissioni di quantità più grosse di dati. In pratica l'acronimo Bluetooth 3 High Speed identifica i dispositivi che hanno entrambe le radio e possono usare la radio Bluetooth durante lo scambio di modeste quantità di dati. Nel momento nel quale è necessario scambiare delle grosse quantità di dati (Burst) viene utilizzata la radio del WiFi per lo scambio. In modo di diminuire i tempi di trasmissione e di conseguenza l'impatto energetico. Da notare che lo switch tra le radio viene eseguito in modo trasparente al utilizzatore in tempi molto ridotti (6ms). L'altra grossa novità è L2CAP-EM (Logical Link Control and Adaptation Protocol Enhanced modes) che migliora la ritrasmissione dei dati e implementa un meccanismo di ritrasmissione in caso di datagram persi (permette di avere connessioni garantite).

Nel 2010 con la standardizzazione della versione 4 di Bluetooth ci sono state moltissime novità. Nella nuova versione sia la radio che lo stack dei protocolli di comunicazione a livello fisico ma anche il modo di creare e gestire le applicazioni è cambiato. L'acronimo associato alla versione 4 è Low Energy. Ma le novità non sono limitate alla diminuzione del consumo di energia che è ridotto notevolmente.

Alcune delle novità sono:

La tecnologia utilizza una radio nuova, molto migliore Dal punto di vista energetico.

La banda non viene più suddivisa in 79 canali ma in 40 con alcuni dedicati solo a determinati tipi di pacchetti.

La lunghezza dei pacchetti (frame) aumenta.

Il modello di comunicazione viene cambiato. Anche se le connessioni sono sempre di tipo server/client sono possibili altri tipi di comunicazioni, tipo l'invio periodico di pacchetti in broadcast.

I dispositivi non sono sempre connessi, una volta accoppiati, la connessione viene attivata soltanto quando devono comunicare. Nei periodi di idle la connessione viene sospesa. La riattivazione delle connessioni sospese è estremamente veloce ed efficiente. Viene eseguita in 3ms.

Vista la diversa struttura dello stack e anche la radio completamente rivisitata, le versioni di Bluetooth da 4 in poi non sono retrocompatibili con le versioni precedenti. Per permettere un certo grado di compatibilità si è pensato di creare due tipi di dispositivi.

Il primo tipo viene chiamato DualMode e supporta sia le novità introdotte nel Bluetooth 4, ma è anche retrocompatibile con le versioni precedenti. Ovviamente questo tipo di dispositivo è dotato di due radio e di due stack Bluetooth. In questo modo è in grado di collegarsi sia a dispositivi con versioni precedenti di Bluetooth che dispositivi LowEnergy.

Il secondo tipo di dispositivi è chiamato SingleMode e non è retrocompatibile con le versioni precedenti del protocollo. Il vantaggio di questa tipologia di dispositivi è che hanno una struttura che permette di essere ancora più economici ed efficienti Dal punto di vista energetico.

Ricapitolando, i dispositivi Bluetooth Smart sono in grado di interagire soltanto con altri dispositivi con Bluetooth con versione della tecnologia pari o superiore a 4.

1.4.3 Reti Bluetooth

Anche per quanto riguarda le reti, le due tecnologie, Bluetooth Classic (BR) e Bluetooth LE (BLE) impongono una struttura differente per quanto riguarda le reti.

Nella versione Classic del protocollo è possibile creare due tipologie di reti. Piconet, cioè delle reti a stella con un master e fino a 7 slave. Il numero di slave è limitato a causa dello spazio di indirizzamento utilizzato nella rete a 3bit.

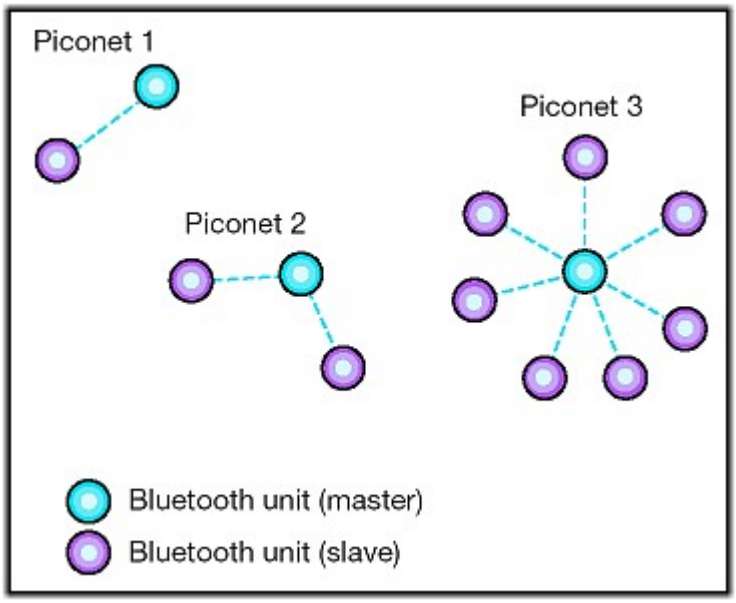

Figura 1.4.3.1: Struttura di una piconet

La seconda tipologia di rete che è possibile realizzare con Bluetooth Classic è scatternet, la quale è l'unione di due o più Piconet per formare delle reti più grandi. Per fare ciò, uno dei dispositivi agisce da slave in una delle piconet e da Master in una seconda Piconet. Le dimensioni delle scatternet sono limitate soltanto dal fatto che ad aumentare del numero di dispositivi collegati, il numero di collisioni crea problemi di trasmissione/ricezione rendendo impossibile la comunicazione oltre certi limiti che non sono specificate nelle specifiche del protocollo, ma dipendono dalla posizione dei dispositivi e della quantità di traffico nella rete.

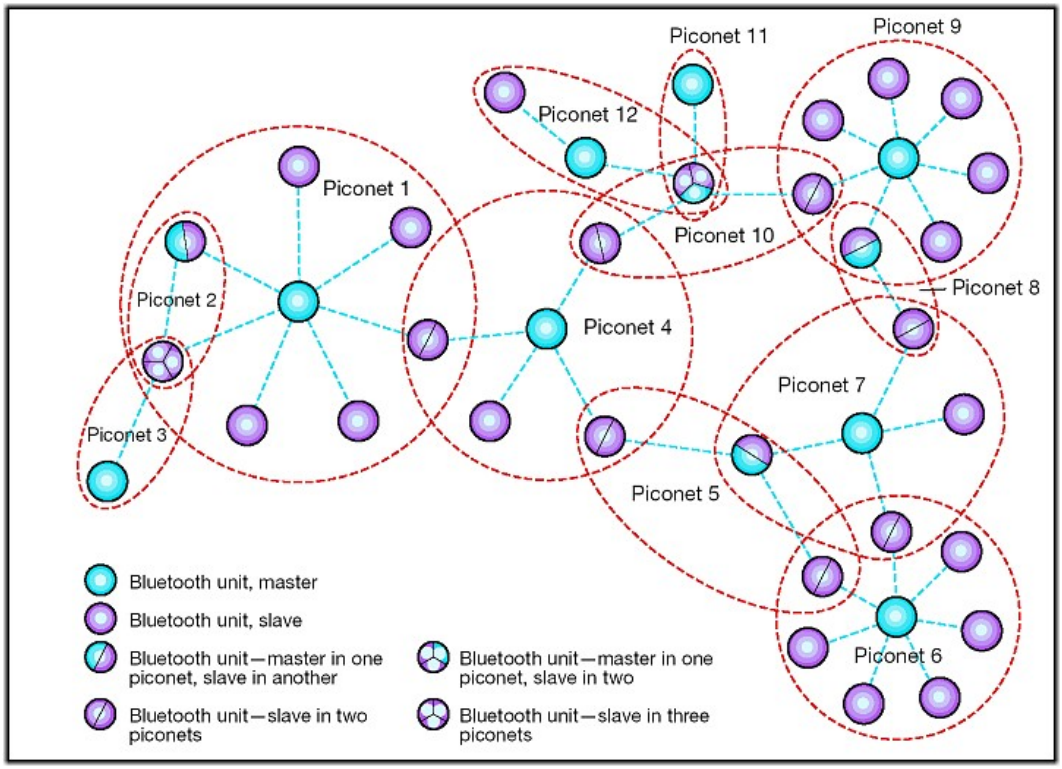

Figura 1.4.3.2: Struttura di una scatternet

Nella versione LowEnergy (Smart) il problema del numero di slave è stato superato grazie a un sistema di indirizzamento più ampio. Cioè, nella definizione del protocollo, non ci sono limitazioni al numero di dispositivi che possono essere collegati a un singolo master (il limite dipende dal implementazione del dispositivo e del software). Anche per quanto riguarda le interferenze le versioni Smart sono molto più robuste e soffrono meno dalle interferenze tra dispositivi, questo grazie soprattutto al modello di comunicazione scelto. Le connessioni non vengono più mantenute attive, ma vengono sospese finché non ci sono dati da trasmettere. Così se i dati non sono tanti, il problema delle collisioni anche in reti molto grandi è notevolmente ridotto.

Anche se il problema del numero di slave è stato risolto e teoricamente un solo dispositivo potrebbe fare da master in tutta la rete, è stata prevista la possibilità che i dispositivi si comportino sia da master che da slave contemporaneamente. Con questo accorgimento è possibile creare reti Bluetooth 4

che vanno oltre la distanza limitata di un singolo master. La topologia del modello di rete in Bluetooth 4 e chiamata Star-Bus.

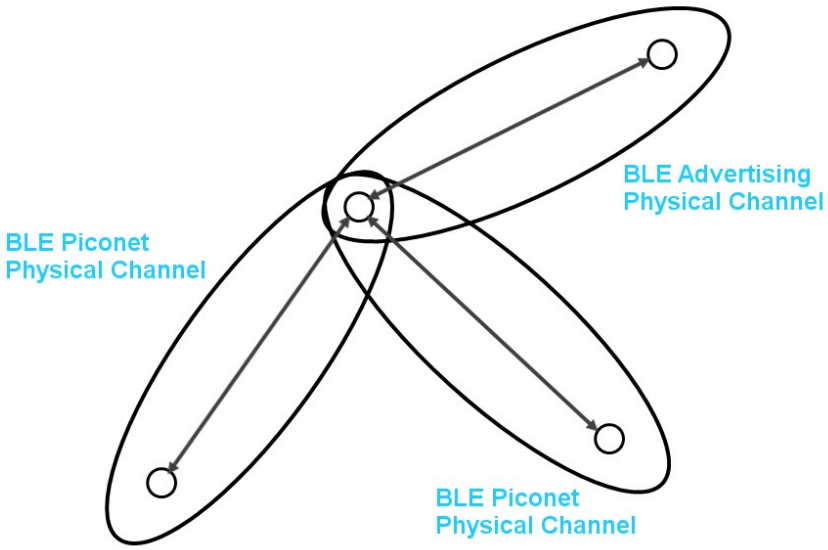

Figura 1.4.3: Struttura di una rete StarBus

1.4.4 Struttura dello stack Bluetooth e comparazione con TCP

Uno degli aspetti importanti per l'utilizzo di Bluetooth con TuCSoN come protocollo di trasmissione dei dati è legato alla sua capacità di collegare più dispositivi e permettergli di comunicare tra loro. Come abbiamo visto nel capitolo precedente, la tecnologia Bluetooth impone alcuni limiti tecnologici che dipendono dalla versione del protocollo e che sono molto diversi se comparati con la tipologia di rete TCP. Per questo motivo dovremo analizzare la tecnologia paragonandola con TCP per poter capire quali sono i passi da compiere per poter fare una integrazione.

A differenza di TCP/IP gli obiettivi del Bluetooth come detto prima sono strettamente legati alla semplicità, al costo (capacità computazionale), l'efficienza

energetica e alle dimensioni dei dispositivi. Queste esigenze hanno condotto alla progettazione di uno stack dei protocolli con una struttura molto più semplificata. Addirittura con accorgimenti che permettono ai produttori di dispositivi con Bluetooth di cambiare il fornitore delle radio senza dover intervenire a livello software.

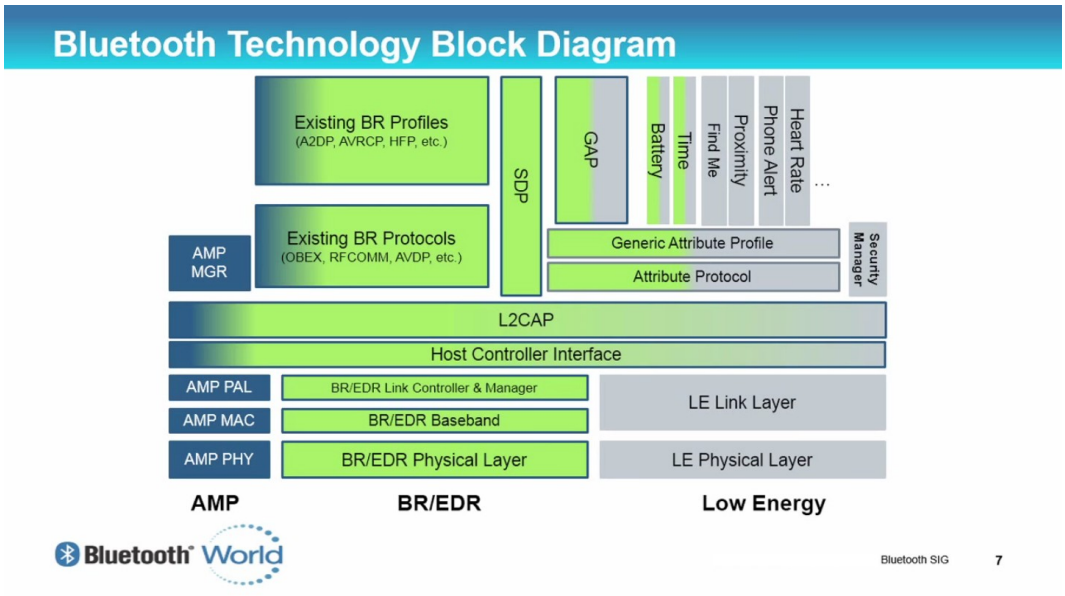

Figura 1.4.4: La struttura dello stack Bluetooth

Questo è un diagramma riassuntivo sia di Bluetooth BR che LE.

Il diagramma dei layer di Bluetooth è molto articolato. I layer sotto Host Controller Interface rappresentano il layer fisico e il data link di ISO/OSI. Da notare che tutti i layer sotto HCI, incluso HCI sono inclusi nel firmware del dispositivo Bluetooth. SIG ha creato standard molto severi per quanto riguarda l'architettura del HCI per poter permettere ai produttori di hardware di realizzare chip che sono compatibili tra loro. Cioè, se uno volesse sviluppare un dispositivo il quale utilizza Bluetooth, potrebbe utilizzare il chip che vuole senza preoccuparsi di compatibilità. HCI da un interfaccia generica standard per lo sviluppo di applicazioni BT.

Sopra HCI troviamo L2CAP (Logical Link Control and Application) il quale funge da multiplexer tra i vari protocolli e profili a livello più alto. Si occupa di dividere i dati in pacchetti di lunghezza giusta e di riassemblarli (non tutti i tipi di dati vengono trattati nello stesso modo), ha funzionalità di analisi per la qualità del servizio e gestione della integrità dei dati.

Sopra L2CAP sono definiti alcuni protocolli e profili. I protocolli sono funzionalità intrinseche necessarie per il funzionamento di Bluetooth e sono definite dalla SIG. Per esempio il SDP (Service Discovery Protocol) serve per poter chiedere al dispositivo al quale ci si collega cosa è in grado di fare (quali profili implementa). I profili sono semplicemente delle dichiarazioni su come devono essere implementati le applicazioni per garantire l'interoperabilità. Un dispositivo può implementare più profili. Quando avviene l'accoppiamento tra i dispositivi con il protocollo SDP ogni dispositivo comunicherà all'altro dispositivo quali sono i profili che implementa, cioè cosa è in grado di fare. Per esempio, quando accoppiamo uno smartphone ad un altoparlante esterno, l'altoparlante comunicherà che implementa il profilo A2DP (Advanced Audio Distribution Profile) e AVRCP (Audio/Video Remote Control Profile). Con il primo profilo (A2DP) si comunica non soltanto la capacità di riprodurre audio in alta definizione, ma anche il modo nel quale verranno trasmessi i dati (i dati realtime tipo musica e voce vengono trattati diversamente). Con il secondo profilo (AVRCP) si comunica che l'altoparlante è dotato di comandi per passare alla prossima canzone, mettere in pausa, controllare il volume ecc. Questo meccanismo permette di avere uno standard sia per quanto riguarda le funzionalità che per quanto riguarda il modo di inviare/interpretare i dati (Presentation Layer). A parte i profili standard, i quali sono specificati dalla SIG, Bluetooth sia nelle versioni BR che LE da la possibilità di implementazione di profili personali nei quali lo sviluppatore ha la libertà di decidere sia il loro identificativo che le funzionalità e il formato dei dati nella trasmissione. Ovviamente in questo caso, il profilo verrà compreso soltanto da dispositivi capaci di capire questo profilo, cioè dotati dalla applicazione che lo implementa.

L'ultima cosa estremamente importante da sapere su Bluetooth è che

nonostante sia uno stack di protocolli di comunicazione che ci offre tutto il necessario per la trasmissione di dati tra due applicazioni, non è previsto un meccanismo di routing, oppure di trasmissione di dati tra dispositivi che non siano direttamente collegati tra loro.

Cioè, se in una piconet due slave volessero scambiarsi dati non possono farlo utilizzando soltanto lo stack Bluetooth. Per farlo devono inviare i dati al master il quale successivamente dovrà reinviarli al secondo slave. In TCP/IP questa operazione di reinvio dei dati per raggiungere il destinatario è gestita direttamente dal IP e fa parte del protocollo, ma non essendo prevista in Bluetooth, il meccanismo di ritrasmissione deve essere implementato a livello applicativo. Nelle reti di tipo scatternet la complessità per implementare un tale meccanismo, soprattutto se i due nodi che vorrebbero comunicare appartengono a due sotto piconet che non hanno nodi in comune aumenta notevolmente.

La decisione di non implementare un algoritmo di routing nel protocollo da parte di SIG è dettata dal fatto che la tecnologia è progettata per dispositivi con scarse capacità computazionali, ma come si è visto negli ultimi anni il suo utilizzo è cambiato tanto. Nel novembre del 2014 SIG ha creato un gruppo di ricerca per progettare e introdurre un meccanismo di routing nelle future versioni.

Parte II

Studio di fattibilità

2 Analisi delle tecnologie

2.1 Analisi implementazione attuale in TCP/IP

Come detto prima, attualmente le comunicazioni in TuCSoN sono implementate esclusivamente con TCP/IP. Per accedere alla rete TuCSoN utilizza le API Java (java.net.Socket) le quali sono usate nelle classi che implementano il protocollo di comunicazione di TuCSoN. Il package nel quale sono incluse le classi è alice.tucson.network. In particolare la classe che implementa la comunicazione è TucsonProtocolTCP.java che a sua volta implementa la classe astratta AbstractTucsonProtocol, utilizzando alcune classi di supporto. La classe ha due costruttori, uno per creare un Socket a lato server (un socket in ascolto su una porta) il secondo costruttore crea un socket già collegato al nodo remoto con il socket in ascolto (chiede come parametri l'indirizzo IP e la porta TCP alla quale collegare il socket). Per eseguire le operazioni di trasmissione e ricezione vengono utilizzati i metodi receiveMsg e sendMsg i quali inviano o ricevano un oggetto di tipo TucsonMsg. In pratica tutta la gestione della rete si affida ai Socket di Java a parte la parte di Presentation layer cioè della rappresentazione dei dati applicativi durante la trasmissione. Come abbiamo detto precedentemente il meccanismo di traduzione dei dati, cioè il layer di presentazione va implementato a livello applicativo. Nonostante da entrambi i lati della comunicazione abbiamo nodi TuCSoN che parlano Java, non è detto che i dati vengono rappresentati nello stesso modo visto che ci possono essere delle differenze architetturali. Ad esempio su architetture e sistemi operativi differenti, la rappresentazione dei dati potrebbe cambiare, ma anche su diverse implementazioni della VJM non abbiamo garanzia che i dati siano strutturati nello stesso modo. Per ovviare a questo problema i messaggi di tipo TucsonMsg implementano l'interfaccia della classe classe java.io.Serializable. Il meccanismo di Serializzazione offerto nativamente da Java permette di stabilire un modo univoco per la rappresentazione dei dati durante il loro trasferimento. Questo è un meccanismo che da la possibilità di serializzare una classe intera, cioè di renderla comprensibile da qualsiasi JVM (Java Virtual

Machine) semplicemente implementando la classe astratta java.io.Serializable nelle proprie classi che devono essere trasmesse in rete. Ricevendo uno stream di byte di un oggetto serializzabile, sapendo a priori il suo tipo, possiamo fare il cast, senza preoccuparsi delle varie rappresentazioni locali della macchina virtuale che ci ha inviato la classe.

2.2 Bluetooth e API Java

Nella fase di studio di fattibilità, si è venuto a conoscenza della situazione delle API Java per Bluetooth, la quale è un po' più complessa rispetto a quanto si pensava inizialmente. Con JSR-82 (Java Specification Request – 82) SIG ha definito uno standard su come devono essere fatte le API Java, ma purtroppo in Java SE la specifica JSR-82 non è mai stata implementata nativamente. È disponibile soltanto la implementazione delle API per J2ME (Java platform Micro Edition).

J2ME è una versione del linguaggio creata per dispositivi mobili e incorporati (embedded) tipo telefoni cellulari o lettori dvd e simili. E caratterizzata da un insieme di API molto più piccolo visto che le applicazioni create sono eseguite su dispositivi con risorse molto limitate. TuCSoN invece come detto prima utilizza Java 7 SE (anche se alcune sue librerie sono scritte in Java 8). Essendo la versione ME molto restrittiva non può essere utilizzata per la compilazione di TuCSoN.

La specifica JSR-82 non è stata implementata nelle API Java SE visto che una parte dello stack Bluetooth fa parte del sistema operativo e su diversi sistemi operativi è implementata diversamente.

Non è stata implementata una API standard in Java SE visto che le diverse JVM per ogni piattaforma avrebbero dovuto essere aggiornate troppo frequentemente per poter stare al passo con la rapida evoluzione del protocollo e con i frequenti aggiornamenti dei vari stack inclusi nei vari sistemi operativi. Per far capire meglio la difficoltà nel stare aggiornati con i vari stack Bluetooth inclusi nei sistemi operativi possiamo prendere come esempio Windows prima del

Windows 8. Non c'era uno stack unico, ma ogni dispositivo importava uno stack di terze parti in Windows per garantire il proprio funzionamento. Ci sono ben 6 diversi stack (Widcomm, Microsoft Windows Stack, CSR Harmony, Bluetooth Toshiba Stack, BlueSoleil, BlueFritz!). In Linux ci sono due Bluetooth stack BlueZ e BlueDroid (nato come fork per Android), ma vengono aggiornati molto frequentemente.

Java è un linguaggio che cerca di avere meno dipendenze dal architettura possibile e vista la situazione molto complessa a lato stack Bluetooth integrato nei diversi sistemi operativi, garantire che il codice scritto sia compatibile con diverse piattaforme è estremamente difficile.

Per colmare questo gap, sono nati vari progetti proprietari e open source con lo scopo di creare delle librerie da includere in JavaSE che permettono l'utilizzo delle API JSR-82 in modo trasparente allo stack del sistema operativo sottostante. Questo è il diagramma a blocchi di uno dei più riusciti di questi progetti (BlueCove):

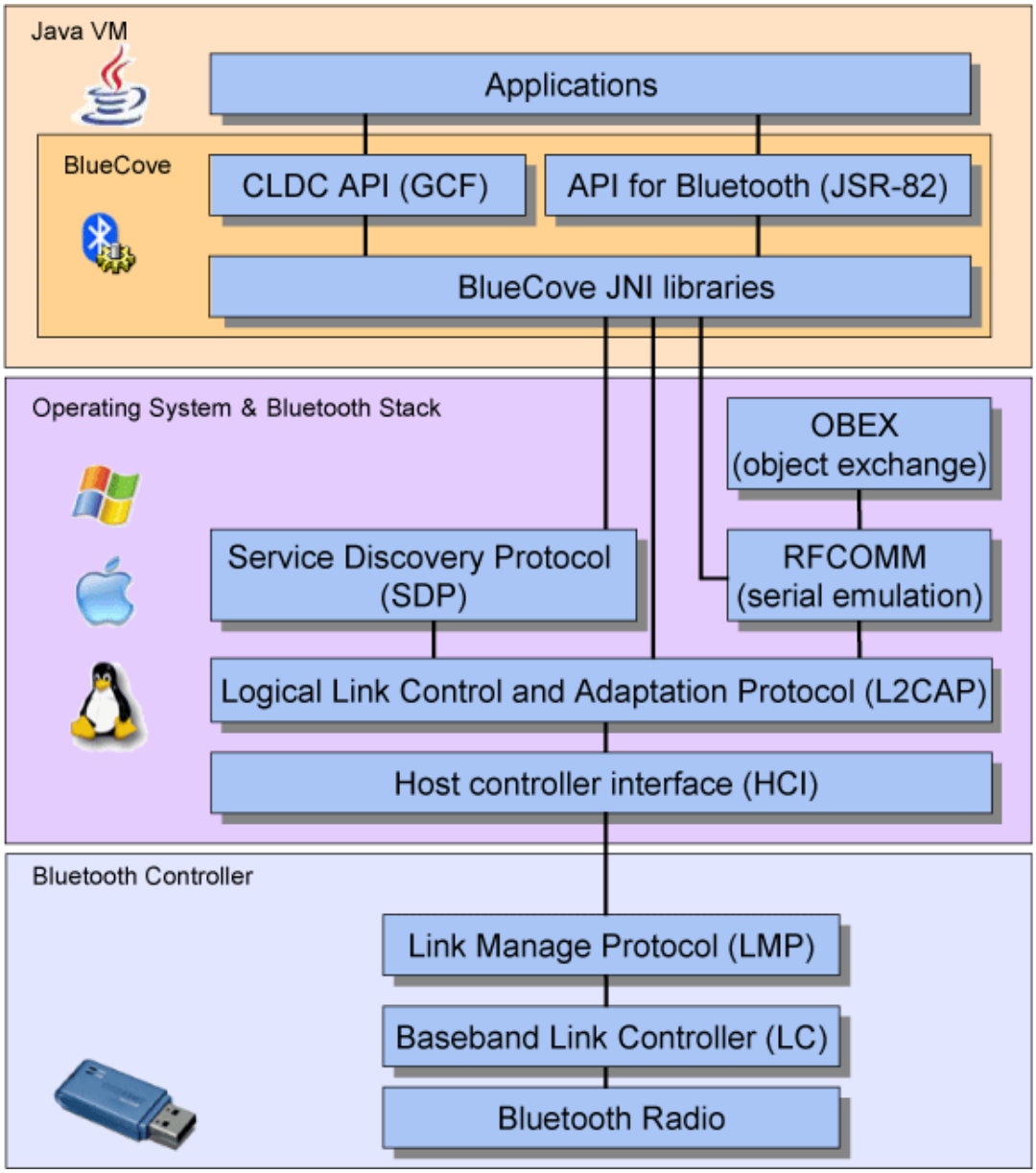

Figura 2.2: Schema a blocchi delle librerie BlueCove

BlueCove non è l'unico progetto che ha adottato questo approccio, ma è quello che ha avuto più successo nelle sue implementazioni.

Purtroppo nonostante lo slancio iniziale di questi progetti, quasi tutti hanno avuto una vita molto breve. La maggioranza ha interrotto lo sviluppo nel 2007/2008 e le librerie fatte, anche se funzionanti quando sviluppate, sono ormai

obsolete. Elencherò alcuni dei progetti di migliore successo e le loro caratteristiche per maggior chiarezza:

2.2.1 BlueCove

E' un progetto Open source con licenza non molto chiara. Nel sito ufficiale dichiara una licenza Apache 2.0, ma nella pagina del progetto su Google Code la licenza dichiarata è GPL. Comunque le indicazioni nelle FAQ specificano che è permesso di utilizzare le librerie anche in applicazioni non open source senza dover pubblicare il codice. (se non si modificano le librerie)

La documentazione del progetto non è ben curata, anche se teoricamente si può affidare alla documentazione delle API JSR-82 per lo sviluppo di un'applicazione.

Lo sviluppo del progetto è stato interrotto nel 2008. L'ultimo aggiornamento delle librerie è del 26 dicembre 2008.

Le librerie dichiarano di funzionare su:

- Mac OS X 10.4 e 10.5 (PowerPC e Intel)
- Windows XP x86 SP2 (WIDCOMM, BlueSoleil, Microsoft Bluetooth Stack)
- Windows Vista x86 (WIDCOMM, BlueSoleil, Microsoft Bluetooth Stack)
- Linux i386 e 64bit con BlueZ 3.9 (con Fedora 7 e Ubuntu 7.4)

Sul sito del progetto ci sono vari esempi di codice già fatto che dovrebbe funzionare, ma dai test eseguiti con Ubuntu 14.04LTS 64bit (Linux 3.13.0-76 con BlueZ 5.35) gli esempi non riuscivano ad individuare il dispositivo Bluetooth presente e funzionante del mio portatile. Anche con l'installazione di versioni precedenti di BlueZ il problema non è stato risolto.

2.2.2 Avetana

Avetana è un progetto proprietario, ma è presente anche una versione gratuita per 14 giorni. È sviluppato da una software house tedesca, ma non viene

più aggiornato dal 2007. La documentazione del progetto è quasi inesistente ma si rimanda soltanto alla documentazione delle API JSR-82. Nelle FAQ del sito dichiarano questa compatibilità:

- Windows 98 SE, ME, 2000, XP, XP SP2 con stack WIDCOMM 1.4.2.10 SP5 oppure Bluetooth Microsoft Stack del XP SP2
- Mac OS X 10.3
- Per quanto riguarda Linux non è chiaro se è compatibile o no.

Il sito del progetto è abbandonato da molto tempo. Le informazioni reperibili sono veramente poche. Il download della versione di prova per 14 giorni va richiesto via email. Ho richiesto la versione di prova con i recapiti presenti sul sito due volte, ma non ho ricevuto nessuna risposta. In vari forum ho trovato commenti su svariati problemi con queste librerie, ma non ho potuto testarle di persona.

2.2.3 Rocco

Le librerie di Rocco non sono più disponibili sul loro sito. La software house adesso si occupa di Beacons e creazione di software nel ambito di proximity sensoring. Su siti esterni è tuttora possibile trovare alcune versioni delle librerie, ma comunque sono affette dagli stessi problemi delle librerie precedenti, cioè licenza proprietaria, sviluppo interrotto da molto tempo (da quanto emerge nel 2006), l'impossibilità di trovare documentazione e nessuna garanzia su compatibilità.

Purtroppo tutte le librerie di terze parti che implementano le API JSR-82 per Java SE non sono più in sviluppo e le versioni precedentemente sviluppate non funzionano con gli stack Bluetooth aggiornati inclusi nei sistemi operativi di oggi.

2.3 Bluetooth e Android

Visto che TuCSoN è compatibile anche con Android, sono state analizzare anche le API offerte direttamente da Android per quanto riguarda la tecnologia Bluetooth. In Android ci sono due gruppi di API, un gruppo per quanto riguarda il Classic Bluetooth (BR) e un altro per Bluetooth Low Energy.

Le API per la versione classica sono disponibili da API level 5, cioè Android 2.0 Eclair. Sono molto ben documentate e sul sito dedicato agli sviluppatori Android si trovano numerosi esempi di codice per per testare velocemente le loro funzionalità. Le operazioni permesse da questo gruppo di API sono:

- Controllare se Bluetooth è presente sul dispositivo
- Abilitare il Bluetooth ed eventualmente la visibilità
- Cercare dispositivi Bluetooth visibili nelle vicinanze
- Controllare se nelle vicinanze ci sono dispositivi non visibili ma precedentemente accoppiati (dispositivi che conosciamo già)
- Collegarci agli dispositivi scoperti, con il servizio Service Discovery (quando lanciamo la richiesta di connessione come risposta il dispositivo ci dice che tipo di profili implementa, cioè quello che è in grado di fare e come dovremo strutturare i dati per comunicare correttamente)
- Stabilire canali di comunicazione RFCOMM (l'equivalente di una porta seriale)
- Trasferire dati da e verso altri dispositivi Bluetooth
- Gestire più dispositivi connessi.
- Definire propri profili per le applicazioni che creiamo.

Il funzionamento dello scambio di dati è molto simile a quello offerto dalle Socket TCP connesso di Java SE con alcuni accorgimenti nel modo di leggere e scrivere i dati nel socket, dovuti alla piattaforma.

Il secondo gruppo di API riguarda il LowEnergy. L'idea dieto al BTLE è

che i dispositivi si scambino piccole quantità di dati con intervalli di tempo nel quale la connessione viene sospesa. Il modello di comunicazione cambia. Il collegamento tra i dispositivi viene gestito con un gestore di profili GATT nel sistema operativo. La creazione di profili specifici con lo standard Bluetooth Low Energy avviene a partire dal profilo generico GATT (Generic Attribute Profile), che utilizza solo il protocollo ATT (Attribute Protocol) in aggiunta ai protocolli inferiori. Il protocollo ATT definisce due ruoli: server e client. Il server è in grado di contenere dei dati (attributi) che sono accessibili dal client tramite il protocollo. L'applicazione sottoscrive l'interesse di ricevere gli aggiornamenti di un determinato dispositivo di un determinato tipo fornendo una procedura di callback che verrà lanciata ogni volta che viene ricevuto un aggiornamento che corrisponde a quello che abbiamo sottoscritto. Così l'ottimizzazione energetica viene gestita dal sistema operativo. Le API BTLE fanno parte dal level 18 cioè Android Jelly Bean.

La versione attuale di TuCSoN è stata implementata con Android 4.4 KitKat (API level 19) e entrambi gruppi di API, sia per il collegamento classico che quello Low Energy possono essere utilizzate.

3 Integrazione TuCSoN e Bluetooth

Dopo aver introdotto le tecnologie che riguardano questa tesi e aver analizzato lo stato attuale di tali tecnologie, verranno illustrati i risultati della analisi svolta per il conseguimento del obbiettivo di estendere TuCSoN con la possibilità di comunicare sia via Bluetooth che via TCP.

L'idea iniziale era di implementare tale comunicazione generalizzando le classi già presenti in tucson.network e rendendo trasparente la comunicazione indipendentemente dalla tecnologia con la quale il nodo è connesso alla rete. Purtroppo analizzando lo stato della tecnologia è emerso che fare l'integrazione come si è pensato al inizio non sarebbe possibile. I problemi emersi dalla analisi sono i seguenti.

3.1 Problema di indirizzamento e Naming

Come detto prima in TuCSoN ci sono delle regole prestabilite per il naming. In particolare gli identificativi dei nodi e dei centri di tuple sono strettamente dipendenti dal indirizzamento TCP/IP. Come sappiamo già ogni nodo è identificato in questo modo:

netid : portno

Identificato il nodo, il centro di tuple su questo nodo viene identificato in questo modo

tname @ netid : portno

Essendo netid e portno rispettivamente l'indirizzo IP e il numero di porta TCP/IP, non sono semplicemente parte del nome, ma sono gli elementi necessari per individuare il nodo nella rete TCP/IP. Se la comunicazione venisse implementata con una tecnologia diversa da TCP/IP, questi elementi non identificherebbero il nodo nella rete.

In particolare Bluetooth sia nella sua versione classica che nella versione LE utilizza indirizzamento diverso. Gli indirizzi in una piconet (Bluetooth Classico) sono sempre assegnati dal master (il dispositivo che richiede la connessione) anche se il ruolo di master potrebbe essere passato al dispositivo al quale si richiede la connessione. Gli indirizzi nella piconet sono a 3 bit (massimo 7 slave). Gli indirizzi in una scatternet sono assegnati dai due master ed essendo solo a 3 bit, nelle due piconet sono presenti nodi con lo stesso indirizzo. Esistono anche gli indirizzi fisici dei dispositivi i quali sono a 48 bit, ma vengono utilizzati soltanto per etichettare i pacchetti nei collegamenti punto a punto.

L'indirizzamento in BTLE è molto più ampio, ma comunque

l'indirizzamento non è accessibile a livello applicativo. In generale sia in BTBR che in BTLE a livello applicativo non viene usato l'indirizzo per distinguere i dispositivi nella rete, ma il Bluetooth socket nel caso di BTBR oppure il GATT nel caso di BTLE e a priori non possiamo conoscerli. E i dispositivi devono essere connessi direttamente in assenza di un meccanismo di routing (vedi sotto).

Se si volesse estendere il layer di comunicazione di TuCSoN per poter usare Bluetooth, ma anche qualsiasi altro protocollo di comunicazione diverso da TCP/IP il modo in cui viene definito il naming (gli identificativi) dovrebbe cambiare, oppure a livello applicativo bisognerebbe introdurre un meccanismo che traduca gli indirizzi IP in indirizzi di diverso tipo per poter eseguire il collegamento. Ovviamente, questo approccio permetterebbe soltanto l'identificazione dei nodi a livello applicativo perdendo tutta la semantica del indirizzo IP visto che la rete è di una tipologia diversa.

3.2 Topologia e Routing nelle reti Bluetooth

Come visto prima le topologie delle reti in Bluetooth sono differenti rispetto alle topologie utilizzate dalla TCP/IP utilizzata da TuCSoN. In Bluetooth le topologie di reti definite nel protocollo sono le piconet e le scatternet per BTBR e in BTLE abbiamo una tipologia di rete chiamata star-bus (centro stella su mezzo trasmissivo comune). Ma a parte queste topologie è possibile organizzare le proprie reti anche con topologie differenti a seconda delle proprie necessità. Essendo Bluetooth una tecnologia wireless è possibile creare topologie ibride (ad-hoc) con diverse strutture.

Alcuni esempi

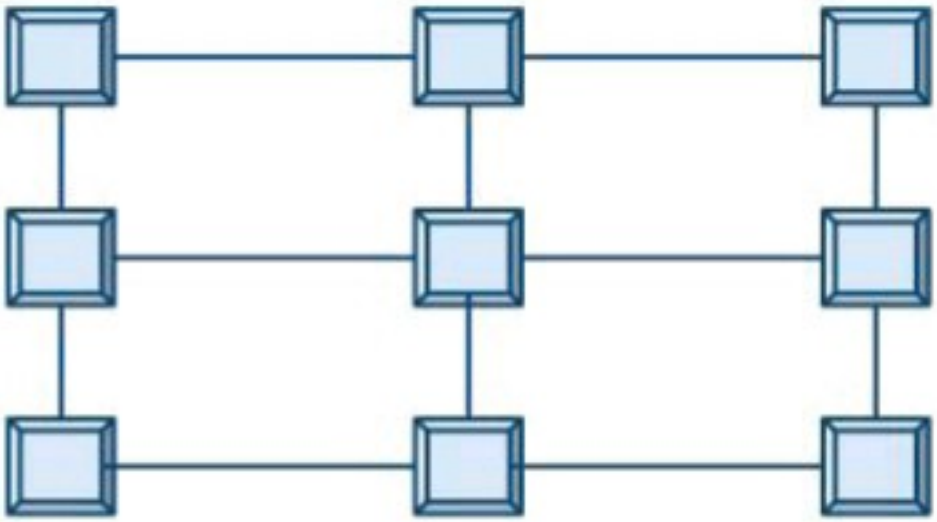

Figura 3.2.1: Topologia Mesh

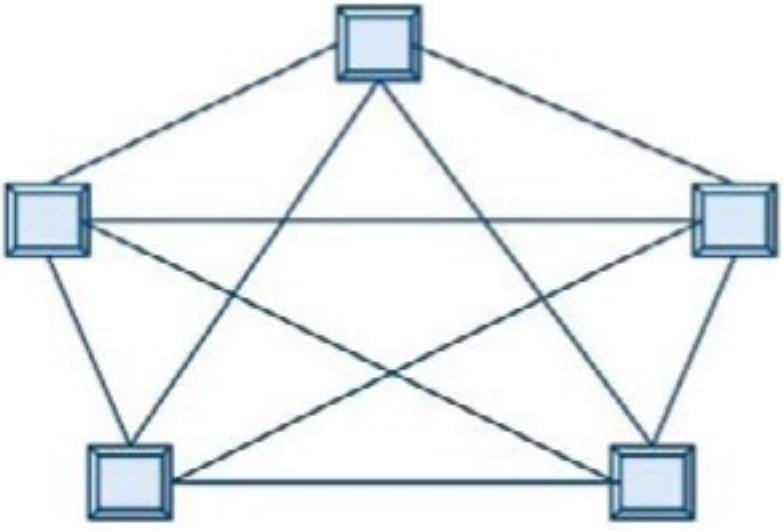

Figura 3.2.1: Topologia Full connected

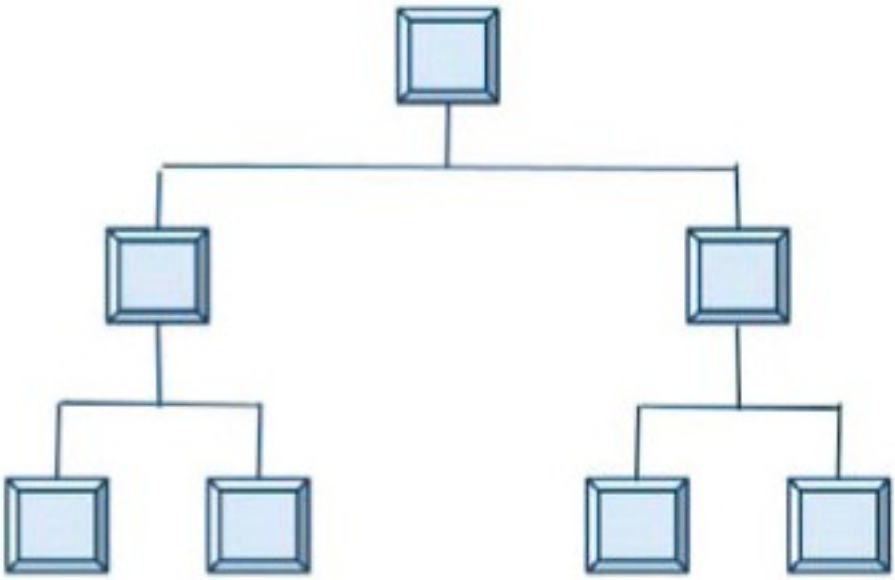

Figura 3.2.1: Topologia Tree (ad albero)

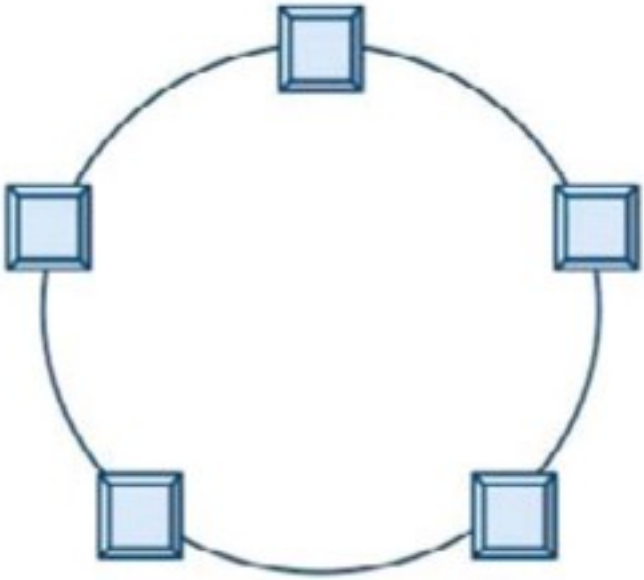

Figura 3.2.1: Topologia Ring (ad anello)

Le topologie differenti richiedono anche meccanismi di instradamento (routing) differenti. Negli standard Bluetooth non viene definito alcun meccanismo di routing. Cioè, unendo più piconet in una scatternet, Bluetooth non ci da nessun meccanismo per inviare i dati da un slave/master di una piconet a uno che appartiene a un'altra piconet. Ma anche nella stessa piconet, due slave non possono scambiarsi i dati direttamente. Dovrebbero inviare i dati al master il quale poi dovrebbe sapere a chi ritrasmetterli. Ma visto che questo meccanismo non è definito dentro lo stack tutta la parte di instradamento dovrebbe essere gestita a livello applicativo. Visto che è possibile organizzare la topologia delle proprie reti anche in configurazioni differenti, la complessità per garantire l'instradamento è ancora maggiore. Gli algoritmi di routing sono sviluppati per funzionare efficientemente su una determinata topologia e non sono universali.

La mancanza di un meccanismo di routing in Bluetooth limita fortemente l'utilizzo della tecnologia nel campo del Internet delle cose, la Bluetooth SIG ha creato un gruppo che si focalizzi sullo studio di un estensione dello standard per rendere possibile la creazione di reti mesh con Bluetooth. Il gruppo ha iniziato i lavori nel novembre del 2014, ma il gruppo non ha ancora pubblicato niente.
Nel frattempo sono nati diversi progetti che tentano di implementare in modo diverso una rete ad-hoc di tipo Mesh utilizzando Bluetooth e WiFi insieme. La maggior parte dei progetti è nata dal esigenza di comunicare durante manifestazioni di protesta dove i partecipanti si dovevano coordinare senza la possibilità di utilizzare la rete cellulare. Quasi tutti i progetti nati con questo scopo però non hanno licenze proprietarie e codice non disponibile.

Un interessante progetto che si distingue è Underdark P2P. È un progettato sviluppato da una piccola società polacca (Estimote).

Le librerie hanno una licenza non ancora ben definita, ma sembra che l'intenzione sia quella di renderle GPL. Sono state rilasciate 21 gennaio 2016 e non sono ancora completamente definite. Gli obiettivi che si sono posti è di creare reti Mesh che utilizzano collegamenti ad-hoc Bluetooth e WiFi per creare reti mesh nei quali trasmettere messaggi di testo e immagini tra i nodi. Parte dal codice è pubblicata su GitHub, anche se la parte principale del progetto è

incapsulata in una libreria compilata e non è possibile visionare/modificare il sorgente.

Un modo semplice ma molto inefficiente per permettere la comunicazione tra nodi distanti in una rete Bluetooth è il flooding. Che consiste nella ritrasmissione dei messaggi verso tutti i nodi finché il messaggio non ha raggiunto anche il destinatario. Ovviamente dovrebbero essere implementati anche meccanismi per limitare le ritrasmissioni ridondanti, visto che l'eccesso di traffico causa problemi di interferenze nella banda Bluetooth.

3.3 Disponibilità delle API Android su architettura x86

Il terzo problema riguarda l'ambiente di esecuzione dei nodi TuCSoN. Come detto prima, TuCSoN prevede la possibilità di essere eseguito in ambiente Android come una estensione e non come ambiente principale. Il suo ambiente “naturale” è x86.

È stata analizzata anche la possibilità di utilizzare Android sotto l'architettura x86, questo è necessario visto che le API per Bluetooth esistono soltanto in ambito Android.

Nonostante è basato su Kernel Linux, Android è nato per il mobile e la sua dipendenza dal hardware nella sua implementazione concreta è molto forte. Google non ha sviluppato nessuna distribuzione x86 generica non dipendente dal hardware. Esiste comunque un progetto Open Source il quale si pone come obiettivo proprio la creazione di una distribuzione per x86. Purtroppo è un progetto molto piccolo il quale sta facendo molta fatica a mantenere il suo supporto aggiornato visto che la velocità di sviluppo di Android è notevole. Durante la stesura di questa tesi, ad ottobre 2015 il progetto sta testando una R3 (terza versione stabile) di Android 4.4 KitKat, ma visti i commenti sul loro forum, la stabilità è parziale. Nello stesso mese Google ha rilasciato la versione 6.0 Marshmallow del suo sistema operativo. Purtroppo nonostante si dichiara x86

compatibile la compatibilità non è ancora raggiunta. I test da parte loro sono stati eseguiti su:

- ASUS Eee Pcs/Laptops
- Viewsonic Viewpad 10
- Dell Inspiron Mini Duo
- Samsung Q1U , Viliv S5
- Lenovo ThinkPad x61 Tablet

Ma hanno evidenziato, da modello a modello, diversi problemi di compatibilità hardware. Sopratutto per quanto riguarda la connettività.

Parte III

Conclusioni e futuri sviluppi

4 Conclusioni

Visti i vari limiti delle tecnologie attualmente disponibili emersi dall'analisi ci si è resi conto che non è possibile estendere layer di comunicazione di TuCSoN cercando di implementare tutte le comunicazione in modo trasparente e indipendente dalla tecnologia utilizzata per il collegamento alla/e reti.

Visto che questa estensione serviva per permettere a TuCSoN ad interagire con dispositivi dotati di protocolli di comunicazione differenti si è allora pensato ad un approccio differente per ottenere questa possibilità con un approccio alternativo.

L'approccio differente consiste nella creazione di un insieme di librerie le quali possono essere utilizzate dagli agenti TuCSoN per permettere/agevolare la comunicazione con questi dispositivi dotati di connettività differente. Lasciando la struttura della comunicazione del middleware TuCSoN invariata ci permette di limitare la complessità delle tecnologie di comunicazione differenti e di incapsularla in uno o più agenti che avranno questo compito.

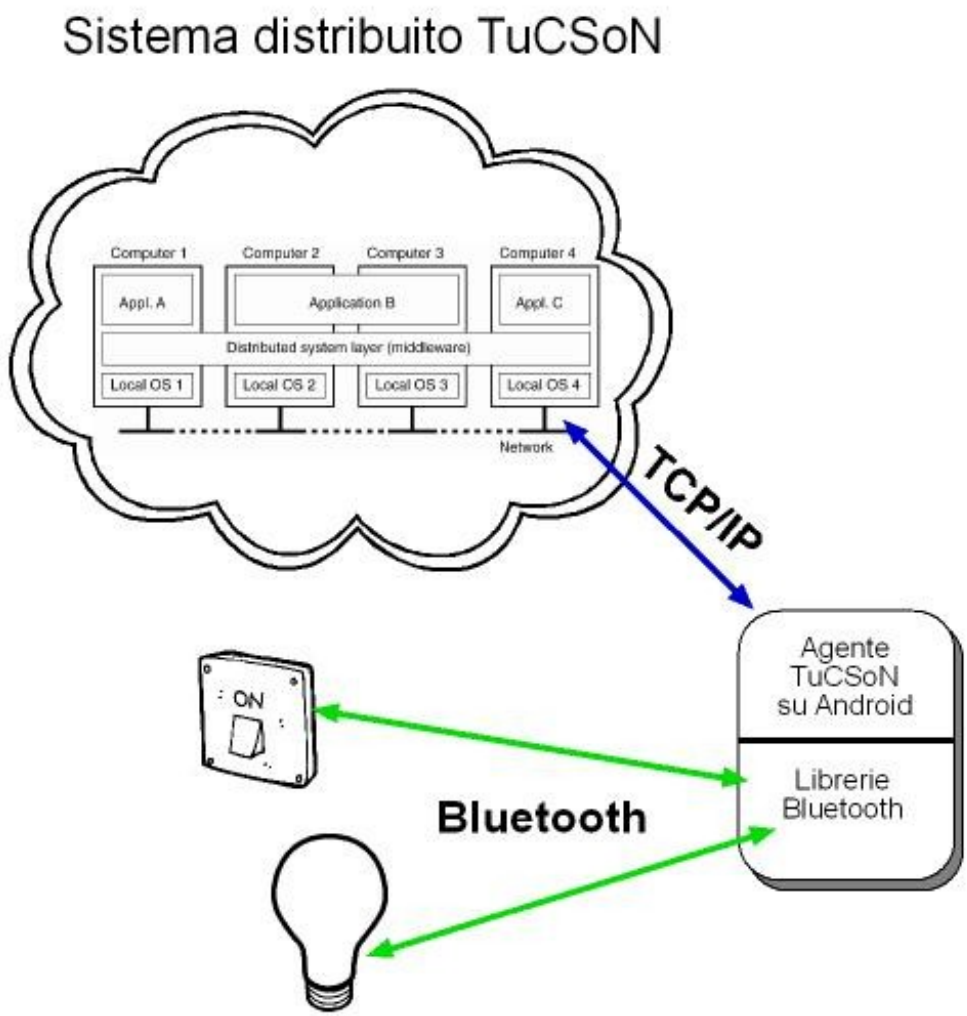

Figura 4: Schema riassuntivo

Come si vede dallo schema, alcuni agenti faranno da ponte tra il sistema distribuito utilizzando TCP/IP, interfacciando di fatto le entità che devono essere

messi in collegamento con il sistema distribuito con delle tecnologie differenti per quanto riguarda la loro connettività.

Questo approccio generale ci permette di creare agenti in grado di interfacciare TuCSoN con diverse piattaforme le quali hanno la possibilità di interfacciarsi direttamente al mondo IoT. Per esempio, su Android possiamo sfruttare le api native per Bluetooth, oppure su sistemi tipo RasperryPi possiamo sfruttare le GPIO per interagire con il mondo dei microcontrollori.

A scopo dimostrativo e per implementare il primo tassello di queste librerie è stata implementata una applicazione per Android la quale implementa un agente TuCSoN il quale sfruttando la connettività Bluetooth del dispositivo è in grado di ricevere/inviare messaggi via Bluetooth.

5 Sviluppi futuri e approfondimenti

Durante i lavori su le librerie Bluetooth sono state individuate alcune librerie che potrebbero essere incluse in futuro per estendere ulteriormente le tecnologie con le quali TuCSoN potrà interagire. Inoltre vista la mancanza di una implementazione di un protocollo di routing su Bluetooth, sono stati analizzati alcuni algoritmi di routing su reti ad-hoc.

In seguito elencherò le librerie che ho ritenuto di maggior lrilievo per il futuro sviluppo di TuCSoN e una panoramica sugli algoritmi di routing.

5.1 Raspberry Pi

Rasperry Pi è un mini computer con sistema operativo Linux dotato di porte GPIO. Si trovano diverse librerie in Java che implementano oltre la possibilità di utilizzare i pin GPIO anche di utilizzare alcuni protocolli di comunicazione che possono essere sfruttati da un agente TuCSoN per interfacciarsi con diversi dispositivi hardware.

Il progetto di maggior successo si chiama Pi4J e implementa oltre la possibilità di manipolare i pin GPIO, i seguenti protocolli:

- I2C – Inter Integrated Circuid – è un bus seriale che permette la comunicazione tra circuiti integrati.
- SPI – Serial Peripheral Interface – è un bus sincrono di comunicazione seriale
- UART - Universal Asynchronous Receiver-Transmitter

A parte i protocolli già implementati grazie ai GPIO è possibile interagire con piattaforme di sviluppo tipo Arduino per poter integrare tutte le tecnologie disponibili da sulla piattaforma Arduino.

5.2 Routing Ad-Hoc

In letteratura questo tipo di reti è chiamato MANET (Mobile Ad-hoc NETwork) e si trovano vari algoritmi sviluppati che permettono l'implementazione dell'instradamento, ma sono stati sviluppati per tecnologie wireless (802.11) e il loro utilizzo su Bluetooth non è mai stato applicato.
Gli algoritmi di routing dovrebbero:

- assicurarsi che le tabelle di routing siano ragionevolmente piccole, anche alla luce delle risorse ridotte delle quali spesso dispongono i nodi in una rete ad hoc
- riuscire a scegliere il miglior percorso per raggiungere gli altri nodi (in base a vari parametri, come la velocità, l'affidabilità e l'assenza di congestione),
- Tenere le proprie tabelle di routing aggiornate nel caso la topologia di rete cambi,
- Raggiungere il funzionamento ottimale in poco tempo e inviando un numero esiguo di pacchetti,
- Eventualmente offrire path multipli per raggiungere una destinazione, magari ordinando i path in ordine crescente di costo.

I protocolli possono essere generalmente classificati nelle seguenti categorie:

- Proattivi – alcuni esempi di questo tipo sono DSDV e OLSR
- Reattivi – alcuni esempi di questo tipo sono AODV e DSR
- Ibridi – alcuni esempi di questo tipo sono ZRP

- Gerarchici – alcuni esempi di questo tipo sono DART, ART e HSR

5.3 Layer di presentazione

Nei package di TuCSoN (alice.tucson.persistency) è presente un meccanismo per la persistenza di un centro di tuple in un fire XML. Con le oportune modifiche lo stesso codice può essere utilizzato per convertire gli attuali messaggi serializzati in messaggi XML per poter comunicare con piattaforme con conettività TCP/IP, ma con linguaggi di programmazione diversi da Java. Ad esempio, si potrebbe comunicare direttamente con Arduino (con shield ethernet).

In alternativa è possibile sfruttare le librerie del progetto ArduLink il quale permette il controllo completo di diversi modelli di Arduino.

Biblografia

TuCSoN:

- http://apice.unibo.it/xwiki/bin/view/TuCSoN/

Bluetooth:

- https://it.wikipedia.org/wiki/Bluetooth
- http://www.oracle.com/technetwork/articles/javame/index-156193.html
- http://www.giuseppesicari.it/articoli/java-micro-edition-bluetooth-api/
- https://en.wikipedia.org/wiki/Java_APIs_for_Bluetooth
- http://sisinflab.poliba.it/scioscia/resources/Service_Discovery_in_API_JSR-82.pdf

Librerie JavaSE per Bluetooth

- http://www.bluecove.org/
- http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml

Bluetooth per Android:

- http://developer.android.com/guide/topics/connectivity/bluetooth.html

TCP/IP

- http://dm.unife.it/~salati/cesena/annunci.htm#Testi

ReSpecT

- https://apice.unibo.it/xwiki/bin/view/ReSpecT/CompletedTheses

Topologie Ad-Hoc e routing

- https://it.wikipedia.org/wiki/Mobile_ad-hoc_network

RaspberryPi and Java:

- www.pi4j.com
- http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/RaspberryPi_GPIO/RaspberryPi_GPIO.html