



Dipartimento di Informatica
Corso di laurea in Informatica

Studio apparati Bluetooth/Wi-Fi di controllo termico e realizzazione di sw di gestione

RELATORE: Dott. Andrea Trentini

TESI DI LAUREA DI:

Sergio Alberti

Mat. 811070

Anno Accademico 2015/2016

*Ringrazio i miei genitori e i miei parenti
per avermi pazientemente spronato in questo percorso.*

*Ringrazio i miei amici e i miei compagni di corso
per l'enorme sostegno.*

Indice

| | | |
|----------|---|-----------|
| 1 | Introduzione | 6 |
| 1.1 | Obiettivi e motivazioni | 8 |
| 2 | Reverse engineering del protocollo | 9 |
| 2.1 | BLE: principi di funzionamento | 9 |
| 2.2 | Logging tramite Android | 13 |
| 2.3 | Analisi applicazione CalorBT | 16 |
| 3 | Descrizione del protocollo | 21 |
| 3.1 | Servizio e Caratteristiche BLE | 21 |
| 3.2 | Extended Backus-Naur Form | 22 |
| 3.3 | Comandi | 24 |
| 3.4 | Notifiche | 30 |
| 4 | Realizzazione software | 34 |
| 4.1 | Licenza GPL v3 | 34 |
| 4.2 | Funzioni per la gestione di una singola valvola | 35 |
| 4.3 | Script per la gestione di più valvole | 40 |
| 5 | Test sul campo | 46 |
| 5.1 | Portata e relative problematiche | 46 |
| 5.2 | Connessione parallela | 47 |
| 6 | Sviluppi futuri e conclusioni | 48 |

Elenco delle figure

| | | |
|---|--|----|
| 1 | Impianto centralizzato senza contabilizzazione del calore | 6 |
| 2 | Impianto centralizzato con contabilizzazione del calore e termovalvole | 7 |
| 3 | Termovalvola eQ-3 modello Equiva montata | 8 |
| 4 | Topologia | 10 |
| 5 | Gerarchia profili, servizi, caratteristiche | 11 |
| 6 | File di log in Wireshark | 13 |
| 7 | Applicazione CalorBT su Android | 17 |
| 8 | Esempio funzionamento di <code>profile_req.sh</code> | 36 |
| 9 | Sequence diagram dello script <code>auto_mode.sh</code> | 44 |

1 Introduzione

Le valvole termostatiche sono dispositivi di termoregolazione composti da due parti: una valvola e un fluido ad alto coefficiente di dilatazione contenuto in un bulbo a contatto con l'ambiente circostante. L'insieme di queste due componenti permette di creare un sistema che, in funzione di un range di valori posto sulla valvola, espande o contrae il fluido provocando l'attivazione o l'interruzione del flusso che si intende gestire. É quindi chiaro che l'ambito applicativo prevalente è quello degli impianti di riscaldamento e di raffreddamento.

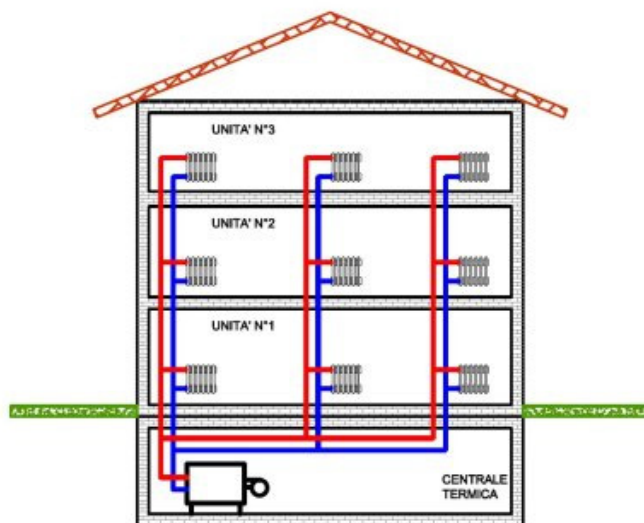


Figura 1: Impianto centralizzato senza contabilizzazione del calore (fonte: [15])

L'utilizzo di valvole termostatiche su caloriferi domestici permette la riduzione di consumi ed emissioni, migliorando inoltre il livello di comfort percepito nell'appartamento in cui vengono applicate. A sancirne l'importanza è l'Unione Europea, la quale, come previsto dal decreto di recepimento della direttiva 2012/27/UE [1, 5, 7, 14], richiede che entro il 31 Dicembre 2016 tutti gli italiani residenti in condomini con impianto centralizzato (Figura 1) installino le termovalvole sui propri termosifoni affiancate da sistemi di contabilizzazione. Questi quantificano il calore effettivamente consumato, permettendo un'autonoma gestione delle temperature in ogni unità immobiliare e la suddivisione delle spese secondo i singoli consumi (Figura 2). Il panorama commerciale attuale permette di reperire le valvole termostatiche a prezzi

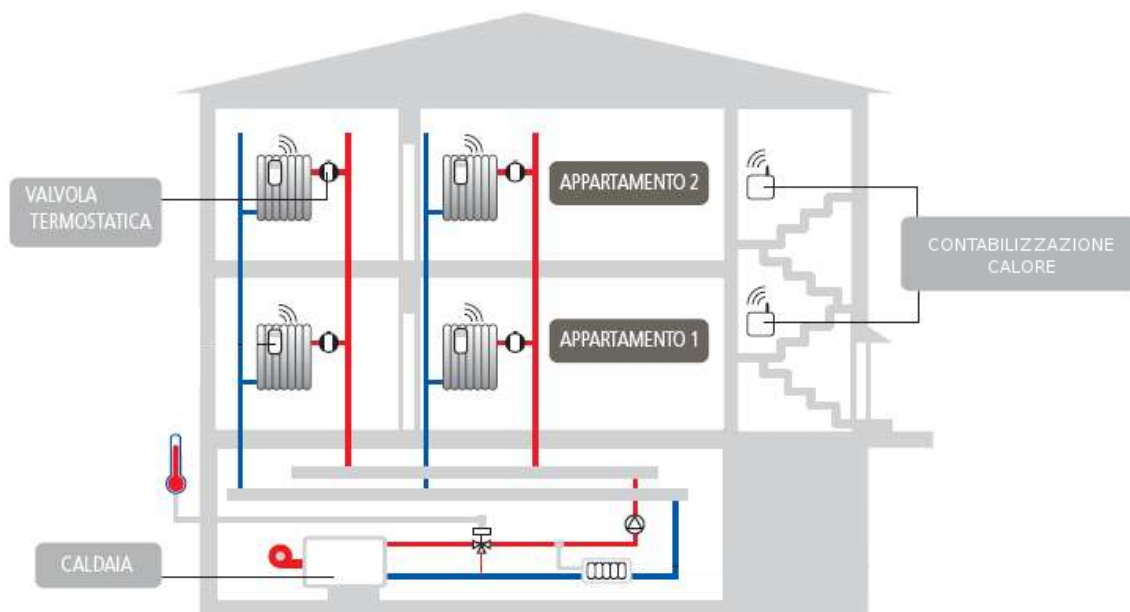


Figura 2: Impianto centralizzato con contabilizzazione del calore e termovalvole (fonte: [8])

modici. Aziende leader del settore termo-sanitario quali *Danfoss*¹, *eQ-3*², *LightwaveRF*³ e le italiane *Tiemme*⁴, *FAR*⁵ ed *Emmeti*⁶ offrono buone soluzioni anche per poche decine di euro.

Da pochi anni sono disponibili sul mercato versioni elettroniche di queste valvole, alimentate a batteria e dotate di un termostato integrato in sostituzione al classico fluido. Queste offrono la possibilità di programmarle lungo tutto l'arco settimanale e consentono una selezione più accurata della temperatura desiderata, aspetto che ha assunto particolare importanza a fronte dell'introduzione dei contacalorie. Per agevolare l'utilizzo, l'interazione con le termovalvole elettroniche avviene usualmente tramite un'applicazione *proprietaria* per smartphone che sfrutta una connessione Bluetooth o Wi-Fi per lo scambio dei dati.

¹www.danfoss.com

²www.eq-3.de

³www.lightwaverf.com

⁴www.tiemme.com

⁵www.far-spa.it

⁶www.emmeti.com



Figura 3: Termovalvola eQ-3 modello Equiva montata

1.1 Obiettivi e motivazioni

Il lavoro svolto prende in considerazione le valvole della serie *Equiva* prodotte dall'azienda tedesca *eQ-3* (Figura 3) con la relativa applicazione *CalorBT* disponibile per dispositivi Android e iOS e si pone due obiettivi:

- risalire al protocollo che l'applicazione utilizza per la comunicazione
- realizzare un software *libero* che ne consenta la gestione

Il risultato di ciò si traduce nella possibilità di integrare questo prodotto in impianti di domotica libera e di generare software ad hoc che ne svincoli l'utilizzo da quello di uno smartphone.

A tale fine si è scelto di scrivere una serie di funzioni di semplice utilizzo che consentono tramite Bluetooth la gestione dei vari aspetti di una singola valvola e il parsing delle notifiche ricevute. A queste sono stati affiancati alcuni script che automatizzano la comunicazione con più dispositivi in contemporanea e forniscono esempi di funzionamento di alcune funzioni implementate.

2 Reverse engineering del protocollo

La comunicazione tra smartphone e valvola termostatica avviene attraverso la tecnologia Bluetooth Low Energy (BLE o Bluetooth Smart) che è stata introdotta nel 2010 come parte della specifica Bluetooth 4.0. Inizialmente sviluppata come progetto personale dall'azienda finlandese Nokia e solo successivamente integrata nello standard, essa vuole garantire bassi consumi energetici pur mantenendo un ampio range di comunicazione. L'importanza assunta dal BLE nel panorama Internet Of Things e nella progettazione di dispositivi in grado di comunicare con tutte le piattaforme più moderne ha fatto sì che i principali produttori di sistemi operativi in ambito desktop e mobile ne forniscano il completo supporto.

Il lavoro di reverse engineering si è svolto seguendo parallelamente due strade: il logging e l'ispezione dei pacchetti Bluetooth scambiati tra smartphone e valvola, e la decompilazione dell'applicazione su piattaforma Android. Questo ha permesso di intuire il protocollo utilizzato, sfruttando il codice ottenuto per verificare quanto dedotto e garantire maggiore coerenza rispetto alle specifiche originali.

2.1 BLE: principi di funzionamento

La connessione e la trasmissione di dati tra due dispositivi prevede il susseguirsi di più fasi e il coinvolgimento di più elementi. Di seguito verranno elencate e discusse le componenti di alto livello più importanti.

GAP (Generic Access Profile)

Affinché la presenza di un dispositivo BLE venga notificata al mondo esterno è necessario un processo denominato *advertising*. Esso consiste sostanzialmente nel continuo invio di pacchetti informativi verso le periferiche abilitate all'utilizzo del Bluetooth entro una certa distanza. Ciò che gestisce gli aspetti relativi alla connessione, all'*advertising* e determina infine se due dispositivi possono tra loro interagire è il GAP, acronimo di Generic Access Profile.

Di fondamentale importanza è la suddivisione in ruoli. Distingueremo:

- **Dispositivi periferici** (o *periferiche*) dotati di poche risorse e di natura estremamente variabile
- **Dispositivi centrali** quali smartphone, tablet e computer caratterizzati da molta potenza di calcolo e memoria

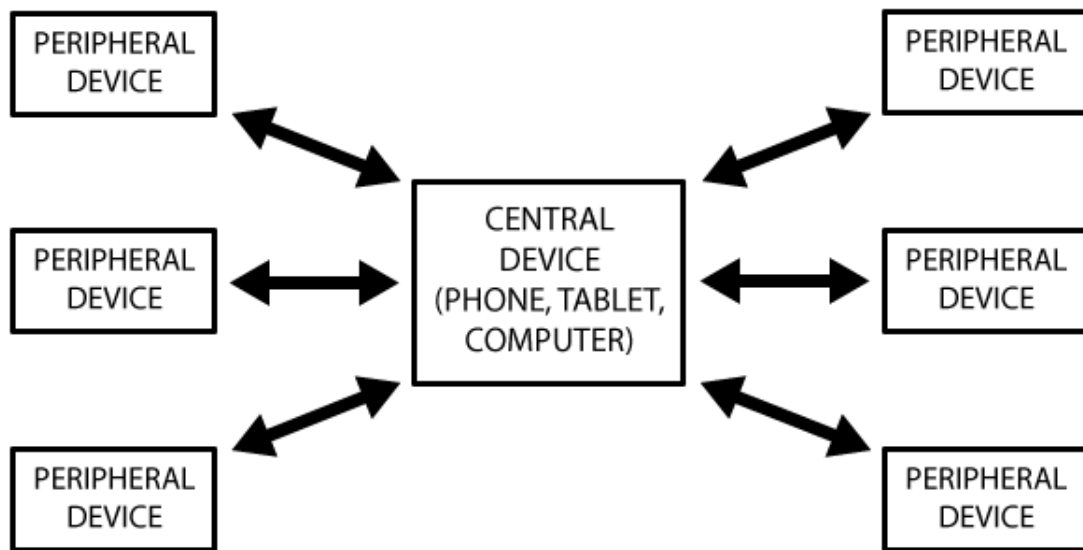


Figura 4: Topologia (fonte: [13])

Dispositivi periferici non possono essere tra loro connessi e dialogano con un solo dispositivo centrale per volta (Figura 4). Pertanto, stabilita una connessione, essi bloccheranno il processo di advertising fino all'interruzione del collegamento. Al contrario, i dispositivi centrali possono gestire contemporaneamente la ricezione e l'invio dei dati a più periferiche. Di conseguenza, la comunicazione tra due periferiche richiede la creazione di un apposito sistema che sfrutti questa possibilità. Le valvole termostatiche che stiamo prendendo in considerazione ricadono chiaramente nella tipologia di dispositivi periferici, mentre l'applicazione *CalorBT* che le gestisce viene installata su un dispositivo centrale.

GATT (Generic Attribute Profile)

A connessione stabilita, la trasmissione bidirezionale avviene attraverso il protocollo

ATT (Attribute Protocol) e utilizza i concetti di profilo, servizio e caratteristica GATT.

Un aspetto significativo è dato dalla relazione che si viene a creare tra il dispositivo periferico, indicato come *GATT Server* (o Slave) in quanto fornitore di servizi e caratteristiche e il dispositivo centrale, detto *GATT Client* (o Master). Tutte le transazioni partono da quest'ultimo e ricevono risposta dallo Slave, il quale al momento della prima connessione suggerisce un *intervallo di connessione*. Allo scadere di ogni intervallo il Master si riconnette per verificare la disponibilità di nuovi dati. Si noti che si tratta solo di un suggerimento fornito dalla periferica, che non pone tuttavia vincoli temporali al dispositivo centrale.

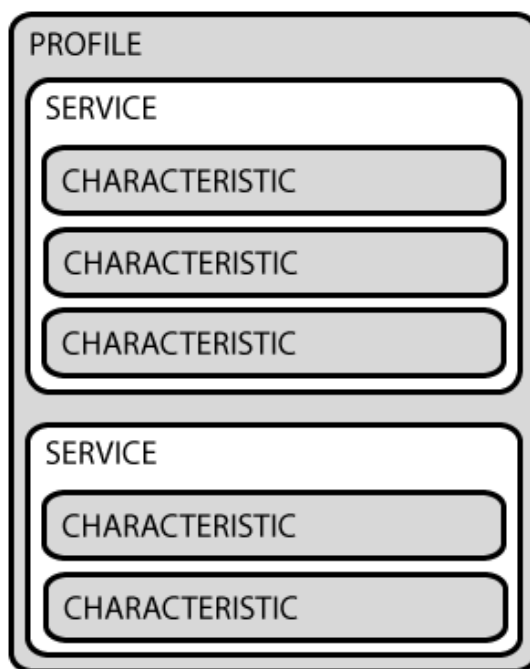


Figura 5: Gerarchia profili, servizi, caratteristiche (fonte: [13])

Come già anticipato, le transazioni GATT sono basate su oggetti di alto livello tra loro gerarchici: profili, servizi e caratteristiche (Figura 5).

Profili Definiscono possibili applicazioni della periferica, descrivendone funzionalità e casi d'uso. La specifica BLE fornisce un'ampia serie di profili standard che trovano utilizzo in svariati ambiti, ma ne consente anche la creazione utilizzando il GATT. Questo facilita lo sviluppo di applicazioni innovative che mantengono comunque l'interoperabilità con altri dispositivi Bluetooth. Ad esempio il "Blood Pressure Profile" ed il "Proximity Profile" sono rispettivamente profili predefiniti per l'interazione con un misuratore di pressione sanguigna e per il monitoraggio della distanza tra due dispositivi.

Servizi Permettono di effettuare una prima e poco approfondita suddivisione logica dei dati e sono composti da una o più caratteristiche. Si identificano attraverso un UUID costituito da 16 bit per i servizi predefiniti e 128 bit per quelli creati appositamente dai produttori di dispositivi periferici. Il sopra citato "Blood Pressure Profile" fornisce i servizi "Blood Pressure Service" e "Device Information Service", necessari alla trasmissione dei dati relativi alla pressione sanguigna e allo stato del dispositivo.

Caratteristiche Sono il punto di interazione principale con la periferica Bluetooth e rappresentano lo strato più granulare nella suddivisione logica dei dati. Ogni caratteristica gestisce informazioni relative ad un singolo aspetto, occupandosi della trasmissione in una o entrambe le direzioni. Così come i servizi, anch'esse vengono identificate attraverso dei UUID di 16 o 128 bit. Ad esempio la caratteristica "Blood Pressure Measurement" fornita dal servizio "Blood Pressure Service" potrà essere utilizzata per la lettura dei valori rilevati dal misuratore di pressione sanguigna.

Come già detto, è usualmente il GATT Client a dare l'avvio a una transazione. Tuttavia il BLE prevede l'utilizzo di *notifiche* e *indications* affinché il GATT Server possa effettuare richieste di dati o semplicemente inviare informazioni alla controparte senza un esplicito segnale da parte di quest'ultima. In generale, le notifiche vengono utilizzate per informare il client riguardo il valore assunto da una caratte-

ristica. Perché il meccanismo funzioni è comunque necessaria una richiesta esplicita di ricezione delle notifiche da parte del client stesso.

2.2 Logging tramite Android

In ambito informatico il *logging* è l'attività di registrazione di dati e informazioni relative a determinate operazioni man mano che vengono effettuate. In questo specifico caso si tratta di tracciare tutti i pacchetti Bluetooth scambiati durante la comunicazione tra valvola termostatica e smartphone al fine di poterne ispezionare il contenuto. Dalla versione 4.4 "KitKat", Android introduce la possibilità di effettuare il logging dei pacchetti inviati e ricevuti via Bluetooth tramite la funzione "Attiva log di esame HCI Bluetooth" presente nella sezione Opzioni Sviluppatore. Una volta attivata, il sistema inizia a popolare il file `/sdcard/bootsnoop_hci.log` in un formato compatibile a quello richiesto da molti software per l'analisi di protocolli.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-----------|--------------|--------------|----------|--------|--|
| 124 | 14.179333 | remote () | localhost () | ATT | 15 | Rcvd Find Information Response |
| 127 | 14.195621 | localhost () | remote () | ATT | 12 | Sent Read Request, Handle: 0x0430 |
| 128 | 14.209384 | remote () | localhost () | ATT | 12 | Rcvd Read Response |
| 129 | 14.216888 | localhost () | remote () | ATT | 19 | Sent Write Request, Handle: 0x0411 |
| 132 | 14.298143 | remote () | localhost () | ATT | 10 | Rcvd Write Response |
| 133 | 14.347386 | remote () | localhost () | ATT | 18 | Rcvd Handle Value Notification, Handle: 0x0421 |
| 134 | 31.961833 | localhost () | remote () | ATT | 14 | Sent Write Request, Handle: 0x0411 |
| 135 | 32.043119 | remote () | localhost () | ATT | 10 | Rcvd Write Response |
| 136 | 32.141600 | remote () | localhost () | ATT | 18 | Rcvd Handle Value Notification, Handle: 0x0421 |
| 138 | 66.802989 | localhost () | remote () | ATT | 14 | Sent Write Request, Handle: 0x0411 |
| 139 | 66.899402 | remote () | localhost () | ATT | 10 | Rcvd Write Response |
| 140 | 66.997462 | remote () | localhost () | ATT | 18 | Rcvd Handle Value Notification, Handle: 0x0421 |

| |
|--|
| ► Frame 134: 14 bytes on wire (112 bits), 14 bytes captured (112 bits) |
| ► Bluetooth HCI H4 |
| ► Bluetooth HCI ACL Packet |
| ▼ Bluetooth L2CAP Protocol |
| Length: 5 |
| CID: Attribute Protocol (0x0004) |
| ▼ Bluetooth Attribute Protocol |
| Opcode: Write Request (0x12) |
| Handle: 0x0411 |
| Value: 4128 |

| |
|---|
| 0000 02 40 00 09 00 05 00 04 00 12 11 04 41 28 .@.....A |
|---|

Figura 6: File di log in Wireshark

Il software scelto per l'ispezione è Wireshark, ma l'analisi non è immediata in quanto di default vengono visualizzati i pacchetti relativi a tutti i protocolli coinvolti nella comunicazione. Come visto precedentemente, la gestione di Servizi e Caratteristiche viene affidata al Generic Attribute Profile che sfrutta il protocollo ATT, identificato

da Wireshark attraverso il CID 0x0004. É quindi possibile rimuovere dall'elenco i pacchetti in eccesso e mantenere solo ciò a cui siamo interessati inserendo nella barra dei filtri l'espressione `'btl2cap.cid == 0x0004'`. Fatto questo diventa immediato osservare quali sono le caratteristiche su cui è stata effettuata un'operazione di scrittura, i relativi valori scritti e il contenuto delle notifiche ricevute. Infatti, selezionando un pacchetto ed espandendo la sezione *Bluetooth Attribute Protocol*, si rendono visibili tutti i dettagli necessari quali il tipo di operazione eseguita, la caratteristica su cui è stata effettuata (identificata da un *Handle* di 16 bit) e i dati trasmessi (Figura 6). Il lavoro di reverse engineering diviene quindi realizzabile applicando le seguenti metodologie.

Operazioni che richiedono parametri

Si tratta di operazioni che necessitano di alcuni dettagli forniti dall'utente. Ad esempio, l'impostazione della temperatura richiede che questa sia stata specificata. Il valore inviato alla valvola sarà quindi divisibile in due sezioni: un *pattern comune*, rappresentante il codice operativo dell'istruzione, e una *parte variabile* in base a quanto fornito dall'utente. L'idea è quella di raggruppare in un singolo file di log l'esecuzione di più operazioni dello stesso tipo, modificando per ognuna il parametro non comune. Spesso questa tecnica rende possibile identificare il pattern e nei casi poco complessi permette di intuire il criterio utilizzato per calcolare la parte variabile.

Impostando consecutivamente la temperatura a 18 °C, 20 °C e 21.5 °C e applicando la procedura appena descritta è possibile osservare che i valori inviati alla valvola sono rispettivamente *0x4124*, *0x4128* e *0x412B* (un esempio è visibile in Figura 6). É immediato notare che il byte 0x41 compare in tutti e tre i casi ed è quindi ragionevole supporre che identifichi il tipo di operazione da svolgere. Conseguentemente, il byte restante rappresenterà una codifica della temperatura scelta dal costruttore. Convertendo i byte esadecimali 0x24, 0x28 e 0x2B in base 10 si ottengono rispettivamente 36, 40 e 43, corrispondenti al doppio dei valori iniziali 18, 20 e 21,5. É quindi chiaro che la codifica utilizzata consiste nel moltiplicare per due la temperatura

desiderata.

Operazioni che non richiedono parametri

Si tratta di operazioni la cui esecuzione non richiede dati esterni. Presumibilmente il valore inviato alla valvola per provocarne l'attivazione sarà composto solo da un codice operativo invariante. Esempi significativi sono i comandi necessari a porre la valvola in modalità manuale o automatica e quelli per attivare o fermare la funzionalità "boost". In questi casi l'obiettivo è quello di creare file di log che rispecchino l'esecuzione di una singola istruzione, in modo da poter identificare immediatamente ciò che è stato trasmesso alla valvola. È utile compiere la stessa operazione partendo da condizioni diverse al fine di verificare l'effettiva invariabilità del codice operativo in tutti gli "stati" in cui il dispositivo si può trovare.

Ad esempio, tramite i file di log è facile osservare che l'esecuzione della funzione "boost" provoca l'invio del valore `0x4501` alla valvola, indipendentemente dal fatto che essa si trovi in modalità automatica, manuale o vacanza. Questo permette di dedurre che con molta probabilità i due byte `0x4501` rappresentano il codice operativo invariante dell'istruzione presa in considerazione.

Notifiche

Le notifiche vengono inviate dalla valvola al dispositivo centrale a seguito dell'esecuzione di ogni comando. Non è necessario applicare un metodo specifico per rilevarle perché sono già presenti nei file di log creati precedentemente. Wireshark ne segnala la presenza attraverso la descrizione "Rcvd Handle Value Notification, Handle: 0x0421" nella colonna "Info". Non c'è modo di conoscere a prescindere la composizione di una notifica; queste potrebbero contenere molte informazioni o anche solo indicare una conferma di avvenuta esecuzione. Ispezionando i pacchetti intercettati si può tuttavia notare che, come per le operazioni con parametri, anche le notifiche sono spesso composte da *pattern comuni* rappresentativi della tipologia della notifica e *valori variabili* che forniscono informazioni dettagliate generalmente codificate secondo criteri scelti dal produttore. Ad esempio, i primi byte permet-

tono di effettuare un'ampia suddivisione: 0x0202 e 0x21 indicano notifiche relative rispettivamente alla scrittura e alla richiesta di un profilo giornaliero, mentre 0x0201 identifica quelle conseguenti all'esecuzione di una qualsiasi altra operazione.

L'applicazione di quanto appena descritto a tutte le funzionalità della valvola termostatica permette di risalire al protocollo nella sua integrità, lasciando tuttavia delle perplessità sulle codifiche utilizzate nella trasmissione dei dati variabili all'interno vari comandi. L'analisi dell'applicazione consente di colmare queste lacune e appurare la correttezza di quanto dedotto.

2.3 Analisi applicazione CalorBT

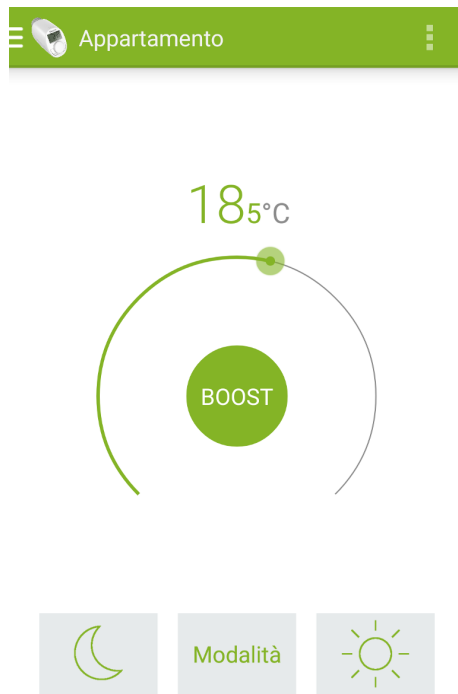
L'applicazione CalorBT, sviluppata direttamente dall'azienda eQ-3, è disponibile per piattaforma *Android 4.4+* e *iOS 8.3+* sui rispettivi Play Store e App Store. La versione presa in considerazione è la 1.1.7 aggiornata al mese di Gennaio 2016. Il funzionamento è banale e la fase di inizializzazione richiede solo l'accoppiamento del dispositivo con le valvole desiderate tramite una semplice procedura di pairing. È interessante notare che l'applicazione permette di comunicare con *una sola periferica per volta*. Effettuata la connessione, si viene quindi rimandati a un'attività di gestione della temperatura (Figura 7a), dalla quale è possibile accedere a tutte le altre funzionalità.

Per ottenere il codice sorgente si è scelto di utilizzare due servizi web gratuiti. Il primo, *APKPure*¹, estrae dal Play Store i pacchetti APK relativi alle applicazioni gratuite. Il secondo, *Java Decompiler*², ha poi permesso di sfruttare il file APK per generare un archivio contenente quanto desiderato. Il risultato viene automaticamente identificato come "progetto Android" da parte di *Android Studio*³ e questo ne garantisce maggiore facilità di lettura e analisi, sfruttando le funzionalità di sviluppo avanzate fornite dal software.

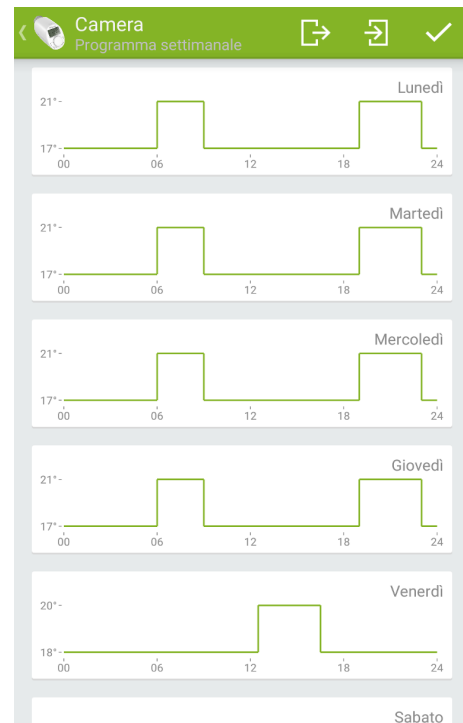
¹www.apkpure.com

²www.javadecompiler.com/apk

³developer.android.com/studio/index.html



(a) Schermata principale



(b) Programmazione settimanale

Figura 7: Applicazione CalorBT su Android

Una volta importato il progetto, è immediato notare come il codice sorgente sia privo di offuscamento. I file sono infatti leggibili nella loro integrità e perfettamente coerenti tra loro. Inoltre, il decompilatore ha mantenuto la suddivisione in *package* dell'intera applicazione. Si tratta di un risultato inaspettato e per nulla scontato che lascia intendere come l'azienda produttrice di queste valvole non abbia cercato di ostacolare eventuali lavori di reverse engineering. Nonostante questo, l'analisi non è immediata; escludendo le librerie esterne quali *ButterKnife* e *FasterXML* l'applicazione si compone di 1398 file di cui ben 144 classi e 16 interfacce Java. È quindi un progetto abbastanza ampio, la cui logica applicativa e gestione dei layout constano rispettivamente di 14338 e 4940 linee di codice¹. L'obiettivo prefissato è quello di analizzare il package dedicato alla gestione della comunicazione tra dispositivo centrale e valvola (*de.eq3.ble.android*) per verificare la correttezza della sintassi e della composizione di comandi e notifiche dedotti tramite l'attività di logging.

¹Calcolate utilizzando il tool 'SLOCCount' di David A. Wheeler

Comandi

Accanto ai file “Java” necessari alla connessione Bluetooth e alla gestione di layout e activity dell’applicazione troviamo il package *de.eq3.ble.android.api.command*, il quale contiene il codice necessario alla composizione di tutti i possibili comandi suddivisi in singole classi. Questo tipo di organizzazione è di particolare aiuto in quanto permette di venire a conoscenza di tutte le istruzioni che l’applicazione può inviare alla valvola fornendo per ognuna informazioni dettagliate. Ogni classe è stata progettata secondo lo stesso principio: il costruttore si occupa di comporre e immagazzinare in un array il valore corrispondente all’istruzione, mentre le activity esterne possono prelevare attraverso il metodo pubblico *getCommandData()*, necessariamente implementato in quanto richiesto dall’interfaccia *IThermostatCommand*.

L’esempio visto nella sezione “Logging tramite Android - Operazioni che richiedono parametri” fa riferimento alla classe *SetTemperatureCommand* di seguito riportata.

```
1  public class SetTemperatureCommand implements IThermostatCommand {
2      private final byte[] commandData;
3      public SetTemperatureCommand(Number setPointTemperature) {
4          this.commandData = new byte[2];
5          this.commandData[0] = (byte) 65;
6          this.commandData[1] = (byte) ((int)
              (setPointTemperature.doubleValue() * 2.0d));
7      }
8
9      public byte[] getCommandData() {
10         return this.commandData;
11     }
12 }
```

L’attività di logging aveva identificato il byte 0x41 come pattern comune a tutte le istruzioni di questa tipologia e un secondo byte come parte variabile equivalente alla temperatura desiderata e raddoppiata. La riga 4 del codice sopra esposto mostra che il comando preso in considerazione sarà composto da 2 byte, il cui contenuto

si evince dalle righe 5 e 6. Il primo contiene sempre il valore *65*, la cui conversione in esadecimale corrisponde proprio a *0x41*. Il secondo è invece variabile in base al parametro *setPointTemperature* fornito al costruttore della classe e rispecchia la codifica della temperatura già citata. È interessante notare come la riga 6 non solo fornisca informazioni sul significato del secondo byte, ma ne indichi anche il metodo utilizzato per il calcolo. Questo è di particolare ausilio nella creazione delle istruzioni all'interno del software di gestione discusso nella sezione "4 - Realizzazione Software", affinché possano essere coerenti con ciò che viene richiesto dalla valvola.

Pertanto, quanto appena visto conferma le deduzioni tratte dall'attività di logging, ne approfondisce alcuni aspetti e si pone come strumento di verifica applicabile a tutti i comandi.

Notifiche

Ricevuta una notifica, il suo contenuto viene interpretato dalla funzione *updateDeviceState*, il cui prototipo è:

```
public void updateDeviceState(String deviceId, byte[] value)
```

Essa è situata nel file *de.eq3.ble.android.api.BluetoothAPI.java* e, così come per i comandi, anche in questo caso i dati sono memorizzati in un array di byte (rappresentato dal secondo argomento "value"). *UpdateDeviceState* si occupa di riconoscere la tipologia della notifica leggendo il valore contenuto nel primo e, nei casi di ambiguità, nel secondo byte e in base a ciò di rinviarne la gestione ad altri metodi. Questo ha permesso di effettuare una prima suddivisione, peraltro coerente con quella individuata nella sezione "Logging tramite Android - Notifiche", per poi ottenere informazioni più dettagliate tramite l'analisi delle funzioni richiamate.

Il metodo adottato è quindi concettualmente molto semplice. Ipotizzando, ad esempio, di avere inviato alla valvola una richiesta di lettura di un profilo giornaliero, otterremo una notifica il cui primo byte corrisponderà al valore *0x21*, cioè *33* nel sistema decimale.

Questa è quindi la parte di codice interessata all'interno della funzione *updateDeviceState*:

```
1    int frameType = value[0] & 255;  //primo byte
2    ..
3    if (frameType == 33) {
4        dayOfWeek = ModelUtil.getDayOfWeek(value[1]);
5        byte[] profileData = new byte[(value.length - 2)];
6        for (int i = 0; i < profileData.length; i++){
7            profileData[i] = value[i + 2];
8        }
9        this.profileListener.profileDataReceived(dayOfWeek, profileData);
10   }
11   ..
```

Come mostra la linea 9, la gestione del tutto viene rimandata alla funzione *profileDataRecived* con due parametri: il giorno della settimana, codificato secondo il metodo statico *getDayOfTheWeek* (riga 4) e il contenuto dell'array "value", rinominato "profileData" perchè privato delle prime due celle (attraverso il ciclo for di riga 6 e 7). É chiaro quindi che si tratta di analizzare le due funzioni appena citate, le quali daranno spazio ad approfondimenti relativi alla semantica di ogni byte e ai metodi di decodifica delle informazioni contenute.

Infine, è interessante notare come il codice delle classi fornite dal package *de.eq3.ble.android.api.command* e il contenuto dei file inerenti la gestione delle notifiche non coinvolgano parametri dipendenti dalla modalità in cui si trova la valvola al momento di esecuzione dell'istruzione.

3 Descrizione del protocollo

Il protocollo utilizzato permette di individuare due metodi di comunicazione: l'invio di comandi alla valvola seguito dalla ricezione della rispettiva notifica entro un breve intervallo di tempo e la ricezione di notifiche a sè stanti. Tutti i valori scambiati contengono le informazioni necessarie alla loro interpretazione, pertanto può essere considerato un protocollo *stateless*. Di seguito ne verrà fornita una descrizione dettagliata.

3.1 Servizio e Caratteristiche BLE

Le due caratteristiche utilizzate nella comunicazione fanno entrambe parte di un servizio identificato dall'UUID "3e135142-654f-9090-134a-a6ff5bb77046".

Caratteristica per l'invio dei comandi

- Proprietà: read/write
- UUID: 3fa4585a-ce4a-3bad-db4b-b8df8179ea09
- Handle: 0x0411

Caratteristica per la ricezione delle notifiche

- Proprietà: read/write/notify
- UUID: d0e8434d-cd29-0996-af41-6c90f4e0eb2a
- Handle: 0x0421

Client Characteristic Configuration Descriptor (CCID)

Questo descrittore, il cui UUID è "00002902-0000-1000-8000-00805f9b34fb", non è stato utilizzato nel lavoro svolto. Il suo ruolo è tuttavia fondamentale in altri contesti, tra cui lo sviluppo di applicazioni per dispositivi mobili. Consente infatti di attivare la ricezione di *notifiche* impostandone il valore a 1 attraverso un'operazione di scrittura.

Tutti i codici identificativi sono composti da 128 bit. Questo mostra che si tratta di servizi e caratteristiche non previsti dalla specifica Bluetooth LE, ma implementati appositamente dal produttore.

3.2 Extended Backus-Naur Form

La Backus-Naur Form [3] è un formalismo frequentemente utilizzato nella descrizione della sintassi e della grammatica di protocolli e linguaggi. Di seguito ne verrà utilizzata la versione *Extended* [9], concepita per una rappresentazione più chiara e compatta e ormai universalmente riconosciuta. É bene ricordare che il simbolo “|” indica delle possibili alternative, “*” il numero di ripetizioni e le parentesi quadre identificano simboli opzionali.

```
protocollo = comando | notifica;

(* DESCRIZIONE COMANDI *)

comando =
    set-data-ora | set-temp | set-comfort | set-ridotta | modifica-comf-rid
    | boost | auto | manuale | vacanza | lock | crea-profilo | leggi-profilo
    | finestra | set-offset;

set-data-ora = '03', anno, mese, giorno, ora, minuti, secondi;
set-temp = '41', temperatura;
set-comfort = '43';
set-ridotta = '44';
modifica-comf-rid = '11', temperatura, temperatura;
boost = '45', (on | off);
auto = '4000';
manuale = '4040';
vacanza = '40', temperatura-128, giorno, anno, ora-e-minuti, mese;
lock = '80', (on | off);
crea-profilo = '10', giorno-settimana, intervallo, 6*[intervallo];
leggi-profilo = '20', giorno-settimana;
```

`finestra = '14', temperatura, minuti-finestra;`

`set-offset = '13', offset-range;`

`(* DESCRIZIONE NOTIFICHE *)`

`notifica = notifica-stato | notifica-profilo ;`

`notifica-stato = '02', '01', stato-valvola, [parametri-vacanza];`

`notifica-profilo = successo-modifica, letto-profilo;`

`stato-valvola = modo, byte, '04', temperatura;`

`parametri-vacanza = giorno, anno, ora-e-minuti, mese;`

`sucesso-modifica = '0202', giorno-settimana;`

`letto-profilo = '21', giorno-settimana, 7*intervallo;`

`(* TIPI GENERICI *)`

`on = '01';`

`off = '00';`

`anno = ('0' | '1' | '2' | '3' | '4' | '5' | '6'), hexdigit;`

`mese =`

`'0', ('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'A' |
'B' | 'C');`

`giorno = ('0' | '1'), hexdigit;`

`ora = ('0' | '1'), hexdigit;`

`minuti = ('0' | '1' | '2' | '3'), hexdigit;`

`secondi = ('0' | '1' | '2' | '3'), hexdigit;`

`temperatura-128 = ('8' | '9' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'), hexdigit;`

`temperatura = byte;`

`ora-e-minuti = byte`

`giorno-settimana = '0', ('0' | '1' | '2' | '3' | '4' | '5' | '6');`

`intervallo = temperatura, byte;`

`minuti-finestra =`

`'0', ('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'A' |
'B' | 'C');`

```

modo = ('0' | '2'), ('8' | '9' | 'A' | 'C' | 'D' | 'E');
offset-range =
    '0', ('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'A' |
    'B' | 'C' | 'D' | 'E');
byte = hexdigit, hexdigit;
hexdigit =
    '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'A' | 'B' |
    'C' | 'D' | 'E' | 'F';

```

3.3 Comandi

I comandi devono essere inviati alla valvola utilizzando il sistema numerico esadecimale, pertanto *i valori di seguito riportati sono già stati convertiti*. I caratteri alfabetici possono essere indifferentemente utilizzati in maiuscolo o in minuscolo. Infine è interessante notare che la valvola, e conseguentemente anche l'applicazione CalorBT, è in grado di gestire solo temperature la cui parte decimale è arrotondata a *.0* o *.5*.

Impostazione data e ora attuali (7 byte)

Comunica alla valvola la data e l'orario attuali. In generale questo comando viene utilizzato per sincronizzarsi con la periferica. La notifica restituita a seguito della sua esecuzione permette infatti di ricavarne informazioni relative allo stato.

byte 0: 03

byte 1: anno%100

byte 2: mese

byte 3: giorno

byte 4: ora

byte 5: minuti

byte 6: secondi

NOTA: mesi e giorni sono calcolati partendo da 1. Di conseguenza sia il mese

di Gennaio che il primo giorno di ogni mese verranno identificati attraverso il valore *0x01*.

ESEMPIO

La data e l'ora 25/05/2016 11:27:28 diventano *03 10 05 19 0B 1B 1C*. Allo stesso modo 29/08/2016 09:55:20 diviene *03 10 08 1D 09 37 14*.

Seleziona temperatura a scelta (2 byte)

Imposta la temperatura selezionata.

byte 0: 41

byte 1: temperatura * 2

ESEMPIO

L'impostazione a 18°C richiede l'invio del comando *4124* in quanto $18 * 2 = 36 = 0x24$. Allo stesso modo per impostare la valvola a 20.5°C è necessario inviare *4129*.

Seleziona temperatura comfort (1 byte)

Imposta la temperatura comfort sulla valvola.

byte 0: 43

Seleziona temperatura ridotta (1 byte)

Imposta la temperatura ridotta sulla valvola.

byte 0: 44

Impostazione temperatura comfort e ridotta (3 byte)

Permette di modificare i valori predefiniti della temperatura comfort e ridotta all'interno delle impostazioni della valvola.

byte 0: 11

byte 1: nuova_comfort * 2

byte 2: nuova_ridotta * 2

ESEMPIO

Per impostare i valori predefiniti delle temperature comfort e ridotta rispettivamente a 23°C e 18.5°C è necessario inviare alla valvola il comando *112E25*, in quanto $23 * 2 = 46 = 0x2E$ mentre $18.5 * 2 = 37 = 0x25$.

Attiva/Disattiva boost mode (2 byte)

Attiva o disattiva la modalità boost sulla valvola.

byte 0: 45

byte 1: 01 on // 00 off

Seleziona modalità automatica (2 byte)

Attiva la modalità automatica sulla valvola. La temperatura rispecchierà quella selezionata attraverso la programmazione settimanale.

byte 0: 40

byte 1: 00

Seleziona modalità manuale (2 byte)

Attiva la modalità manuale sulla valvola. La temperatura dovrà essere selezionata attraverso i comandi già riportati.

byte 0: 40

byte 1: 40

Seleziona modalità vacanza (6 byte)

Attiva la modalità vacanza sulla valvola. Richiede la temperatura da mantenere insieme alla data e all'orario di fine.

byte 0: 40

byte 1: $(\text{temperatura} * 2) + 128$

byte 2: giorno

byte 3: $\text{anno} \% 100$

byte 4: $(\text{ora} * 2) + (\text{minuti} / 30)$

byte 5: mese

NOTA: mesi e giorni sono calcolati partendo da 1. Di conseguenza sia il mese di Gennaio che il primo giorno di ogni mese verranno identificati attraverso il valore *0x01*. Inoltre, è possibile programmare i minuti di fine vacanza solo a intervalli di mezz'ora (XX:00 o XX:30) pertanto *il valore di (minuti/30) sarà sempre equivalente a 0 oppure 1*.

ESEMPIO

Per mantenere la temperatura a 17.5°C fino alle ore 20:00 del 10/09/2017 è sufficiente inviare alla valvola *40 A3 0A 11 28 09*. Il byte *0xA3* deriva dal calcolo $(17.5 * 2) + 128 = 163 = 0xA3$, mentre il byte *0x28* è stato calcolato attraverso l'orario selezionato come $(20 * 2) + (00/30) = 40 + 0 = 0x28$.

Attiva/Disattiva blocco comandi (2 byte)

Permette di bloccare i tasti fisici presenti sulla valvola, ma ne consente ugualmente la gestione attraverso l'applicazione.

byte 0: 80

byte 1: 01 on // 00 off

Impostazione offset temperatura (2 byte)

Permette di impostare un offset della temperatura in un range tra -3.5°C e +3.5°C.

byte 0: 13

byte 1: $(\text{temperatura} * 2) + 7$

Impostazione modalità finestra (3 byte)

Permette di impostare la durata e la temperatura da mantenere quando subentra la modalità finestra. Questa viene attivata automaticamente dalla valvola in presenza di un significativo calo di temperatura.

byte 0: 14

byte 1: (temperatura * 2)

byte 2: (minuti / 5)

NOTA: i minuti possono assumere soltanto valori multipli di 5, pertanto il contenuto finale del byte 2 sarà compreso tra *0x00* e *0x0C*.

ESEMPIO

Per modificare le impostazioni della modalità finestra a 12°C per la durata di 15 minuti è necessario inviare il comando *14 18 03*. Notiamo infatti che $12 * 2 = 24 = 0x18$ e $15 / 5 = 3 = 0x03$. Seguendo calcoli analoghi, è possibile impostare la stessa modalità a 16°C per 20 minuti attraverso il comando *14 20 04*.

Richiesta profilo giornaliero (2 byte)

Richiede alla valvola i dati relativi alla programmazione di una determinata giornata. L'informazione viene ricevuta sotto forma di notifica.

byte 0: 10

byte 1: giorno della settimana

NOTA: i giorni della settimana vengono contati partendo da sabato. (*00 = Sabato*, ..., *06 = Venerdì*)

Impostazione profilo giornaliero (16 byte max)

Gestisce la programmazione relativa ad un dato giorno della settimana. É necessario scegliere una temperatura base ed è possibile modificarla per *al più tre intervalli* di

tempo. Se per il giorno scelto è già presente un profilo esso viene rimpiazzato.

byte 0: 10

byte 1: giorno della settimana

[byte 2-15]: una sequenza di al più sette coppie di byte (XX,YY) in ognuna delle quali:

- YY rappresenta l'orario, codificato come $(numero_minuti/10)$, fino al quale è necessario mantenere la temperatura dichiarata in XX
- XX rappresenta la temperatura da mantenere fino a quel momento codificata come $(temperatura*2)$

NOTA: l'intera sequenza di byte [2-15] deve permettere di dedurre la temperatura da mantenere in ogni momento della giornata. Eventuali coppie di byte non necessarie perché in eccesso possono essere mantenute a zero od omesse. Il *numero_minuti* si intende calcolato dall'inizio della giornata (00:00). Inoltre, i giorni della settimana vengono contati partendo da sabato (*00 = Sabato, .., 06 = Venerdì*).

ESEMPIO

Si vuole programmare la valvola affinché ogni Martedì mantenga una temperatura base di 17°C, impostandosi automaticamente a 20°C nella fascia 10:00-12:30, 19°C nella fascia 12:30-14:00 e nuovamente 20°C nella fascia 15:00-17:00. Il comando da inviare alla valvola è "10 03 22 3C 28 4B 26 54 22 5A 28 66 22 90 00 00", costruito nel seguente modo:

byte 0: 10 (valore predefinito)

byte 1: 03 (martedì = 0x03, come spiegato nella precedente nota)

byte (2,3): 22 3C (17 gradi, temperatura base, fino alle 10:00)

byte (4,5): 28 4B (20 gradi fino alle 12:30)

byte (6,7): 26 54 (19 gradi fino alle 14:00)

byte (8,9): 22 5A (17 gradi fino alle 15:00)

byte (10,11): 28 66 (20 gradi fino alle 17:00)

byte (12,13): *22 90* (17 gradi, temperatura base, fino alle 24:00)

byte (14,15): *00 00* (non necessari, possono essere omessi)

3.4 Notifiche

Come già discusso, le notifiche vengono inviate dalla valvola al dispositivo centrale al fine di riportare all'utente lo stato della periferica o l'esito di un'operazione. Anche in questo caso i valori sono già convertiti nel sistema numerico esadecimale.

Notifiche di stato - Mod. Auto/Manuale (6 byte)

Si presentano quando la valvola è in modalità automatica o manuale a seguito dell'esecuzione di tutti i comandi, esclusi quelli relativi alla lettura o scrittura dei profili.

byte 0: *02*

byte 1: *01*

byte 2: identificando con X i primi 4 bit e Y i restanti 4 bit:

- il valore di X indica se è attivo il blocco dei tasti fisici:
X=*0* tasti sbloccati
X=*2* tasti bloccati
- il valore di Y indica la modalità in cui si trova la valvola:
Y=*8* modalità automatica
Y=*9* modalità manuale
Y=*A* modalità vacanza
Y=*C* modalità boost. al termine torna in modalità automatica
Y=*D* modalità boost. al termine torna in modalità manuale
Y=*E* modalità boost. al termine torna in modalità vacanza

byte 3: non definito

byte 4: non definito

byte 5: *temperatura*2*

ESEMPIO

Se la valvola è in modalità automatica, impostata a 20°C e senza tasti fisici bloccati, eseguendo il comando "Attiva boost mode" viene ricevuta la notifica *02 01 0C XX XX 28*. Il byte *0x0C* fornisce infatti l'informazione relativa allo stato di sblocco e alla modalità in uso, mentre l'ultimo byte *0x28* corrisponde al doppio della temperatura impostata. Secondo la stessa logica, ponendo la valvola in modalità manuale, con i tasti fisici bloccati e impostando la temperatura a 21.5°C le notifiche assumono il valore *02 01 29 XX XX 2B*.

Notifiche di stato - Mod. Vacanza (10 byte)

Si presentano a seguito dell'esecuzione di tutti i comandi, esclusi quelli relativi alla lettura o scrittura dei profili, in uno dei seguenti casi:

- la valvola è in modalità vacanza
- la valvola è in modalità boost e al suo termine tornerà in modalità vacanza

byte (0-5): vedi "Notifiche di stato - Mod. Auto/Manuale"

byte 6: *giorno_fine_vacanza*

byte 7: *anno_fine_vacanza%100*

byte 8: $(\text{ora_fine} * 2) + (\text{minuti_fine} / 30)$

byte 9: *mese_fine_vacanza*

NOTA: è possibile programmare i minuti di fine vacanza solo a intervalli di mezz'ora (XX:00 o XX:30) pertanto *il valore di (minuti_fine/30) sarà sempre equivalente a 0 oppure 1*.

ESEMPIO

Attivando la modalità vacanza con i tasti fisici bloccati, temperatura a 20°C e data di termine impostata alle 18:30 del 14/12/2016, la valvola fornisce la seguente notifica: *02 01 2A XX XX 24 0E 10 25 0C*. I primi sei byte sono coerenti con quanto dichiarato nella sezione precedente: *0x2A* dichiara infatti che la modalità vacanza è attiva. Dei restanti quattro byte, *0x0E*, *0x10* e *0x0C* indicano rispettivamente

giorno, anno e mese di termine. Infine, il $0x25$ codifica l'orario "18:30" secondo il metodo descritto: $(18 * 2) + (30/30) = 36 + 1 = 37 = 0x25$.

Notifiche di modifica profilo giornaliero (3 byte)

Si presentano una volta inviato un comando del tipo "Impostazione profilo giornaliero" e ne confermano l'avvenuta esecuzione.

byte 0: 02

byte 1: 02

byte 2: giorno di cui si è modificato il profilo

NOTA: i giorni della settimana vengono contati partendo da sabato ($00 = Sabato, \dots, 06 = Venerdì$).

Notifiche di richiesta profilo giornaliero (16 byte)

Si presentano una volta inviato un comando del tipo "Richiesta profilo giornaliero" e forniscono tutte le informazioni necessarie a individuare temperature e fasce orarie relative alla giornata selezionata.

byte 0: 21

byte 1: giorno della settimana

byte (2-15): una sequenza di *sette copie di byte* (XX,YY) in ognuna delle quali:

- YY rappresenta l'orario codificato come $(numero_minuti * 10)$ fino al quale verrà mantenuta la temperatura dichiarata in XX
- XX rappresenta la temperatura da mantenere fino a quel momento codificata come $(temperatura * 2)$

NOTA: l'intera sequenza permette di dedurre la temperatura da mantenere in ogni momento della giornata. Eventuali coppie di byte superflue assumeranno il valore $XX = temperatura_base$ e $YY = 0x90$. Inoltre, i giorni della settimana vengono contati partendo da sabato ($00 = Sabato, \dots, 06 = Venerdì$)

ESEMPIO

Il profilo della giornata di Lunedì è impostato alla temperatura base di 17°C. Questa viene posta a 21°C nelle due fasce 06:00-09:00 e 17:00-23:00. Richiedendone la lettura otteniamo la notifica: *21 02 22 24 2A 36 22 66 2A 8A 22 90 22 90 22 90*, costruita nel seguente modo:

byte 0: *21* (valore predefinito)

byte 1: *02* (lunedì = 0x02, come spiegato nella precedente nota)

byte (2,3): *22 24* (17 gradi fino alle 06:00)

byte (4,5): *2A 36* (21 gradi fino alle 09:00)

byte (6,7): *22 66* (17 gradi fino alle 17:00)

byte (8,9): *2A 8A* (21 gradi fino alle 23:00)

byte (10,11): *22 90* (17 gradi fino alle 24:00)

byte (12,13): *22 90* (superfluo)

byte (14,15): *22 90* (superfluo)

4 Realizzazione software

Il software realizzato si compone di una serie di funzioni utili all'interazione con una singola valvola affiancate da alcuni script per la gestione contemporanea di più dispositivi. Il tutto è stato sviluppato e testato su shell *Bash* e questo dovrebbe garantirne portabilità su buona parte dei sistemi *Unix* attraverso un numero pressoché nullo o abbastanza contenuto di modifiche. Tuttavia, la dipendenza dallo stack Bluetooth *BlueZ*¹ ne limita l'utilizzo ai sistemi operativi *GNU/Linux*.

4.1 Licenza GPL v3

Come anticipato nella sezione "Obiettivi e motivazioni", il progetto è tra le altre cose volto a fornire la possibilità di integrazione delle valvole termostatiche in impianti di domotica libera. Coerentemente con ciò, il codice prodotto verrà rilasciato con licenza *GPL v3* (GNU General Public License), le cui condizioni hanno lo scopo di garantire le quattro "libertà fondamentali" [6] definite dalla *Free Software Foundation*:

- libertà di eseguire il programma per qualsiasi scopo
- libertà di studiare il programma e modificarlo
- libertà di ridistribuire copie del programma
- libertà di migliorare il programma e di distribuirne pubblicamente i miglioramenti

In sostanza, si tratta di una licenza marcatamente *copyleft*: chiunque voglia distribuire copie di un software vincolato a tali condizioni, sia gratis che dietro il pagamento di un prezzo, è obbligato a riconoscere al ricevente gli stessi diritti che esso stesso ha ricevuto, garantendo inoltre l'accesso al codice sorgente.

¹www.bluez.org

4.2 Funzioni per la gestione di una singola valvola

Tutto il necessario alla gestione di una singola valvola è contenuto in due file. Il primo, *funzioni_base.sh*, si occupa dell'invio e ricezione di dati, della traduzione delle notifiche e contiene inoltre alcune funzioni di uso frequente. Il secondo, *comandi_valvola.sh*, gestisce la composizione a livello sintattico/semantico di ogni comando. Entrambi i file sono commentati e descrivono ogni funzione, pertanto di seguito ne verranno discussi solo alcuni aspetti rilevanti.

send_command() - *funzioni_base.sh*

input: $\langle \text{device_address} \rangle$ $\langle \text{command} \rangle$

output: valore della notifica ricevuta a seguito dell'esecuzione

Scrive il valore dell'argomento "*command*" sulla caratteristica identificata dall'handle *0x0411*, provocando così l'esecuzione del comando corrispondente. La trasmissione pone le sue basi nell'utilizzo del tool *Gatttool*, incluso nello stack Bluetooth *BlueZ*, attraverso la seguente linea di codice:

```
output=$(timeout $TIMEOUT_SEC gatttool -b $1 --char-write-req -a
0x0411 -n $2 --listen)
```

I parametri *-b* e *-n* permettono di specificare rispettivamente l'indirizzo del dispositivo e il valore da inviare; al momento dell'esecuzione verranno quindi sostituiti da $\langle \text{device_address} \rangle$ e $\langle \text{command} \rangle$. L'uso di *-listen* pone il tool in stato di blocco, in attesa di una o più notifiche dalla controparte. Tuttavia non vi è modo di indicare un termine temporale a questa condizione, rendendo necessario l'utilizzo di "*timeout \$TIMEOUT_SEC*" per memorizzare quanto ricevuto nella variabile *output* dopo un determinato arco di tempo e passare all'istruzione successiva. La variabile *TIMEOUT_SEC* è definita nel file *config.sh* e verrà approfondita nella sezione "5 - Test sul campo".

É interessante evidenziare che nonostante l'applicazione CalorBT richieda una procedura di pairing per poter dialogare con la valvola, questa non risulti effettivamente

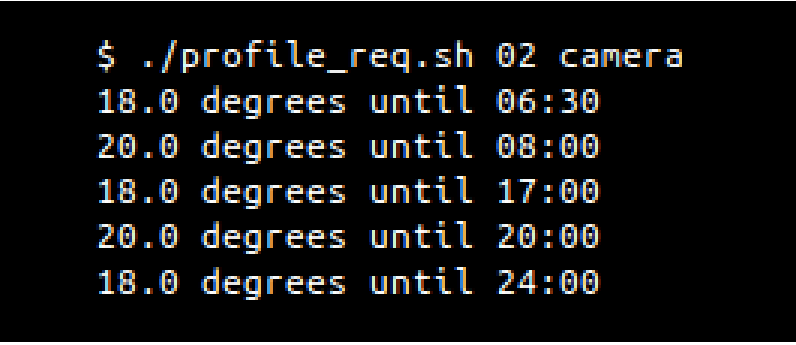
necessaria al di fuori del contesto Android/iOS. Si tratta chiaramente di una falla di sicurezza in quanto consente la comunicazione con il dispositivo pur conoscendone solo l'indirizzo MAC, peraltro facilmente ricavabile attraverso tool esterni quali *hcitool*, anch'esso incluso nello stack *BlueZ*.

parse_return_value() - *funzioni_base.sh*

input: $\langle \text{valore_notifica} \rangle$

output: notifica tradotta in contenuto "leggibile"

Traduce il contenuto della notifica e lo porta sullo *standard output* attraverso una serie di comandi *echo*. Il parametro $\langle \text{valore_notifica} \rangle$ dev'essere composto dai byte ricevuti, ognuno separato da uno spazio, sfruttando quindi lo stesso formato utilizzato da *gatttool*. La traduzione avviene interpretando il valore di ogni byte coerentemente con quanto riportato nella sezione "3.4 - Notifiche". Qualora si tratti di una notifica contenente informazioni sul termine della modalità vacanza, il parsing avviene tramite l'ausilio della funzione *parse_holiday_params()*, anch'essa contenuta nel file *funzioni_base.sh*, alla quale vengono forniti i byte 7, 8, 9 e 10.



```
$ ./profile_req.sh 02 camera
18.0 degrees until 06:30
20.0 degrees until 08:00
18.0 degrees until 17:00
20.0 degrees until 20:00
18.0 degrees until 24:00
```

Figura 8: Esempio funzionamento di *profile_req.sh*

A titolo esemplificativo, lo script *profile_req.sh* permette di richiedere ad una singola valvola informazioni sulla programmazione di una giornata tramite l'apposito comando. L'output prodotto è un chiaro esempio di utilizzo della funzione *parse_return_value()*. La figura 8 mostra la traduzione della notifica ricevuta a seguito

della richiesta del profilo relativo alla giornata di Martedì alla valvola denominata "camera".

calculate_temp() - *funzioni_base.sh*

input: temperatura in base 10

output: *temperatura * 2* in base 16 arrotondata

Permette di codificare la temperatura secondo il formato utilizzato dalla termovalvola. Il calcolo avviene attraverso la sua moltiplicazione per 2 e il successivo arrotondamento ad un valore pari a XX.0 o XX.5.

NOTA: il comando "Seleziona modalità vacanza" è l'unica eccezione. Esso richiede infatti che la temperatura sia codificata secondo la regola $(temperatura * 2) + 128$. A tal fine è stata predisposta la funzione *calculate_temp_128()*, anch'essa contenuta nel file *funzioni_base.sh*.

search_by_name() - *funzioni_base.sh*

input: <nome_valvola> <nome_file>

output: MAC address della valvola (*-1 se non esiste*)

Permette di reperire il MAC address di un singolo dispositivo, ognuno dei quali è identificato tramite un nome assegnato dall'utente all'interno del file <nome_file>. Ogni valvola dev'essere rappresentata su una nuova linea secondo il formato: *NAME/ADDRESS*. La funzione non effettua distinzioni tra lettere maiuscole e minuscole; di conseguenza, all'atto della ricerca, l'ipotetica linea "*valvola1/00:11:22:33:44:55*" è del tutto equivalente a "*VALVOLA1/00:11:22:33:44:55*". Il file viene scansionato riga per riga: in presenza di doppiati verrà selezionata la prima ricorrenza individuata.

File "comandi_valvola.sh"

Come già anticipato, il file *comandi_valvola.sh* gestisce la composizione a livello

sintattico di ogni possibile comando e ne richiede l'esecuzione attraverso le seguenti funzioni:

- **send_init()** $\langle \text{device_address} \rangle$
Invia data e l'orario attuali, automaticamente calcolati attraverso il comando *date*. Non è indispensabile, ma è utile *all'avvio della comunicazione* per garantire l'effettiva sincronizzazione tra dispositivo centrale e valvola e ricevere una notifica che ne riporta lo stato.
- **boost_mode()** $\langle \text{device_address} \rangle$
Provoca l'attivazione della modalità boost.
- **stop_boost_mode()** $\langle \text{device_address} \rangle$
Provoca la disattivazione della modalità boost.
- **auto_mode()** $\langle \text{device_address} \rangle$
Attiva la modalità automatica e ne regola la temperatura di conseguenza (secondo quanto impostato attraverso la programmazione settimanale).
- **manual_mode()** $\langle \text{device_address} \rangle$
Attiva la modalità manuale.
- **set_temperature()** $\langle \text{device_address} \rangle \langle \text{temperature} \rangle$
Imposta la temperatura al valore indicato dal secondo parametro.
- **set_comfort_reduction_temp()** $\langle \text{device_address} \rangle \langle \text{comfort_temp} \rangle \langle \text{reduction_temp} \rangle$
Modifica i valori della temperatura comfort e ridotta all'interno delle impostazioni della valvola.
- **holiday_mode()** $\langle \text{device_address} \rangle \langle \text{DD/MM/YYYY} \rangle \langle \text{hh:mm} \rangle \langle \text{temperature} \rangle$
Attiva la modalità vacanza, mantenendo la stessa temperatura fino al termine indicato dai parametri.
- **read_profile()** $\langle \text{device_address} \rangle \langle \text{day} \rangle$
Richiede il profilo relativo ad uno specifico giorno della settimana. I giorni vengono calcolati partendo da *00 (Sabato)* a *06 (Venerdì)*.

- **set_profile()** $\langle \text{device_address} \rangle \langle \text{day} \rangle \langle \text{int1} \rangle [\text{int2}] [\text{int3}] [\text{int4}] [\text{int5}] [\text{int6}] [\text{int7}]$
Imposta il profilo relativo ad un giorno della settimana. I giorni vengono calcolati partendo da *00 (Sabato)* a *06 (Venerdì)*. Ogni intervallo dev'essere nella forma *TEMP/hh:mm* e nel loro insieme devono garantire la copertura dell'intera giornata seguendo le linee guida riportare nella sezione "Impostazione profilo giornaliero". Di conseguenza gli intervalli 2-7 possono risultare non necessari e l'ultimo specificato riporterà l'orario *24:00* o *00:00*.
- **lock()** $\langle \text{device_address} \rangle$
Blocca i tasti fisici sulla valvola. Questo non ne impedisce l'interazione attraverso il Bluetooth.
- **unlock()** $\langle \text{device_address} \rangle$
Sblocca i tasti fisici sulla valvola.
- **set_window()** $\langle \text{device_address} \rangle \langle \text{temperature} \rangle \langle \text{duration} \rangle$
Imposta temperatura e durata della modalità finestra.
- **set_offset()** $\langle \text{device_address} \rangle \langle \text{temperature} \rangle$
Imposta la temperatura di offset. Dev'essere compresa tra *-3.5* e *+3.5*.

Tutte le funzioni presentate arrotondano automaticamente la temperatura inserita a valori del tipo *XX.0* o *XX.5*. Il loro funzionamento è pressoché identico e verrà illustrato attraverso il seguente esempio.

```

1  read_profile() {
2      if [ $2 -lt 0 -o $2 -gt 6 ]; then
3          echo "Week goes from 00 (saturday) to 06 (friday)."
4          return
5      fi
6
7      day=$(printf "%02x" $2)          # $2 = day
8      echo $( send_command $1 20$day ) # $1 = device_address
9  }
```

La prima parte, qui rappresentata dalle righe 2-7 ma non sempre necessaria, si occupa di effettuare controlli di correttezza degli input ed eventuali calcoli per la codifica dei parametri secondo il formato richiesto dalla valvola. La seconda parte (riga 8) invia il comando sfruttando la funzione *send_command()* vista precedentemente e ne stampa la notifica ricevuta.

Come si evince dall'esempio, il comportamento predefinito non prevede il parsing e la traduzione di quanto ricevuto. Affinché questo avvenga è necessario utilizzare il metodo *parse_return_value()*, sostituendolo al comando *echo* nell'ultima linea di ogni funzione (riga 8 nell'esempio) in questo modo:

```
parse_return_value $( send_command $1 20$day )
```

In alternativa è possibile ottenere lo stesso risultato spostando la chiamata di *parse_return_value()* all'esterno della funzione richiesta (*read_profile()* nell'esempio), ottenendo quindi il seguente codice:

```
parse_return_value $( read_profile ..... )
```

4.3 Script per la gestione di più valvole

Le funzioni descritte nella sezione precedente formano uno strumento particolarmente utile alla creazione di programmi volti ad automatizzare la gestione contemporanea di più dispositivi.

Verranno di seguito elencati gli script realizzati, i quali vogliono formare una semplice linea guida nello sviluppo di progetti più ampi. Il loro utilizzo, per semplicità d'uso, richiede di attribuire all'indirizzo MAC di ciascuna valvola un nome a scelta dell'utente. A tale fine è necessario indicare quale sarà il file contenente le associazioni *nome-indirizzo* effettuate, modificando opportunamente il contenuto della variabile *VALVE_FILE* all'interno di *config.sh*. Il suddetto file dovrà essere compilato secondo la sintassi richiesta dalla documentazione della funzione "*search_by_name()*", mantenendo quindi il formato *NOME/INDIRIZZO*. È possibile reperire gli indirizzi MAC

sfruttando il tool esterno *hcitool*, incluso nello stack *BlueZ*, attraverso il comando "*hcitool lescan*".

- **auto_mode** <nome_valvola> [nome_valvola ...]

Pone in modalità automatica tutte le valvole identificate dai nomi forniti da linea di comando. É richiesto l'inserimento di almeno un parametro, mentre i successivi sono opzionali.

NOTA: le temperature impostate su ogni valvola a seguito dell'esecuzione del comando sono *dipendenti* dalla programmazione di ognuna di queste.

- **manual_mode** <nome_valvola> [nome_valvola ...]

Pone in modalità manuale tutte le valvole identificate dai nomi forniti da linea di comando. É richiesto l'inserimento di almeno un parametro, mentre i successivi sono opzionali.

- **set_temperature** <temperature> <nome_valvola> [nome_valvola ...]

Imposta la temperatura rappresentata da <temperature>, dopo eventuali arrotondamenti, su tutte le valvole indicate dai nomi forniti da linea di comando. Richiede l'inserimento del primo e del secondo parametro, mentre i successivi sono opzionali.

- **set_all_temp** <temperature>

Imposta la temperatura rappresentata da <temperature> su tutte le valvole indicate dal file referenziato dalla variabile *\$VALVE_FILE*.

NOTA: il file viene scansionato riga per riga. La presenza di doppioni implica una doppia esecuzione del comando sulla stessa valvola.

- **set_profile** <profile_file> <nome_valvola> [nome_valvola ...]

Imposta i profili relativi a *una o più* giornate sulle valvole indicate tramite parametro, sfruttando il contenuto del file <profile_file>. Quest'ultimo deve contenere le istruzioni riguardanti la programmazione di uno o più giorni nel seguente formato:

```

giorno (1=lunedì, ..., 7=domenica)
temperatura base
HH:MM-HH:MM-TEMP      (primo intervallo)
HH:MM-HH:MM-TEMP      (secondo intervallo, opzionale)
HH:MM-HH:MM-TEMP      (terzo intervallo, opzionale)
end

```

Ad esempio, la programmazione del Lunedì con temperatura di base 18°C e 20°C nelle due fasce 06:30-08:00 e 17:00-20:00 viene effettuata in questo modo:

```

01
18
06:30-08:00-20
17:00-20:00-20
end

```

NOTA: all'interno dello stesso file è possibile specificare la programmazione di più giornate separando ogni blocco mediante una riga vuota.

- **profile_req** <giorno> <nome_valvola> [nome_valvola]

Richiede e stampa il profilo relativo ad una giornata per le valvole indicate dai nomi forniti da linea di comando. Il parametro <giorno> è calcolato come:

01 = Lunedì, ..., 07 = Domenica.

NOTA: la traduzione della notifica ricevuta, contenente le informazioni richieste, avviene come comportamento di default e utilizza la funzione già discussa "*parse_return_value()*".

Il principio di funzionamento di ogni script è pressoché identico. In primo luogo viene verificata la presenza dei parametri richiesti per l'esecuzione. Fatto ciò, ogni argomento rappresentante un dispositivo provoca l'esecuzione delle operazioni di

ricerca del relativo indirizzo MAC e l'invio del comando alla valvola. Le seguenti righe di codice, estratte dal file *"auto_mode.sh"*, pongono chiarezza rispetto alle funzioni utilizzate.

```
1  #!/bin/bash
2  . ./comandi_valvola.sh
3
4  if [ -z $1 ]; then
5      printf "Usage: ./auto_mode <nome_valvola> [nome_valvola ...]\n"
6      exit
7  fi
8
9  for name in "$@"; do
10     address=$( search_by_name $name $VALVE_FILE )
11     if [ "$address" != "-1" ]; then
12         auto_mode $address
13     else
14         printf "%s valve not found\n" "$name"
15     fi
16 done
```

L'inclusione di *comandi_valvola.sh* (riga 2) mette a disposizione tutte le primitive esposte nella sezione "4.2 - Funzioni per la gestione di una singola valvola". Questo provoca infatti l'implicita inclusione dei file *funzioni_base.sh* e *config.sh*, rendendo di fatto utilizzabili la funzione *search_by_name()* e la variabile *\$VALVE_FILE*. Le righe 4-7 verificano la presenza dei parametri richiesti e provocano la sospensione dell'esecuzione qualora non venga superato il controllo. Il ciclo *for* (righe 9-16) permette di scorrere tutti i nomi delle valvole forniti come argomento allo script, da ognuno dei quali, attraverso la funzione *search_by_name()*, vengono ricavati i relativi indirizzi MAC. Fatto ciò l'invio della richiesta alla valvola è immediato e si rimappa sulla chiamata a una funzione già nota (riga 12). Il file *comandi_valvola.sh* fornisce il necessario allo svolgimento di tutte le operazioni messe a disposizione dall'applicazione CalorBT.

La parte di codice che si occupa della trasmissione dei dati alla valvola si trova all'interno del ciclo *for*. Per questo motivo il comando richiesto viene inviato a *un dispositivo per volta*, nell'ordine in cui i nomi sono forniti allo script all'atto della sua invocazione. L'output prodotto corrisponde alle notifiche ricevute una volta eseguito ogni comando. Se l'indirizzo di una valvola non è reperibile in quanto non presente nel file referenziato da *\$VALVE_FILE*, il controllo di riga 11 provoca la stampa di un messaggio di errore (riga 14) e il passaggio all'eventuale ricerca del successivo indirizzo.

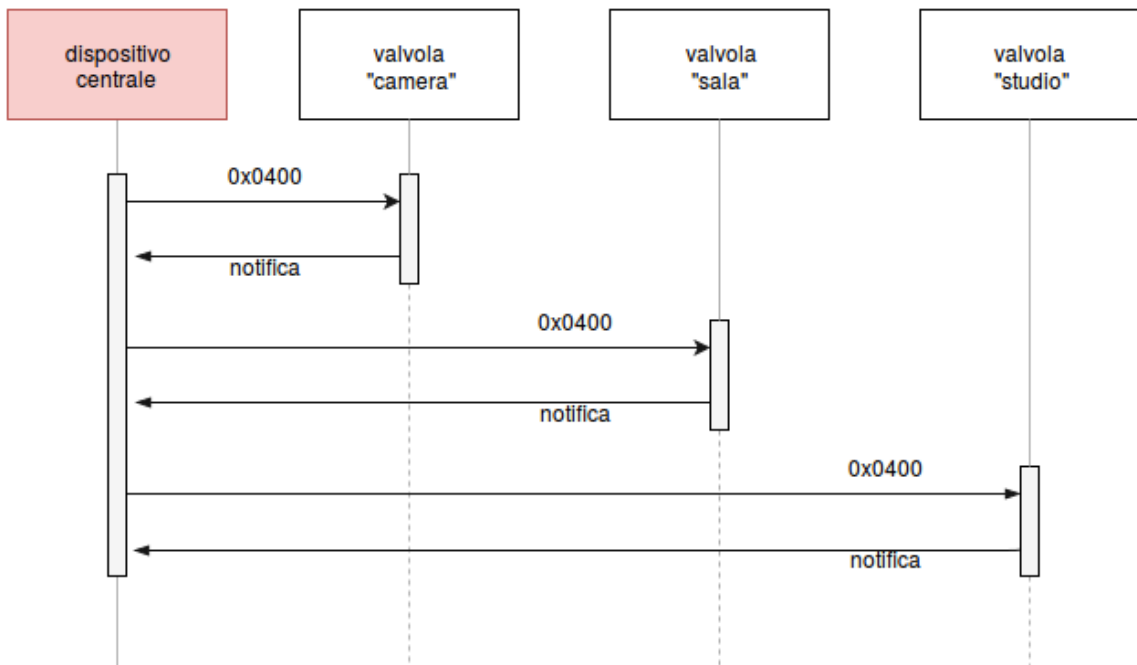


Figura 9: Sequence diagram dello script *auto_mode.sh*

Il *sequence diagram* [4] in figura 9 mostra lo scambio di messaggi tra valvole e dispositivo centrale durante l'esecuzione dello script sopra riportato, attivato nel seguente modo:

```
./auto_mode.sh camera sala studio
```

Coerentemente a quanto appena visto, l'invio del valore *0x0400*, richiesto per l'esecuzione del comando "Seleziona modalità automatica", avviene in successione per ogni valvola e solo dopo aver ricevuto la notifica relativa all'istruzione precedente.

Lo script *set_profile.sh* si presenta come unica eccezione a quanto descritto, a fronte della necessità di lettura e traduzione del file fornitogli. Quest'ultimo può contenere istruzioni relative alla programmazione di più giornate, rendendo necessario l'invio di *più comandi alla stessa valvola*. Per questo motivo si è scelto di minimizzare il numero di chiamate alla funzione *search_by_name()* effettuando a priori la ricerca degli indirizzi MAC richiesti e memorizzandone il risultato in un *array*, rappresentato dalla variabile *\$VALVES_ADDR* all'interno dello script.

5 Test sul campo

Quanto esposto nelle sezioni precedenti è stato testato utilizzando un adattatore Bluetooth 4.0 di *Classe 2* applicato a un dispositivo centrale dotato della distribuzione GNU/Linux *Ubuntu 14.04 LTS*. La versione installata dello stack *BlueZ* è la *4.101*, inclusa di default nel sistema operativo scelto, ma il funzionamento è stato verificato anche sulle versioni più recenti.

5.1 Portata e relative problematiche

L'azienda eQ-3, produttrice delle valvole prese in considerazione, dichiara una portata massima di 10 metri all'aperto. Nonostante questo l'utilizzo via smartphone, tramite l'apposita applicazione, ha permesso di raggiungere la distanza di *12,5 metri* senza presentare cali di ricezione.

Tabella 1: Test con adattatore Bluetooth di Classe 2

| DISTANZA | TEMPO MEDIO | DEV. STANDARD |
|----------|-------------|---------------|
| 2 metri | 02.99s | 0.44s |
| 4 metri | 03.78s | 0.59s |
| 6 metri | 13.73s | 1.80s |

La portata e la potenza del segnale influenzano direttamente il contenuto della variabile *\$TIMEOUT_SEC*, definita nel file *config.sh*. Essa è utilizzata nella funzione "send_command()" e il suo valore rappresenta pertanto il tempo massimo, espresso in secondi, entro il quale si ha la certezza di poter effettuare la connessione alla valvola, inviare un comando e ricevere la relativa notifica. Il tempo è impostato di default a *5 secondi* a fronte dei risultati ottenuti con l'adattatore di Classe 2 utilizzato nei test. Come si evince dalla tabella 1, quest'ultimo presenta evidenti perdite di segnale già alla distanza di 6 metri, rendendo impossibile effettuare analisi più approfondite. Si è perciò selezionato il valore più consono rispetto ai risultati ottenuti in condizioni di buona ricezione (2 e 4 metri), osservando che con

buona probabilità questi non subiranno forti incrementi sulle medio-lunghe distanze utilizzando un connettore Bluetooth più potente (Classe 1). Si tratta comunque di un parametro fondamentale che necessita di essere adattato a seconda del contesto applicativo delle valvole al fine di trovare un buon compromesso tra tempi di attesa ed errori.

Il calcolo del singolo tempo necessario alla connessione, l'invio di un'istruzione e la ricezione della relativa notifica è stato effettuato sfruttando il comando *time* e inviando alle valvole l'istruzione "Seleziona modalità automatica" nel seguente modo:

```
time gatttool -b $ADDRESS --char-write-req -a 0x0411 -n 4000
```

Le medie e le deviazioni standard sopra riportate risultano quindi dalla reiterazione di quest'istruzione su *tre diverse valvole* per un totale di *20 prove per ogni distanza*, garantendo sempre lo stesso stato di partenza (valvole disconnesse dal dispositivo centrale). La disconnessione avviene automaticamente 45 secondi dopo l'esecuzione di un comando.

5.2 Connessione parallela

Il tool *gatttool* utilizzato per la connessione non consente l'effettiva parallelizzazione nell'invio dei comandi; il dispositivo centrale è infatti in grado di comunicare con una sola periferica alla volta. Effettuata un'operazione di scrittura di una caratteristica, lo stesso *gatttool* non permette l'esecuzione di operazioni analoghe su altri dispositivi fino alla ricezione dell'effettiva conferma di successo, rendendo vano ogni tentativo di comunicazione contemporanea.

6 Sviluppi futuri e conclusioni

Per ragioni tempistiche e di complessità alcuni aspetti sono stati volontariamente trascurati all'interno del progetto. Si tratta per lo più di componenti inerenti gli script creati, che non compromettono la gestione delle valvole ma ne agevolerebbero comunque l'utilizzo.

Identificazione automatica delle valvole

Per facilitarne il riconoscimento, a ogni periferica Bluetooth è assegnato un nome non univoco: tutti i dispositivi modello *Equiva* prodotti dall'azienda *eQ-3* sono infatti identificabili attraverso la sigla "CC-RT-BLE". Sfruttando quest'osservazione e il comando *hcitool lescan* è quindi possibile individuare e memorizzare automaticamente gli indirizzi MAC di ogni valvola presente nell'ambiente circostante. Ciò consente la creazione di uno script che eviti all'utente la gestione manuale del file contenente le associazioni nomi/indirizzi discusso nella sezione "4.3 - Script per la gestione di più valvole". È inoltre possibile che l'applicazione *CalorBT* applichi, tra le altre cose, una metodologia simile all'atto della procedura di pairing. Questi aspetti sono analizzabili attraverso i metodi *scanForDevices()* e *onLeScan()* presenti nel file *de.eq3.ble.android.api.BluetoothAPI.java*.

Gestione degli errori

Tutte le funzioni incluse nel file *comandi_valvola.sh* e gli script creati partendo da queste presumono un corretto dialogo tra dispositivo centrale e valvole, trascurando eventuali malfunzionamenti. Risulta quindi di particolare utilità un gestore degli errori, la cui costruzione può basarsi sui tre possibili output prodotti. Il messaggio "*connect: No route to host*" indica che il Bluetooth è disattivato sul dispositivo centrale. La generazione di un output vuoto rappresenta un errore all'atto della connessione. Infine, la ricezione di una notifica in valori esadecimali è segnale di una corretta comunicazione.

Update del firmware

Il *firmware* è un programma integrato nella valvola necessario all'avvio del dispo-

sitivo stesso e alla comunicazione tra l'hardware e il software di cui è composto. Il suo aggiornamento è fondamentale per la correzione di eventuali errori divenuti noti in seguito al rilascio del prodotto. Nel caso preso in considerazione si tratta di un'operazione poco frequente e, per quanto essenziale, i comandi relativi all'aggiornamento non sono stati inclusi nel lavoro svolto. Gli aspetti implementativi sono tuttavia analizzabili attraverso il codice dell'applicazione osservando i file *StartFirmwareUpdateCommand.java* e *SendFirmwareDataCommand.java* inclusi nel package "de.eq3.ble.android.api.command" e tutti i metodi contenuti nel file *FirmwareUpdater.java* in "de.eq3.ble.android.api.util".

In conclusione, il lavoro presentato ha raggiunto gli obiettivi prefissati. L'attività di logging, affiancata all'analisi dell'applicazione Android, ha permesso di decifrare con buona precisione il protocollo utilizzato nella comunicazione e di porre una base sia dal punto di vista formale che a livello software per l'impiego di queste valvole termostatiche in lavori di più ampia dimensione e per l'integrazione delle stesse in impianti già esistenti. L'utilizzo di una licenza libera quale la GPL v3 intende infine essere una sorta di omaggio al concetto di "software libero" e un piccolo contributo ad un settore, quello dell'Internet Of Things, in continua evoluzione; il quale vede, accanto ad alcuni progetti liberi, il proliferare di dispositivi comunicanti attraverso protocolli proprietari.

Riferimenti bibliografici

- [1] Danea.it. Obbligo valvole termostatiche e condominio: la scadenza è sempre più vicina. www.danea.it/blog/obbligo-valvole-termostatiche-condominio/, 2015.
- [2] Tony DiCola. Reverse engineering a bluetooth low energy light bulb. learn.adafruit.com/reverse-engineering-a-bluetooth-low-energy-light-bulb, 2015.
- [3] Wikipedia - The Free Encyclopedia. Backus–Naur Form. https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form, 2016.
- [4] Wikipedia - The Free Encyclopedia. Sequence diagram. https://en.wikipedia.org/wiki/Sequence_diagram, 2016.
- [5] Gazzetta Ufficiale Dell'Unione Europea. Direttiva 2012/27/UE. <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2012:315:0001:0056:it:PDF>, 2012.
- [6] The Free Software Foundation. What is free software? www.gnu.org/philosophy/free-sw.html, 2016.
- [7] Arch. Carmen Granata. Obbligo di installazione delle valvole termostatiche. www.lavorincasa.it/obbligo-installazione-valvole-termostatiche/, 2015.
- [8] Tulipano Impianti. Impianto centralizzato con contabilizzazione del calore e termovalvole. www.tulipanoimpianti.it/home/blog/.
- [9] ISO/IEC. Extended Backus–Naur Form. <http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf>, 1996.
- [10] Bluetooth SIG. Bluetooth adopted specifications. www.bluetooth.com/specifications/adopted-specifications.

- [11] Bluetooth SIG. Bluetooth core specification. www.bluetooth.com/specifications/bluetooth-core-specification.
- [12] Mike Stirling. Fht8v protocol. www.mike-stirling.com/notes/fht8v-protocol, 2015.
- [13] Kevin Townsend. Introduction to bluetooth low energy. A basic overview of key concepts for BLE. <https://learn.adafruit.com/introduction-to-bluetooth-low-energy/>.
- [14] Gazzetta Ufficiale. Decreto 04/07/2014 di attuazione della direttiva 2012/27/UE. <http://www.gazzettaufficiale.it/eli/id/2014/07/18/14G00113/sg>, 2014.
- [15] Impresa Zambelli. Impianto centralizzato senza contabilizzazione del calore. www.impresazambelli.it/images/energetico1.jpg.