POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Progettazione e sviluppo di un'architettura a microservizi per la raccolta e l'analisi di dati di mobilità e di affollamento di edifici

Relatore
prof. Giovanni Malnati

Candidato
Erik Levi
matricola: 239794

Anno accademico 2018-2019

Abstract

Negli ultimi anni si è assistito ad una sempre maggiore diffusione di device personali, principalmente smartphone, tablet e laptop che integrano capacità di comunicazione wireless. I sistemi che permettono i meccanismi di discovery delle reti WiFi espongono questi device a potenziali rischi legati alla privacy ed alla sicurezza, principalmente a causa della presenza di informazioni, come il MAC address del device, che ne permettono l'identificazione in maniera univoca e quindi il tracciamento. È infatti possibile scoprire molte informazioni sui device e sui proprietari di essi semplicemente intercettando questi segnali, in maniera completamente passiva e difficilmente rilevabile.
Nel corso degli anni, per cercare di proteggere più efficacemente la privacy dei loro utenti, i produttori di smartphone hanno implementato sistemi di mascheramento di questi dati sensibili e ridotto le informazioni diffuse in maniere non sicura.
Questa tesi nasce dall'idea di sfruttare queste informazioni per creare un sistema che permetta il monitoraggio e l'analisi dell'affollamento di zone di edifici e dei movimenti delle persone all'interno di essi con lo scopo di poter fornire uno strumento atto all'individuazione e alla risoluzione di situazioni di criticità.
È stata analizzata la possibilità di utilizzare queste informazioni per ricavare stime sul numero di presenze e sulla distribuzione di persone all'interno di edifici e la possibilità di tracciare gli spostamenti di uno specifico device.
In questa fase è stata ricercata una metodologia che permetta di identificare in maniera univoca un device che implementa sistemi di mascheramento delle proprie informazioni, riuscendo così a tracciare anche questi.
Per la raccolta dei dati è stato necessario sviluppare un firmware custom per dei dispositivi IoT basati sulla piattaforma ESP32, dotati di funzionalità WiFi.
Dal lato back end è stata realizzata un'infrastruttura secondo lo stile a microservizi che permette la gestione dei dispositivi di raccolta dei dati, il salvataggio dei dati raccolti e l'analisi, sia in modalità on demand che tramite processi automatici.
Infine è stato realizzato un front end, usando il framework Angular, che permette l'interazione con i componenti dell'infrastruttura e la visualizzazione di statistiche e report.
Per quanto riguarda le capacità di tracciamento dei device, i risultati ottenuti mostrano che attualmente non risulta possibile identificare in maniera certa device che utilizzano forme di mascheramento dell'indirizzo MAC, mentre risulta triviale per quelli che non implementano questi meccanismi.
È stato possibile evidenziare come molti smartphone, anche di ultima generazione, non implementano queste funzionalità e pertanto risultano tracciabili.
Sono stati evidenziati risultati migliori per quanto riguarda la capacità di conteggio dei device presenti in un certo lasso di tempo nei pressi dei dispositivi di raccolta. Tuttavia è importante sottolineare la difficoltà di verifica di questi risultati in quanto si è rivelato molto difficile ottenere valori di riferimento su cui basarsi per validare i modelli.
Infine si è tentato di creare un sistema che permettesse visivamente di individuare le aree più affollate di un certo edificio sfruttando un approccio basato su mappe di calore che per il momento necessita di ulteriore messa a punto.

Ringraziamenti

Si ringrazia il professor Malnati e tutto lo staff di Tonic Minds per aver messo a disposizione il materiale e le strutture necessarie allo svolgimento di questa tesi.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Introduzione

Negli ultimi anni si è assistito ad una sempre maggiore diffusione di device personali, principalmente smartphone, tablet e laptop che integrano molteplici sistemi di comunicazione wireless. I meccanismi di discovery delle reti WiFi espongono questi device a potenziali rischi legati alla privacy, principalmente a causa delle informazioni presenti all'interno dei frame di Management di tipo Probe Request, come il MAC address del device, che ne permettono l'identificazione in maniera univoca. E' infatti possibile scoprire molte informazioni sui device e sui proprietari di essi semplicemente intercettando questi frame, in maniera completamente passiva e difficilmente rilevabile.
Nel corso degli anni, per cercare di proteggere più efficacemente la privacy dei loro utenti, i produttori di smartphone hanno implementato sistemi di mascheramento di questi dati sensibili, utilizzando indirizzi MAC randomizzati al posto di quello effettivo e ridotto al minimo le informazioni diffuse tramite questo tipo di pacchetti. Questi sistemi di anonimizzazioni, quest'oggi, vengono imlementati nella maggior parte dei nuovi smartphone, sia con sistema operativo IOS che Android.
Dal momento che questi dati vengono emessi dai device in maniera automatica possono essere sfruttati per tracciare i device o realizzare conteggi e statistiche per comprendere flussi e movimenti di persone.

1.2 Scopo della tesi

Lo scopo principale di questa tesi è l'implementare un infrastruttura che permetta la raccolta, la memorizzazione e l'analisi dei dati contenuti in questi frame. L'architattura da implementare deve essere in grado di gestire un flusso di dati potenzialmente di grandi dimensioni: basti pensare all'utilizzo di un sistema del genere all'interno di un università come il Politecnico di Torino, dove ogni giorno transitano migliaia di persone. L'infrastruttura deve essere scalabile in modo tale da supportare eventuali picchi di carico e deve permettere di implementare un sistema di raccolta dati adattabile e dinamico che riesca facilmente a rispondere alla necessaità di riconfigurazioni e cambiamenti. Inoltre è necessario sviluppare un interfaccia che permetta di visualizzare ed effettuare analisi su questi dati.

I dati raccolti potranno essere interpretati ed aggregati per permettere di realizzare stime sul numero di device (e quindi indirettamente di persone) localizzati all'interno di un certo spazio, la loro distribuzione all'interno degli spazi stessi e la tracciatura degli spostamenti.

1.3 Esempi di utilizzo

Le applicazioni di una tecnologia del genere sono molteplici e possono essere applicate in vari campi.
L'idea alla base della tesi nasce con l'intento di fornire uno strumento di monitoraggio della congestione degli ambienti, come nel caso dei corridoi del Politecnico che, a causa di un sempre maggior numero di alunni, risultano in certi orari difficilmente percorribili, o delle aule studio. Ad esempio uno studente, dovendo raggiungere una certa zona del Politecnico potrebbe ,tramite un'apposita applicazione, ricevere suggerimenti su quello che potrebbe essere il percorso più veloce o che evita zone particolarmente congestionate, oppure individuare l'aula studio meno affollata.
Ovviamente l'idea non è nuova, e nel mondo sono già stati messi in funzione sistemi che sfruttano questi tipi di dati per fare analisi del genere.
Recentemente [4] la società che gestisce la metropolitana di Londra, ha messo in funzione un sistema di monitoraggio basato sulla raccolta dei dati Wi-Fi degli utilizzatori della metropolitana. Nonostante la società avesse visibilità sul punto di ingresso e di uscita degli utenti grazie ai dati raccolti dai tornelli, non era in grado di identificare il percorso effettivamente seguito dell'utente all'interno della rete della metropolitana. Il sistema, tuttavia, raccoglie solamente i dati degli utenti che si collegano volontariamente alla rete della metropolitana, rivelando quindi il loro effettivo indirizzo MAC e quindi permettendo la tracciabilità del device.
Questo sistema è nato in seguito ad un periodo di test effettuato alla fine del 2016 [5] che ha permesso di individuare le potenzialità dell'utilizzo di uno strumento del genere. Ovviamente, oltre allo studio degli spostamenti dei passeggeri tra le varie stazioni della rete metropolitana, l'interesse della società è individuare le zone delle stazioni in cui si accumulano più viaggiatori in modo da poter fornire alle aziende pubblicitarie un'indicazione quantitativa sulla visibilità dei loro annunci.
Uno strumento del genere può anche essere utilizzato per studiare le abitudini dei consumatori all'interno di spazi come supermercati o centri commerciali per determinare i percorsi effettuati, quali aree sono soggette ad un maggiore flusso di persone ed a che ora.

1.4 Background

Di seguito verranno descritti alcuni concetti propedeutici alla lettura del resto del documento.

1.4.1 Mac address e livello collegamento

L'indirizzo MAC (Media Access Control) è un codice di 48 bit (6 bytes) che viene assegnato in maniera univoca a qualsiasi dispositivo di rete e wireless e viene usato per identificare il dispositivo a livello di rete locale. Solitamente viene rappresentato come una stringa di dodici caratteri esadecimali suddivisi a due a due da ":". I primi tre byte, corrispondenti

ai primi 6 caratteri vengono chiamati OUI (Organisation Unique Identifier) identificano il produttore del dispositivo e vengono assegnati dall IEEE (Institute of Electrical and Electronics Engineers) mentre i successivi, identificati come NIC (Network Interface Controller) vengono assegnati in maniera sequenziale al momento della produzione del dispositivo. In questo modo, conoscendo l'OUI di un dispositivo è possibile risalire al produttore del dispositivo attraverso apposite tabelle di lookup. Gli indirizzi assegnati direttamente dal produttore vengono chiamati universally administered e, come detto in precedenza, non possono esistere due dispositivi con lo stesso indirizzo. Tuttavia questo indirizzo, per quanto imposto dal produttore del dispositivo, può essere modificato. Secondo lo standard, questi indirizzi modificati vengono chiamati locally administered. Ovviamente nel caso una stazione dovesse usare un indirizzo di questo tipo, l'assegnazione deve essere fatta con un indirizzo non assegnato. Per distinguere indirizzi locally administered da quelli globally administered, viene appositamente settato il penultimo bit meno significativo del byte più significativo, 0 quando l'indirizzo è globally administered e 1 altrimenti (figura 1.1). Pertanto, i device che effettuano un mascheramento dell'effettivo indirizzo MAC tramite randomizzazione, devono rispettare questo vincolo.

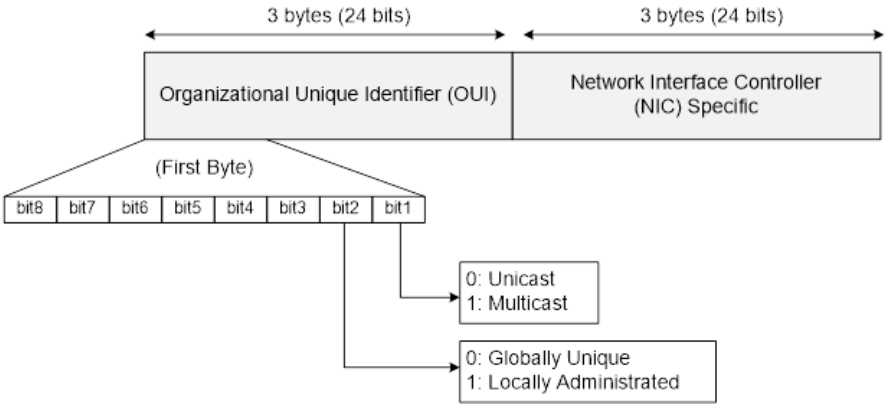


Figura 1.1: Struttura di un indirizzo MAC

1.4.2 I tipi di frame del protocollo 802.11

802.11 [2] è lo standard IEEE che descrive l'insieme di specifiche necessarie all'implementazione di reti LAN tramite tecnologia wireless, comunemente nota come Wi-Fi. A livello collegamento, secondo lo standard OSI, possiamo individuare tre tipi di pacchetto, chiamati frame, che vengono utilizzati per implementare le varie funzionalità del protocollo. I frames di tipo Data sono ovviamente utilizzati per il trasporto di dati.
I frames di tipo Control servono ad implementare meccanismi di scambio di informazioni tra stazioni e sono i frame di Acknowledgement usati per indicare la corretta ricezione di un Data Frame, Request to Send e Clear to Send che permettono di implementare sistemi opzionali di riduzione delle collisioni in casi particolari.
I frame di tipo Management sono utilizzati per la gestione e l'interruzione delle comunicazioni.

Questi frame vengono utilizzati per la gestione della connessione e disconnessione delle stazioni e anche per i meccanismi atti alla scoperta degli access point disponibili.

1.4.3 Sistemi di discovery delle reti WiFi

La scoperta delle reti WiFi utilizzabili in una certa aree avviene in due modalità.
La prima è operata dagli access point e prevede l'invio di di un frame di tipo beacon che contiene le informazioni necessarie per effettuare il collegamento, come il SSID (Service Set Identifier, il nome con cui è identificata la rete). I device in ascolto potranno entrare a conoscenza delle reti disponibili nelle vicinanze in modo totalmente passivo in quanto l'invio di questi dati non è avvenuto in seguito ad una richiesta del device.
La seconda modalità prevede che siano i device stessi a richiedere agli access point in ascolto di annunciarsi. Il device che da inizio alla comunicazione invia un pacchetto broadcast di tipo probe request (1.2) a cui segue un messaggio di tipo probe response inviato dagli access point presenti nelle vicinanze.

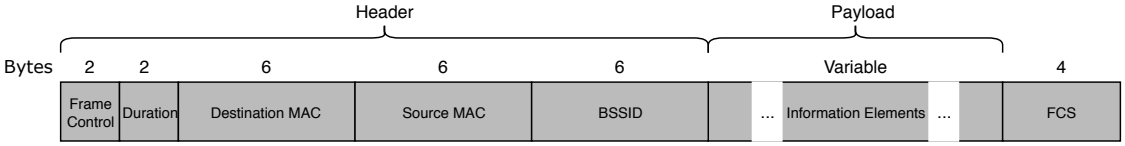


Figura 1.2: Probe Request frame

Il frame probe request è composto da diversi campi di lunghezza fissa e altri di variabili chiamati information elements. I campi fissi sono i seguenti:

- Frame Control: permette di identificare il tipo di frame, in questo caso è presente il valore 0x04 in esadecimale.
- Duration
- Destination MAC: l'indirizzo MAC del destinatario, broadcast nel caso di probe request.
- Source MAC: l'indirizzo mac del device
- BSSID
- Information Elements: anche noti come tagged parameters, sono dati aggiuntivi che vengono inseriti nel pacchetto probe request per indicare funzionalità del dispositivo o parametri utili alla connessione come velocità di trasmissione supportate

Questa seconda opzione è preferita rispetto alla prima in quanto più efficiente dal punto di vista energetico, aspetto determinante soprattutto in campo mobile. Infatti il device dovrà lasciare l'antenna accesa in ricezione solo il tempo necessario all'arrivo di un frame probe response piuttosto che dover attendere un beacon frame per un tempo indefinito.
Solitamente i probe request sono emessi raggruppati in gruppi di numero variabile all'interno di una finestra temporale solitamente inferiore al secondo. Quest'emissione prende il nome di *burst*. All'interno di un burst, nel caso il device effettui il maschermento dell'indirizzo MAC, quello utilizzato sarà consistente per l'intera durata del burst.

1.4.4 Gli Information Elements

I frame probe request contengono informazioni aggiuntive che prendono il nome di Information Elements o anche tagged parameters e vengono usati per annunciare parametri relativi alle funzionalità del dispositivo. Si presentano nel formato tag-lunghezza-contenuto dove tag e lunghezza sono un byte.
La maggior parte dei tag sono opzionali e non vengono inclusi all'interno dei probe request. Nella seguente lista vengono riportati i tagged parameters più comuni (per ogni tag viene riportato il valore in esadecimale):

- 00 TAG_SSID : contiene l'eventuale SSID
- 01 TAG_SUPP_RATES: indica la lista di velocità supportate er lo scambio di dati
- 03 TAG_DS_PARAMETER: indica il canale usato per la trasmissione
- 2d TAG_HT_CAPABILITY: introdotti dallo standard 802.11n, annuncia le capacità di high troughput del device
- 32 TAG_EXT_SUPP_RATES: estende il tag 01 in caso siano supportate più di 8 velocità
- 6b TAG_INTERWORKING
- 72 TAG_MESH_ID
- 7f TAG_EXTENDED_CAPABILITIES: il contenuto varia in base alle funzionalità implementate dal device
- bf TAG_VHT_CAPABILITY: introdotti dallo standard 802.11ac, annuncia le capacità di very high troughput del device
- dd TAG_VENDOR_SPECIFIC_IE: tag che individuano funzioni implementate da secifici produttori
- ff TAG_ELEMENT_ID_EXTENSION

Nella figura 7.3 è possibile vedere la distribuzione dei tag all'interno dei dataset utilizzati in fase di test.
Il contenuto o la presenza o meno di questi tag è caratteristica dell'hardware e del software utilizzati dal dispositivo.

1.5 Problematiche evidenziate

Per quanto tracciare e conteggiare i dispositivi che utilizzano il loro indirizzo globally administered risulti banale, quanto descritto in precedenza determina alcune problematiche che dovranno essere indirizzate dalla soluzione sviluppata: è possibile individuare una metodologia in grado di de anonimizzare i dispositivi che mascherano il loro indirizzo MAC in modo da poterli tracciare e conteggiare?
Nel prossimo capitolo verranno analizzate alcune metodologie presenti in letteratura che possono essere compatibili con il contesto della tesi.

1.6 Chiarimento sulla nomenclatura utilizzata e alcune scelte stilistiche

Nel seguito verranno usati i termini *pacchetti global* e *pacchetti local* così come *indirizzi global* ed *indirizzi local* per intendere rispettivamente pacchetti probe request contenenti un indirizzo MAC del mittente globally o locally administered e indirizzi globally o locally administered. Per quanto riguarda le scelte stilistiche, si è evitato di riportare parametri e tipi di ritorno delle funzioni per non rendere il testo poco comprensibile. I nome delle funzioni sono eviedenziati nel seguente modo `funzione()` in modo da distinguerle dalle `variabili`, che non presentano le parentesi.

Capitolo 2

Lavori correlati

In questo capitolo verrà effettuata una rassegna degli studi effettuati sull'argomento di interesse per questa tesi, che sono stati utilizzati per individuare possibili tecniche e metodologie utili al conseguimento degli obiettivi preposti.

2.1 Osservazioni sul comportamento dei dispositivi mobili

Uno dei primi punti da chiarire è come avviene l'emissione di probe request da parte dei device. In letteratura sono presenti diversi lavori che hanno cercato di stabilire quali sono i criteri che influenzano l'emissione di frame probe request.
Freudiger [12] ha analizzato come diversi fattori impattano sull'emissione di probe request. In part icolare ha evidenziato come lo stato di funzionamento del device (in uso, in carica, connesso ad una rete) influenzano pesantemente l'emissione di probe request. Dai risultati ottenuti si nota come il numero di probe request emessi è molto maggiore durante lo sblocco dello schermo del device e durante l'utilizzo dell'app di configurazione del WiFi. La connessione ad una rete, invece, riduce drasticamente il numero di probe request emessi, nel caso in esame, soprattutto in smartphone Android. Le osservazioni effettuate attestano che la maggior parte dei device analizzati emette probe request mediamente una volta ogni minuto. Si evidenzia inoltre che i device Apple tendono ad inviare molti meno probe request rispetto ad Android.
Altri studi hanno analizzato come avviene l'emissione di probe request: Lim et al. [14] hanno analizzato la frequeza di emissione di probe request all'interno di alcuni dataset pubblici e ha scoperto come l'85% dei device ha l'80% di possibilità di emettere un probe request in una finestra temporale di 3 minuti.
Questi risultati, per quanto interessanti, sono stati ottenuti analizzando dataset vecchi e potrebbero non essere più attendibili.
Matte [13] ha analizzato il comportamento di diversi device, sia dispositivi con sistema operativo Android che dispositivi Apple. Dalle osservazioni effettuate emerge che i dispositivi Android tendono ad avere comportamenti molto variabili, in base al modello del dispositivo: in alcuni casi l'emissione di probe request avviene in maniera regolare, in altri varia in base all'utilizzo o al fatto che il device sia connesso ad una rete. I tempi che trascorrono tra l'emissione consecutiva di due burst sono comparabili a quelli descritti in precedenza.

2.2 Sistemi di mascheramento del Mac Adress

In letteratura è possibile trovare diversi articoli che analizzano gli schemi utilizzati per il mascheramento degli indirizza MAC tramite varie metodologie, principalmente legate alla randomizzazione degli indirizzi.
Vanhoef et al. [11] hanno descritto le diverse metodologie di randomizzazione utilizzate nei device più diffusi. Per i device che Apple, la randomizzazione dei MAC address è stat introdotta dalla versione 8 di iOS e a partire dalla versione 9 viene utilizzata per ogni probe request.
I device che usano il sistema operativo Android possono utilizzare la randomizzazione a partire dalla versione 6, sempre che l'hardware supporti questa funzionalità. Martin et al.[15] hanno approfondito gli schemi di randomizzazione usati arrivando a determinare che i device Android che utilizzano randomizzazione, utilizzano principalmente come OUI `DA:A1:19` che è registrato da Google e randomizzano i tre byte successivi. Analisi sulla randomizzazione degli indirizzi usati da device Apple indicano invece che, esclusi gli ultimi due bit del MAC address (il penultimo settato ad 1 in quanto locally administered e l'ultimo settato a zero) tutti gli altri vengono randomizzati anche se l'OUI utilizzato non è registrato presso l'IEEE. Inoltre, a partire da iOS 10, è stato introdotto un Information Element vendor specific che permette di identificare in maniera certa dispositivi Apple.

2.3 Sistemi di device fingerprinting

Molti lavori hanno cercato di trovare tecniche e metodoligie in grado di de anonimizzare i device che implementano meccanismi di mascheramento del MAC adress. Le tecniche utilizzate spaziano da analisi dei dati trasmessi a livello fisico, a sofisticati attacchi che forzano il device a rivelare il proprio MAC address.
Matte [13] ha evidenziato diversi metodi che possono essere usati per tentare di deanonimizzare un device basati sull'utilizzo dei frame probe request.
Una prima tecnica è basata sull'utilizzo del contenuto dei Information Elements come un fingerprint per identificare in maniera univoca un device. Questo approccio si basa sul lavoro di Gentry e Pennarum [16] che propongono l'utilizzo del contenuto dei frame probe request per generare un impronta che permetta di identificare uno specifico modello di device.
Una seconda metodologia si basa sull'analisi dei tempi con cui vengono emessi i probe request, cioè l'intervallo che intercorre tra l'emissione dei due pacchetti consecutivi appartenenti allo stesso burst e la durata del burst stesso in termine di numero di pacchetti. Entrambe queste metodologie tuttavia hanno il problema di non essere in grado di individuare in modo univoco un device, ma piuttosto il modello. Infatti i dati considerati sono specifici di un modello o comunque di una classe di modelli che condividono simili implementazioni hardware e software. Nel lavoro viene anche accennato il possibile utilizzo dei SSID annunciati all'interno dei probe request come metodo di identificazione di un device in quanto potrebbero essere annunciati SSID univocamente associabili ad un numero ristretto di device.
Infine viene descritta una procedura che permette di derivare, per alcuni device, il MAC address del dispositivo a partire dal contenuto del tag WPS.

Capitolo 3

Architettura generale

Di seguito verrà descritto il processo di design ed implementazione dell'architattura che risponde ai criteri descritti precedentemente.

3.1 Soluzione proposta

Nello schema 3.1 è descritto lo schema generale dell'architettura che è stata sviluppata per effettuare la raccolta e l'analisi dei pacchetti probe request catturati.

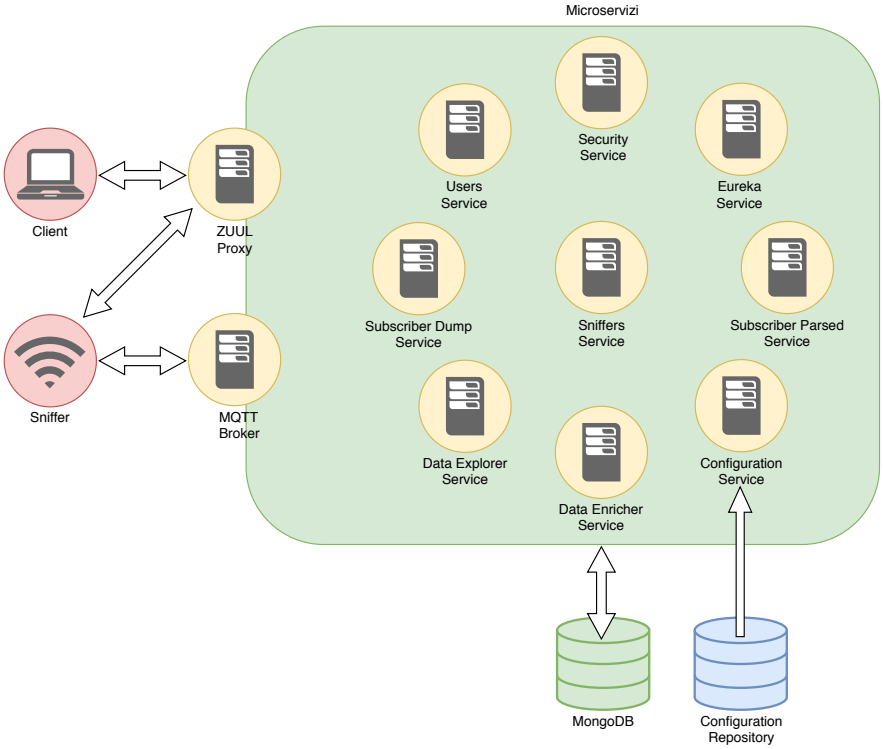


Figura 3.1: Schema generale dell'architettura sviluppata

3.1.1 Architettura generale

Dallo schema è possibile individuare i diversi attori che compongono l'architattura realizzata. Gli sniffer sono gli strumenti impiegati per raccogliere i pacchetti probe request. Al momento dell'avvio gli sniffer si collegano ad una rete wifi, reperiscono la loro configurazione dopo essersi autenticati con l'apposito servizio e iniziano la raccolta dei dati in base alle impostazioni. La comunicazione tra gli sniffer e gli appositi servizi del back end viene effettuato tramite il protocollo MQTT. A questo punto i dati vengono ulteriormente elaborati e salvati in un database per le future analisi.
Questi componenti sono stati realizzati attraverso dispositivi IoT con capacità WiFi facenti parte della famiglia ESP32.
L'aspetto interessante derivato dall'utilizzo di questi dispositivi è il loro costo molto basso che rende l'implementazione di questa architattura poco onerosa in termini finanziari. L'unica necessità è rappresentata dal bisogno di una rete WiFi esistente per permettere la connessione degli sniffer alla rete per l'invio dei dati raccolti.
Il back end è stato realizzato secondo i criteri delle architetture a micro servizi, che prevede la creazione di vari servizi che gestiscono rispettivamente una precisa funzione all'interno dell'applicativo e risultano tra loro indipendenti per poter permettere una maggiore flessibilità.
I servizi sono stati realizzati con il framework Java Spring Boot, che rappresenta lo standard per la realizzazione di applicativi web.
I servizi sviluppati possono essere raggruppati in tre gruppi.
Un primo gruppo è quello che implementa la raccolta e l'elaborazione dei dati provenienti dagli sniffer. Questi elementi sono il servizio MQTT, ed i due servizi Subscriber Dump e Subscriber Parsed che hanno il compito di raccogliere i dati inviati dagli sniffer.
Un secondo gruppo è costituito da Security Service, User Service e Sniffer Service. Questi tre servizi implementano quella che è la logica di gestione degli utenti e degli sniffer e permettono di implementare meccanismi di autorizzazione, autenticazione, configurazione degli sniffer e di creazione, modifica, lettura e cancellazione delle informazioni salvate nel database.
Un terzo gruppo è formato da Data Enricher Service e Data Explorer Service che implementano i sistemi di aggregazione e di analisi dei dati.
Per visualizzare i dati raccolti in formato aggregato e modificare le impostazioni dei vari servizi è stato realizzato un front end utilizzando il framework Angular che nello schema è rappresentato dal client.
Come accennato in precedenza, il protocollo prescelto per l'invio dei dati raccolti dagli sniffer è MQTT, pertanto sarà necessario realizzare un apposito servizio a livello di back end in grado di utilizzare questo protocollo. La comunicazione tra il client ed i vari servizi del back end e tra gli sniffer ed il back end al momento della configurazione sarà gestita tramite richieste HTTP secondo il paradigma REST. Dato che REST prevede servizi di back end stateless, sarà necessario implementare un sistema di autenticazione ed utorizzazione compatibile con queste condizioni. Per questo è stato scelto di implementare il back end Oauth2. Dal momento che il deploy e la gestione di un'architettura del genere può rivelarsi piuttosto ostica, si è deciso di eseguire ogni servizio all'interno di un container Docker, in modo da permettere una più facile gestione e configurazione.
Questi aspetti verranno ulteriormente approfonditi nel corso del capitolo.

3.1.2 Metodologia di calcolo dei fingerprint

La ricerca in letteratura non ha portato soluzioni risolutive: la maggior parte dei sistemi di fingerpinting basati sugli information elements e sui frame Probe Request, infatti, non è utilizzabile per effettuare il tracciamento di un device a lungo termine dato che i dati estraibili sono specifici di un modello e non di un singolo device. Più in generale, al momento non è stata trovata una tecnica che permetta la deanonimizzazione certa di ogni tipo di device, che non preveda l'utilizzo di attacchi mirati o tecniche non attuabili con i mezzi a disposizione.
Inoltre la tecnica di derivazione del MAC address di un dispositivo utilizzando il contenuto del tag WPS si è rivelata inutilizzabile in quanto i frame probe request provenienti da device che mascherano il proprio indirizzo MAC, dalle osservazioni effettuate, non contengono più questo tag. Infatti, i produttori di smartphone, hanno tenuto in considerazione le conclusioni tratte dai lavori citati nel capitolo precedente ed hanno preso provvedimenti.
La scelta è ricaduta sulla metodologia di fingerprinting basata sugli Information Element. In questo modo per ogni pacchetto ricevuto sarà creato un fingeprint e non sarà necessario mantenere un modello predittivo basato su clustering come previsto da altre metodologie. Pertanto la tecnica di fingerprinting verrà utilizzata per lo meno per fornire una stima più plausibile del numero di device che randomizzano il proprio MAC address, che altrimenti porterebbero ad una sovrastima del numero dei device in quanto, ad ogni emissione di frame Probe Request, verrebbero considerati un device diverso.
Questi aspetti verranno approfonditi nel capitolo 7.
La metodologia ha subito varie iterazioni nel tentativo di ottenere un errore di conteggio minore possibile. Attualmente il fingerpint viene calcolato nel seguente modo:

MD5(a + b + c)

- a = lunghezza del payload del pacchetto probe request al netto di eventuali SSID
- b = elenco ordinato dei tagged parameters utilizzati
- c = contenuto dei tag DD, 01, 32, 7F, 2D, BF

La scelta dei parametri è dovuta al fatto che le osservazioni effettuate in una fase preliminare, effettuate tramite l'utilizzo di Wireshark o script Python, hanno evidenziato come la lunghezza dei frame probe request osservati variava molto. A tale informazione si è aggiunto anche l'elenco ordinato dei tag presenti come Information Element in modo da poter distinguere ulteriormente i device. Infine è stato aggiunto il contenuto di alcuni tag selezionati che dovrebbero andare a massimizzare la differenziazione tra i device.
Tuttavia, effettuando analisi sui pacchetti global, è stato osservato che i valori di questi tag non è stabile nel tempo quindi un singolo device è individuato da un insieme di fingerprint. Questo ovviamente va a rendere ancora più difficoltoso l'utilizzo di questa tecnica per il tracciamento dei dispositivi ma non sembra essere molto influente per quanto riguarda il conteggio.

3.2 Le architetture a microservizi

Per la realizzazione del back end si è deciso di optare per una architettura a microservizi. L'utilizzo di questo stile architatturale ci permette di:

- realizzare un infrastruttura altamente scalabile
- realizzare un infrastruttura modulare, a cui potrenno essere aggiunti e tolti componenti in modo da implementare nuove funzionalità

L'utilizzo di architetture a microservizi [3] ha iniziato ad essere adottato dalla community degli sviluppatori a partire dal 2014 in risposta alle tante sfide tecnologiche e di gestione che i team di sviluppo si sono trovati ad affrontare cercando di rendere scalabili le loro applicazioni.
A differenza della classica architettura di tipo monolitico, dove un applicativo web è sviluppato e distribuito come un'unica entità, le funzionalità svolte dall'applicazione vengono divise in n servizi (figura 3.2) ognuno implementato e distribuito separatamente dagli altri, secondo il principio della singola responsabilità.

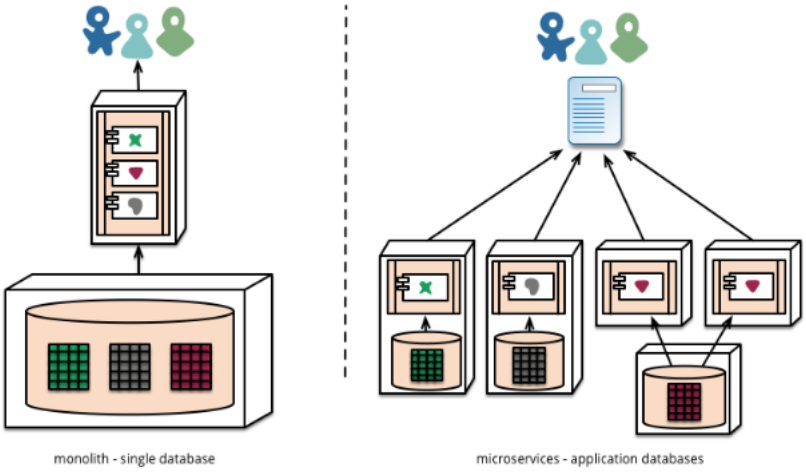


Figura 3.2: Schema di un'architettura a microservizi

Questo approccio rende molto più facile il mantenimento e la gestione di un'infrastruttura altamente scalabile ed orientata al cloud. Ormai la nostra vita è sempre più legata all'utilizzo di internet e sono sempre di più i device connessi. Le aziende che offrono prodotti e servizi basati sul web devono essere in grado di rispondere ad un numero sempre maggiore di richieste in un ambiente in fortissima espansione e altamente competitivo. L'approccio a microservizi è infatti un ottimo modo per cercare di affrontare le sfide imposte dal mercato.
La presenza di un bug o un problema in un certo servizio non implica che tutti gli altri ne vengano influenzati ed è possibile isolare tale problema e risolverlo senza dover interrompere le funzionalità degli altri servizi aumentando la disponibilità dei servizi
Gli sviluppatori che si troveranno a lavorare su di un servizio non dovranno necessariamente conoscere il funzionamento del resto dell'applicazione e questo permette l'uso di team più piccoli che possono lavorare più agilmente e rilasciare più frequentemente nuove versioni e correzioni così come nuovi servizi e quindi funzionalità.
I servizi possono essere replicati per poter essere in grado di far fronte momenti di particolare congestione o sopperire ad eventuali malfunzionamenti.
Dal momento che ogni servizio è separato dagli altri, viene eliminato qualsiasi impegno a

lungo termine per quello che riguarda lo stack di tecnologie utilizzate.
Seppur riuscendo a risolvere alcuni problemi di scalabilità e gestione dell'infrastruttura, quest'approccio introduce nuove criticità legate all'aumento di complessità che vanno tenute in considerazione.
E' necessario mettere in campo sistemi che permettano la gestione dei servizi, come la gestione delle configurazioni, e l'orchestrazione delle istanze in modo da automatizzare il ciclo di vita dei servizi, ad esempio tramite Docker Compose.
E' necessario implementare meccanismi di comunicazione tra i microservizi che però a loro volta vanno ad introdurre latenze e possibili malfunzionamenti.
Dal momento che possono esserci variazioni per quanto riguarda indirizzi e porte su cui sono in ascolto i vari servizi è necessario implementare meccanismi di service discovery, rendendo irrilevante l'effettiva posizione del microservizio.

3.3 Tecnologie utilizzate

Di seguito verranno descritti alcuni dei protocolli e delle tecnologie utilizzate, di interesse trasversale ai vari elementi dell'architattura.

3.3.1 Il protocollo MQTT

MQTT (Message Queue Telemetry Transport Protocol) [1] è un protocollo di messaggistica basato su TCP realizzato specificamente per l'uso in contesti di banda limitata e basso impatto energetico che trova quindi la sua naturale applicazione nei contesti IoT.
Esso si basa sul pattern Publish/Subscribe dove vari client si registrano presso un broker che media la comunicazione tra questi, inviando ai client che si sono iscritti ad un certo topic tutti i messaggi pubblicati relativi a tale topic. Uno dei punti forti di questo approccio è dato dal fatto che i vari client, per comunicare non hanno bisogno di conoscere indirizzi o porte degli altri client grazie alla presenza del broker che funge da mediatore.
Il protocollo si basa principalmente su tre messaggi che vengono scambiati tra il client ed il broker: CONNECT, SUBSCRIBE e PUBLISH.
Il messaggio CONNECT (3.4) viene inviato nel momento in cui un nuovo client intende registrarsi al broker. Tale messaggio contiene vari campi, tra cui username e password in caso il broker li richieda, l'identificativo del client, il Clean Session flag che indica se si intende avviare una sessione persistente ed il tempo di keepalive che indica il tempo massimo in seondi che può trascorrere tra l'invio di due messaggi consecutivi da parte del client. Trascorso una volta e mezzo questo tempo, se il broker non dovesse ricevere un pacchetto di PINGREQ da parte del client , questo verrà considerato disconnesso.
Al messaggio CONNECT segue un CONNACK inviato dal broker che include un valore che indica il successo o meno dell'operazione e l'eventuale motivo del fallimento.
Il messaggio SUBSCRIBE (3.3) indica al broker che il client intende ricevere tutti i messaggi pubblicati relativi a tale topic. Questo messaggio contiene un codice identificativo e la lista di topic a cui si desidera iscriversi con relativo livello di qualità del servizio. In risposta a questo messaggio, il broker invia un pacchetto SUBACK che contine un identificativo ed un codice per ogni topic indicato precedentemente che segnala il successo o meno dell'operazione.

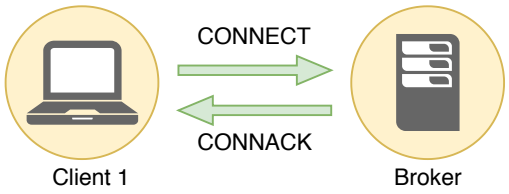


Figura 3.3: Scambio messaggi CONNECT

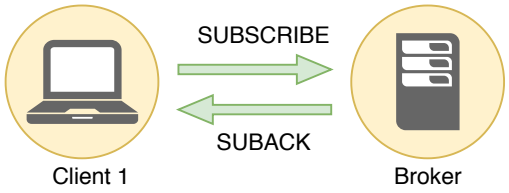


Figura 3.4: Scambio messaggi SUBSCRIBE

Quando un client intende pubblicare un messaggio invia al broker un messaggio di tipo PUBLISH (3.5), all'interno del quale vengono indicati il topic su cui pubblicare il messaggio con relativo livello di servizio e il payload. A questo punto è compito del broker distribuire il messaggio a tutti i client che si sono iscritti al relativo topic. É possibile comporre i topic su più livelli, separandoli con il carattere /.
Ad esempio se un client si iscrive al topic A/B, riceverà tutti i messaggi pubblicati su A e su A/B ma non quelli pubblicati su A/C. In questo modo è possibile indirizzare messaggi a singoli client o a gruppi.
Il protocollo prevede tre livelli di qualità del servizio che rappresentano la garanzia di ricezione di un certo messaggio.

- QoS 0: il messaggio inviato viene ricevuto al massimo una volta, pertanto non è garantita la consegna del messaggio
- QoS 1: il messaggio inviato viene ricevuto almeno una volta. Il client che invia il pacchetto attende di ricevere una conferma di ricezione e lo reinvia in caso questa non arrivi. In questo caso è possibile l'invio multiplo dello stesso messaggio pertanto viene settato un particolare flag che indica il caso in sui il pacchetto sia stato ritrasmesso
- QoS 2: il messaggio inviato viene ricevuto esattamente una volta. Per otenere questo livello di servizio, viene effettuato un protocollo di handshake in quattro fasi

Il livello di QoS viene indicato sia quando si pubblica un messaggio, sia quando viene effettuata l'iscrizione ad un topic. Pertanto, pubblicando un messaggio con un certo livello di qualità del servizio, questo verrà utilizzato per recapitare il messaggio al broker che poi lo invierà al subscriber secondo il livello di qualità di servizio indicato in fase di iscrizione.

3.3.2 Docker e Docker Compose

Docker [17] è una piattaforma software che permette di semplificare il deploy di un'applicazione integrando in un unico pacchetto, detto container, tutto il software e le dipendenze

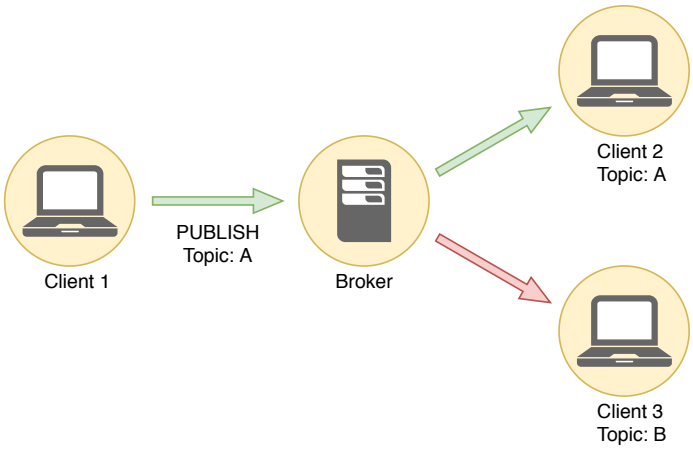
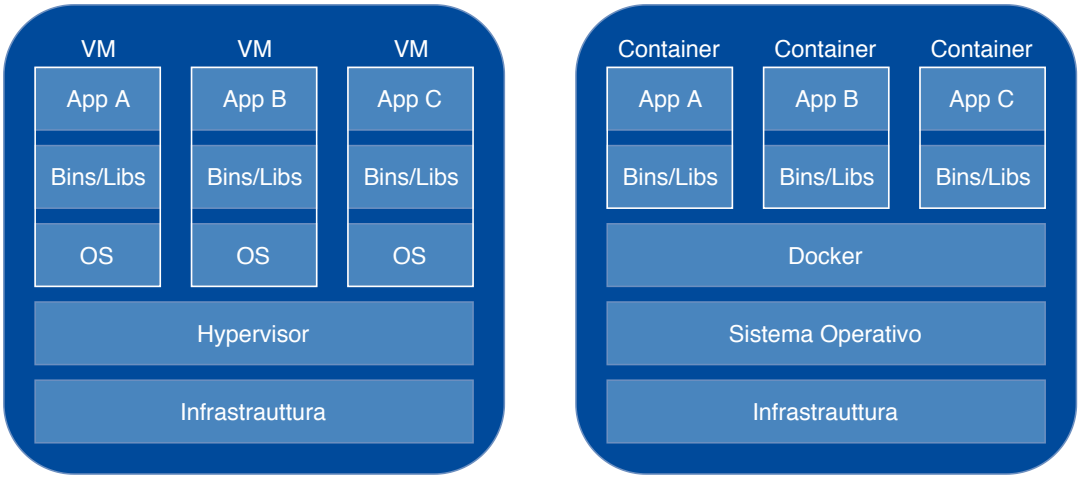


Figura 3.5: Pubblicazione di un messaggio sul topic A, solo il Client 2 riceve il messaggio

necessarie al corretto funzionamento del software. Precedentemente, la soluzione più comune era rappresentata dall'impacchettamento del software in un'istanza di una macchina virtuale che però comporta alcuni problemi: innanzitutto è necessario gestire e deployare le macchine virtuali ed inoltre molte risorse vengono sprecate dal fatto che la virtual machine ospita un sistema operativo completo. Così come una macchina virtuale permette un astrazione del livello fisico, un container docker permette l'astrazione del livello applicativo. Al contrario delle macchine virtuali che hanno bisogno di un sistema operativo a se stante,



(a) Virtual Machine (b) Docker

Figura 3.6: Confronto tra una macchina virtuale e Docker

Docker utilizza le funzionalità del kernel Linux della macchina su cui è installato e grazie all'isolamento delle risorse del kernel, ogni container è isolato dagli altri, rendendo questo approccio sicuro e permettendo una gestione molto granulare di quelle che sono le risorse messe a disposizione per ogni container.

Due dei concetti chiave di Docker sono le immagini ed i container. Le prime sono costruite a partire da una serie di istruzioni che compongono i layer necessari per il funzionamento della nostra applicazione. Questi layer vengono definiti all'interno di un Dockerfile che viene interpretato da Docker portando alla creazione di un'immagine che viene inserita nel docker repository, locale o remoto tramite il comando `build`. Tramite il comando `run` è possibile inizializzare un container a partire dall'immagine indicata nel comando e quindi avviare l'applicativo corrispondente. All'interno del comando `run` è possibile indicare diversi parametri che ci permettono di esporre delle porte, linkare il container ad un volume per la persistenza dei dati e connettere il container ad una rete virtuale.
Nel caso l'applicativo che vogliamo eseguire fosse composto da diversi container, e più in generale in un contesto di produzione, Docker mette a disposizione il tool Docker Compose. All'interno di un file di configurazione, solitamente chimato docker-compose.yml, è possibile definire tutti i parametri dei vari container che compongono la nostra applicazione (chiamati servizi) e gestire tutti i container con un solo comando. Tramite Docker Compose è infatti possibile avviare, fermare e rebuildare i servizi, monitorare il loro stato ed eseguire comandi su uno specifico servizio.

3.3.3 Il paradigma REST

REST (Representational State Transfer) [6] è un paradigma architetturale che ha preso piede con la diffusione delle architetture distribuite. Nella maggior parte dei casi si basa sul protocollo HTTP e permette di interagire con un back end tramite i verbi di tale protocollo, per effettuare tramite semplici richieste al server la creazione, cancellazione, accesso e modifica di risorse. I principi fondamentali su cui si base questo approccio sono: separazione tra client e server: il codice del client e del server possono essere sviluppati in maniera completamente indipendente dal momento che il significato ed il formato dei messaggi scambiati è noto ad entrambi. Tramite uno stesso endpoint diverse applicazione possono interagire con il server ed ottenere gli stessi dati assenza di stato: ogni richiesta e risposta è indipendente da quelle precedenti. Una richiesta REST è composta principalmente dai seguenti componenti :

- Un verbo del protocollo HTTP che indica l'azione che si intende intraprendere.
- Un insieme di Header che permettono di includere informazioni supplementari alla richiesta come ad esempio un token di autenticazione.
- Il percorso di una risorsa su cui eseguire l'azione .
- Opzionalmente un body.

REST prevede l'utilizzo dei verbi del protocollo HTTP per effettuare operazioni CRUD sulle risorse accessibili tramite specifici URL secondo il seguente schema:

- GET recupera una risorsa o un insieme di risorse.
- POST crea una nuova risorsa.
- PUT modifica una risorsa già esistente.
- DELETE rimuove una risorsa.

Per indicare il risultato dell'operazione richiesta vengono utilizzati gli status code del protocollo HTTP, di nostro interesse sono principalmente quelli 2xx che indicano il successo dell'operazione, 4xx che indicano un errore da parte del client e quelli 5xx che indicano un errore da parte del server. L'interpretazione di questi codici permette al client di capire se l'operazione richiesta é andata a buon fine ed in certi casi anche il motivo di un eventuale malfunzionamento.
Gli URL sui quali vengono effettuate queste richieste vanno creati con attenzione in modo da rendere esplicito e comprensibile su che risorsa si intende operare. Ad esempio effettuando una GET alla risorsa `/clienti/123` otterremmo le informazioni relative al cliente 123 mentre effettuando una put sulla stessa risorsa andremo ad aggiornare i dati presenti.
Come detto in precedenza, all'interno degli header è possibile andare a includere informazioni aggiuntive da includere insieme alla richiesta. Tra gli header più importanti abbiamo l'header Accept che indica quali sono i formati MIME accettati come risposta e Content_-Type che invece indica il formato con cui i dati sono inclusi nella richiesta per esempio nel caso di una POST. Un altro header di estrema importanza è l'header Authorization che viene usato per includere il token JWT che permette di autenticare le richieste e implementare meccanismi di autorizzazione.

3.3.4 Oauth2 e JWT

Dato che REST prevede un server stateless, è necessario che ogni richiesta includa le credenziali che permettano di decidere se il richiedente ha il diritto o no di accedere ad una data risorsa e che azioni possa svolgere. JWT [8] è uno standard aperto che definisce un modo compatto e autosufficiente di trasmettere informazioni in modo sicuro tra diverse parti sotto forma di un file JSON. L'informazione contenuta all'interno del token viene firmata in modo da certificarne il contenuto. Un token è composto da tre parti: un Header, un Payload ed una Signature. Il primo è solitamente composto da due campi: il primo che indica l'algoritmo usato per effettuare la firma, solitamente HMAC SHA256 o RSA ed il tipo di token che in questo caso sarà sempre JWT. All'interno del payload si trovano le asserzioni relative all'utente che sta effettuando la richiesta. Il payload e l'header vengono codificati in BASE64 ed utilizzati come parametro della funzione di signature assieme ad un segreto che permette di certificare il firmatario, secondo il seguente schema (nel caso di HMAC SHA256):

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

Al momento della verifica i servizi verificheranno il contenuto del token e la corrispondenza della firma contenuta e quella da loro calcolata usando la loro copia del segreto. Anche la firma viene codificata in BASE64 ed il token è dato da queste tre parti combinate e divise da un ".".
L'utilizzo dei token JWT si inserisce nel più ampio contesto del framework OAuth 2 [7] che definisce metodi di autorizzazione per applicativi web, applicazioni desktop e mobile. Il framework permette ad una terza parte di ottenere un accesso regolato ad un servizio HTTP.
Lo standard prevede l'interazione tra quattro attori: il resource owner, che è in grado di fornire l'accesso ad una risorsa protetta, il resource server, che ospita la risorsa protetta, il client, che è l'applicazione che intende accedere alla risorsa protetta e l'authorization server,

l'entità che rilascia il token d'accesso al client in seguito alla corretta autenticazione. Nello schema 3.7 è evidenziata l'interazione tra i vari attori descritti in precedenza. OAuth2

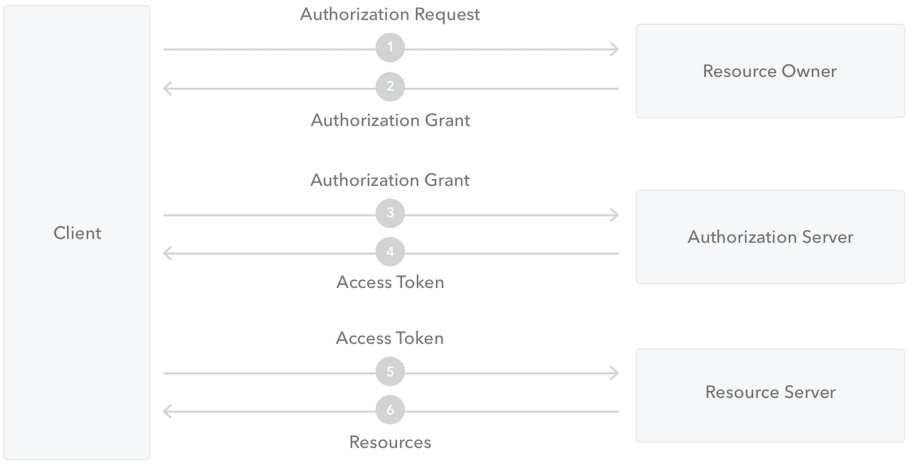


Figura 3.7: Interazione tra le parti in OAuth2

permette dunque di separare il ruolo del client che accede ai dati dal possessore delle risorse in modo da fornire a quest'ultimo un maggiore controllo sull'accesso alle risorse da parte del client. Infatti all'interno del token che verrà rilasciato saranno definiti i termini di accesso alla risorsa.

3.3.5 MongoDB

Come database è stato scelto Mongo Db, che rientra nella categoria dei database non relazionali ed in particolare quelli cosiddetti document-based. A dispetto dei database relazionali, si possono evidenziare alcune principali differenze.
L'unità fondamentale è data dai documenti che sono paragonabili alle righe di una tabella SQL ma non hanno uno schema fisso, questo permette una gestione più semplice dei dati visto che eventuali modifiche non obbligano a modificare lo schema del database. I documenti contengono coppie chiave-valore che sono paragonabili alle colonne di una tabella SQL. I documenti possono essere raccolti all'interno di una collection, che è paragonabile ad una tabella SQL.
I database NoSQL inoltre non fanno uso del linguaggio SQL ma utilizzano apposite API, in questo caso tramite una CLI che utilizza comandi in linguaggio javascript.
I database non relazionali non rispettano le proprietà ACID (Atomicità, Consistenza, Isolamento e Durabilità) tipiche dei database relazionali ma basano il loro funzionamento sul teorema CAP, che è l'acronimo di Consistency, Availability e Partition tolerance.
Il teorema CAP afferma che qualsiasi sistema distribuito può supportare contemporaneamente solo due di queste proprietà.
Consistency implica che quando un dato è distribuito tra più nodi, ogni nodo dovrà vedere lo stesso dato in ogni istante di tempo, pertanto garantendo che la risposta contenga sempre

il dato più recente.
Availability implica che ogni richiesta inviata al sistema genera una risposta valida che però non necessariamente conterrà il dato più recente.
Con partition tolerance si intende che il sistema deve essere in grado di funzionare anche in caso di problemi di rete tra i vari nodi e si ottiene tramite la replicazione dei dati tra più nodi.
Per rendere più chiaro il significato di questo teorema, supponiamo di avere un sistema distribuito su quattro nodi, che si scambiano informazioni mantenendo le informazioni consistenti quindi diamo per scontato che il sistema non sia suscettibile al partizionamento. Il nodo 1 riceve la richiesta di modifica del documento A mentre il nodo 3 riceve la richiesta di lettura dello stesso record prima che la modifica venga propagata al terzo nodo.
In questo momento si profilano due possibili scenari: nel primo caso il nodo 3 fornisce come risposta la propria versione del dato richiesta ma così non è rispettata la consistency. Nel secondo caso il nodo non fornisce una risposta andando a violare l'availability.
Da questo possiamo derivare tre classi per descrivere i sistemi distribuiti: sistemi CA, sistemi CP e sistemi AP All'interno della prima classe troviamo i database relazionali, mentre mongoDb rientra nella seconda categoria.
MongoDb tuttavia permette di operare in una modalità unsafe, dove cioè non viene imposta la consistency ma solamente l'eventual consistency (non si attende la propagazione dei dati in modo consistente tra tutti i nodi) andando però a beneficiare a livello di availability, dato che verrà sempre fornita una risposta ma non necessariamente conterrà il dato più aggiornato.
Tra le principali caratteristiche di MongoDB troviamo infine il supporto nativo per lo sharding, cioè un metodo per la distribuzione di dati tra diverse macchine, permettendo così una forte scalabilità orizzontale, riuscendo a gestire moli di dati sempre maggiori con un'ottima efficienza.
Pertanto il principale motivo che ha portato alla scelta di questo database è stata la capacità di gestire dati con schema variabile, per venire incontro alle modifiche apportate durante la fase di sviluppo. Attualmente, dato lo scarso numero di sniffer impiegati, non è stato necessario suddividere il carico di lavoro tra più repliche ma in un contesto di utilizzo reale questa caratteristica potrebbe tornare molto utile.
Tra le api messe a disposizione da Mongo, che permettono ovviamente operazioni CRUD e query complesse, è presente un insieme di operatori che permettono di eseguire query aggregando insieme diversi documenti, creando vere e proprie pipeline a più stadi per ottenere complesse analisi sui dati. Questa funzionalità, che è stata molto utilizzata all'interno dell'applicativo, è permessa dal framework Mongo Aggregation. Il suo funzionamento si basa sulla combinazione di stage di aggregazione il cui comportamento può essere modificato tramite l'uso di opportuni operatori.

Capitolo 4

Implementazione degli sniffer

4.1 La piattaforma ESP32

La piattaforma hardware scelta per la realizzazione dei sensori è il SoC ESP-WROOM-32 sviluppato dall'azienda cinese Espressif Systems. Questo SoC fa parte della famiglia di micro controllori ESP-32, che include chip low-cost a bassi consumi energetici. Le specifiche tecniche includono:

- Processore Tensilica Xtensa LX6 in configurazione dual e single core ad una frequenza variabile tra 160 e 240 MHz
- 512 KiB di SRAM
- Memoria Flash da 4 MiB
- Wi-Fi 802.11 b/g/n
- Bluetooth 4.2 con supporto BLE

Su vari store online è possibile acquistare development board (figura 4.1) che integrano il chip oltre alla circuiteria necessaria al suo funzionamento out of the box (alimentazione, connessione tramite USB) in modo da poter iniziare facilmente lo sviluppo sulla piattaforma. Il device richiede un'alimentazione di 5 Volt forniti da un alimentatore da smartphone

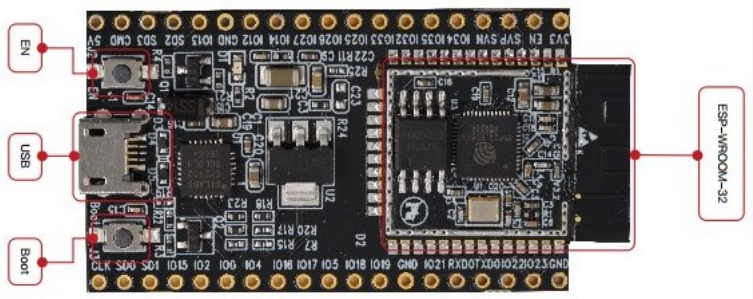


Figura 4.1: Una board che integra il SoC ESP-32

collegato al guscio ottenuto con stampa 3D. Al momento il design è molto spartano ed in una versione successiva si potranno includere i circuiti di alimentazione da rete elettrica direttamente all'interno del guscio così da rendere il tutto più compatto.

4.2 Il framework ESP-IDF

Per quanto riguarda lo sviluppo del firmware è stato usato il framework nativo sviluppato da Espressif, ESP-IDF (Espressif IoT Development Framework) basato sul linguaggio C. Con piccole modifiche ai file sorgenti è stato possibile utilizzare il linguaggio C++. Il framework è composto da diversi componenti: ESP-IDF contiene le librerie software ed il codice sorgente che implementano le API. Toolchain è l'insieme di software necessari alla compilazione del progetto. É possibile scaricare da GitHub progetti già preconfigurati che fungono da base per la creazione di progetti più complessi. Una volta effettuato correttamente il setup del framework è possibile richiamare all'interno della cartella del progetto il comando make per interagire con il Toolchain tramite principalmente tre comandi:

- `make menuconfig` permette di accedere ad un interfaccia di configurazione del progetto in cui è possibile andare a configurare variabili statiche e parametri del progetto. É possibile difinire anche proprie variabili in modo da poterle modificare senza andare ad agire sui file sorgenti una volta compilati. Questi dati vengono salvati dentro il file sdkconfig presente all'interno della cartella del progetto.

- `make flash` permette di richiamare gli script necessari all'upload del firmware nella memoria del device.

- `make monitor` permette di osservare da terminale l'output generato dal device per fini di debug.

Tutti questi comandi scatenano la compilazione del file sorgente se vengono rilevate delle modifiche.

4.2.1 Configurazione della memoria

La distribuzione di dati ed applicazioni all'interno della memoria flash è determinato da una partition table che viene memorizzata nella memoria stessa all'indirizzo `0x8000` e definita tramite un file csv. Ogni partizione prevede un nome (label), il tipo (app, data, ...), un sottotipo e l'offset rispetto il quale la partizione è caricata nella memoria.
Tramite il comando `make menuconfig` è possibile scegliere che configurazione usare tra alcune proposte di fabbrica o una definita dal programmatore, indicando il file .csv in cui è definita la nuova configurazione. Una delle configurazioni standard è "Single factory app, no OTA", descritta qui di seguito:

```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
```

La partizione nvs è adibita allo storage di dati non volatili in formato coppie chiave valore (nvs infatti significa non volatile storage) accessibili tramite le API messe a disposizione dal framework che ne permettono la scrittura, la lettura e la cancellazione.
La partizione phy_init contiene i dati di inizializzazione relativi al layer fisico dei vari sistemi di comunicazione radio integrati. Durante lo sviluppo del progetto questi aspetti non sono stati approfonditi in quanto non necessari al funzionamento del prodotto.
Per lo storage non volatile di informazioni quali ssid e password delle reti, indirizzi dei server di configurazione e stata usato SPIFFS (SPI Flash File System), un file system sviluppato per sistemi embedded. I vari file necessari alla configurazione vengono letti durante l'inizializzazione del dispositivo. Si è preferito usare questo approccio piuttosto che nvs in quanto in questo modo è possibile salvare un numero arbitrario di configurazioni, ad esempio è possibile salvare in un file diverse coppie ssid e password mentre con nvs si sarebbe dovuto usare una chiave diversa per ogni possibile coppia rendendo l'approccio inutilmente complicato. Infine la partizione factory contiene i dati della nostra applicazione.
La partition table risultante è la seguente, si puo notare come sia stato necessario aumentare la dimensione della partizione factory per poter contenere il firmware.

```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 2M,
storage,  data, spiffs,  0x280000,1M,
```

4.3 Funzionalità wireless di ESP32

Le funzionalità wireless della piattaforma ESP32 prevedono tre modalità di funzionamento: STA (station), AP (access point) e AP-STA (modalità ibrida). Nel primo caso il device si comporta come una normale stazione wifi e cerca di connettersi all'access point indicato in fase di configurazione. Nel secondo caso il device funge da access point che resta in attesa di richieste di connessione. Infine, nella modalità AP-STA convivono queste due modalità. Nel progetto è stata usata questa modalità in quanto è necessario che il device sia collegato ad una rete su cui riversare i dati raccolti e permettere il collegamento di un client per poter visualizzare messaggi di errore o per modificare la configurazione a monte del processo di configurazione automatica. In tutte le possibili modalità di funzionamento è possibile attivare anche la modalità promiscua attraverso la chiamata alla funzione `esp_wifi_set_promiscous()`. In questa modalità i pacchetti di tipo 802.11 Management frame, 802.11 Data frame e 802.11 Control frame vengono catturati anche se non diretti allo sniffer. Di default è presente un filtro che rende disponibili all'applicazione solamente i pacchetti di tipo Data e Management. Questo aspetto è di grande importanza dato che la principale funzione del progetto è legata a questa potenzialità del device.

4.4 Funzionamento generale

Il funzionamento del device può essere suddiviso in tre fasi principali:

- Startup e configurazione.
- Raccolta dei pacchetti di management.
- Invio dei dati raccolti ed elaborati.

Questa può essere suddivisa a sua volta in ulteriori cinque sotto fasi:

- Montaggio partizione SPIFFS e lettura dei file di configurazione statica
- Inizializzazione dello stack di rete e connessione ad una rete wifi
- Avvio di un server http per servire le pagine che permettono la modifica della configurazione statica
- Richiesta del token di autenticazione
- Richiesta della configurazione dinamica
- Connessione al broker MQTT e iscrizione ai topic di interesse

4.4.1 Lettura dei file di configurazione

All'interno della partizione spiffs sono presenti due file soprannominati `wifi.txt` e `config.txt` che contengono i dati di configurazione statica dello sniffer. Al momento dell'avvio del dispositivo, viene letto il primo file che contine nel formato `ssid password` le credenziali di accesso salvate dei vari access point ai quali tenterà di connettersi. Tali dati vengono usati per inizializzare oggetti della classe `WifiAccessPoint` che vengono inseriti in una lista. Il file `config.txt` contiene invece l'indirizzo e la porta di ascolto del server di configurazione dal quale verrà scaricata la configurazione dinamica del dispositivo. I dati contenuti nel file vengono salvati in una variabile ed usati in seguito.
Nel caso in cui la partizione non contenesse questi file, ad esempio nel caso di un primo avvio dello sniffer, verranno creati e sarà possibile modificarli accedendo alla pagina di configurazione.

4.4.2 Inizializzazione dello stack di rete

Se i passi indicati precedentemente vanno a buon fine, viene chiamata la funzione `wifi_init()` che è deputata all'inizializzazione e configurazione dello stack wifi. Dal momento che viene usata la modalità ibrida AP-STA è necessario specificare entrambe le configurazione separatamente chiamando due volte la funzione `esp_wifi_set_config()` specificando di volta in volta la modalità a cui si fa riferimento. Nel caso si stia configurando la modalità AP, andranno principalmente indicati il nome della rete che verrà ospitata dal device, la password per accedervi ed eventualmente altri parametri come il numero massimo di device connessi contemporaneamente. Per la configurazione della modalità STA, si deve indicare l'SSID e la password della rete a ci si intende accedere. Dal momento che è possibile salvare più reti nella memoria del dispositivo, la configurazione della modalità station viene ripetuta più volte finché non avviene correttamente la connessione.

4.4.3 Inizializzazione e configurazione del server HTTP

IL framework ESP-IDF mette a disposizione un server HTTP che è stato usato per servire la pagina di configurazione necessaria alla modifica delle impostazioni statiche del dispositivo e per ricevere tramite POST i dati necessari alla modifica dei file di configurazione. La pagine è accessibile connettendosi alla rete Wi-Fi fornita del dispositivo e collegandosi all'indirizzo 192.168.4.1. In tale pagina è possibile:

- Aggiungere le credenziali di accesso di una nuova rete WiFi.
- Modificare l'indirizzo e la porta del servizio di configurazione.
- Cancellare la lista di credenziali WiFi salvate precedentemente.
- Riavviare il dispositivo.

La funzione `startHttpServer()` gestisce l'inizializzazione del server e la registrazione degli hanlder delle richieste. La funzione ritorna una variabile di tipo `httpd_handle_t` che rappresenta la handle dell'istanza del server.
`httpd_start()` è la funzione deputata all'inizializzazione del server http. In caso di esito favorevole vengono registrate le handler delle richieste chiamando la funzione
`httpd_register_uri_handler()`, che riceve come parametri la handle del server e un puntatore ad una struct di tipo `http_uri_t`. La struct di tipo `http_uri_t` permette di specificare una URI, il metodo HTTP supportato e la funzione che deve gestire la richiesta. Per il nostro scopo vengono registrate due di queste struct per gestire richieste GET e POST alla uri "/".
Nel primo caso viene chiamata la funzione `esp_err_t get_handler(httpd_req_t *req)` che risponde alla richiesta fornendo la pagina di configurazione.
La pagina di configurazione, riportata in figura 4.2, contiene dati dinamici e quindi non può essere salvata come file statico all'interno della memoria del dispositivo. La funzione che gestisce la composizione della pagina riceve come parametri i dati dinamici come la lista di reti wifi salvate, eventuali messaggi di errore e l'indirizzo del server di configurazione, e compone la pagina ritornando una stringa che contiene tutto il codice Html e Css che viene inserito nella risposta.
Nel secondo caso viene chiamata la funzione `post_handler()` che esegue una lettura bufferizzata del payload della POST. Il body della richiesta viene inviato alla funzione `urlencoded_translator()` che decodifica il body che è in formato x-www-form-urlencoded pur mantenendo il carattere & tra i vari parametri. Il risultato viene passato alla funzione int `POST_request_parser()` che ha lo scopo di individuare il comando che è stato inviato. Una prima distinzione consta nel rilevare o meno all'interno della stringa un carattere '&' che implica che sono presenti due parametri. Nel caso si stesse cercando di inserire una nuova rete il comando ricevuto sarà:

SSID=mySSID&Password=myPassword

A questo punto mySSID e myPassword vengono passate come parametro alla funzione `addNetwork()` che scrive le nuove credenziali in coda al file di configurazione wifi.txt. Nel caso si volesse modificare l'indirizzo del server di configurazione il comando sarà:

server=serverAddress&port=serverPort

L'update dei dati avverrà in modo analogo al precedente.

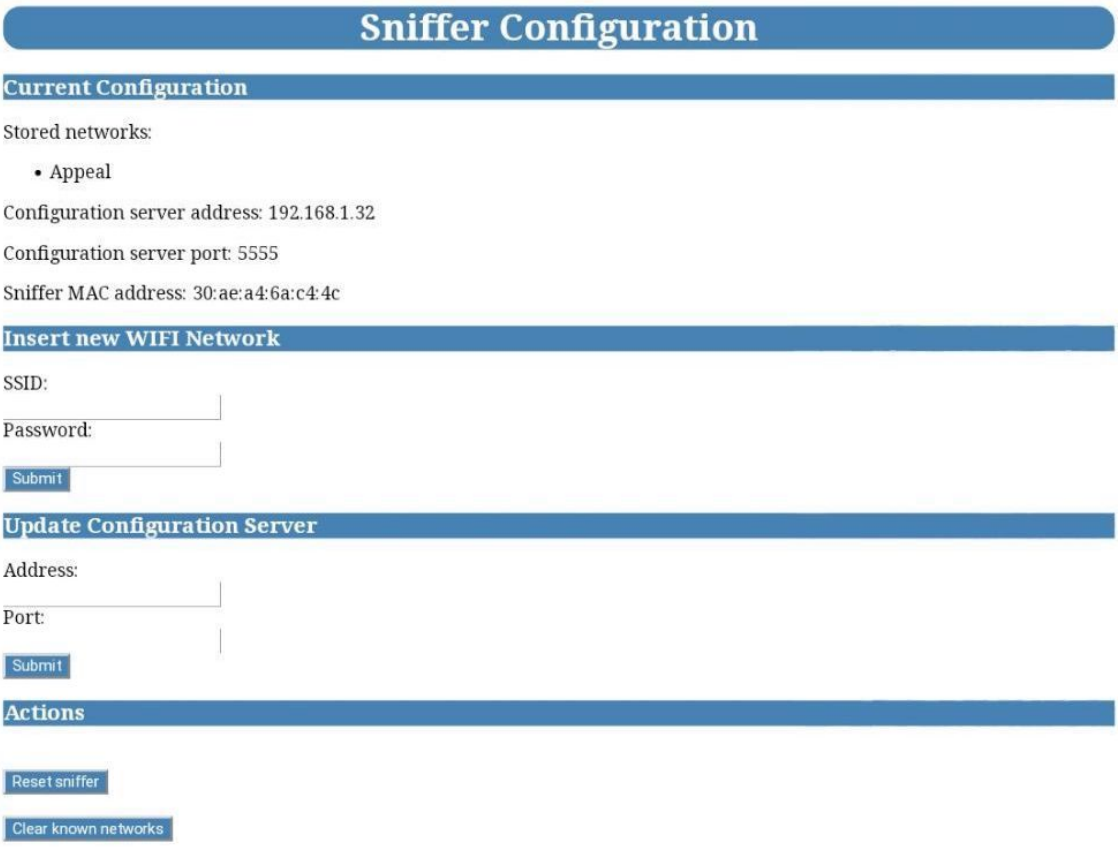


Figura 4.2: Pagina di configurazione degli sniffer

Se non viene individuato il carattere '&', i comandi possibili sono solo reset, a seguito del quale il device si riavvia dopo 3 secondi, e clear. In questo ultimo caso viene chiamata la funzione `clearSavedNetworks()` che cancella il contenuto del file di configurazione `config.txt`. In caso si dovessero verificare degli errori verrà ritornato 1, altrimenti 0. Tornando alla funzione chiamante, in base al valore ritornato verrà settata la variabile message in modo da poter vedere se la procedura è andata a buon fine e verrà inviata la pagina di risposta contenente i dati aggiornati. Dal momento che viene stabilita la connessione WiFi è possibile inviare la richiesta al server di autenticazione per ottenere i token necessario alle successive interazioni con il back end.
La richiesta avviene tramite il client http presente in ESP-IDF. I parametri della richiesta vengono definiti in una struct del tipo `esp_http_client_config_t`. In questa richiesta vengono inviati tramite POST al servizio che gestisce l'autenticazione le credenziali dello sniffer.
Lo username (in questo caso soprannominato deviceName) è formato dall'indirizzo mac dello sniffer dal quale sono stati rimossi i ':' mentre la password è ottenuta prendendo i primi 12 caratteri risultanti da `MD5(deviceName+secret)` dove secret è una stringa comune a tutti i dispositivi, definita in fase di programmazione.

Alla richiesta è necessario aggiungere tre header specifici in modo che il servizio di autenticazione accetti la richiesta correttamente la richiesta:

- Content-Type: application/x-www-form-urlencoded
- Accept: Application/json
- Authorization: Basic + `BASE64(username:password)`.

L'ultimo header è necessario in quanto il servizio che eroga il token JWT è protetto da Http Basic Authentication. I parametri username e passoword sono definiti all'interno del servizio, pertanto la stringa risultante dall'applicazione della codifica BASE64 è in realta definita al momento della scrittura del firmware. All'interno del body della richiesta verranno inseriti il deviceName, la password definita precedentemente come valori dei campi username e password oltre al grant_type che permette di indicare il metodo con cui si intende richiedere il token, in questo caso la stringa "password" per indicare che il token viene richiesto in cambio delle credenziali d'accesso, secondo lo standard OAuth2[7].
Settati i parametri del body e gli header è possibile effettuare la richiesta che viene eseguita in maniera bloccante, cioè non ritorna finche non si ottiene una risposta dal server o la richiesta va in timeout. La funzione indicata come handler della richiesta effettua una lettura bufferizzata della risposta del server.
A questo punto nel buffer di ricezione è contenuto il token JWT inviato dal server insieme ad altri campi in formato JSON. Viene usata la libreria cJSON per effettuare il parsing del payload così da estrarre il token. Avendo ora a disposizione il token che autentica le richieste al server, viene eseguita una seconda richiesta al servizio deputato alla gestione degli sniffer in modo tale da ottenere la configurazione dinamica. Il token, per essere correttamente riconosciuto dal server va inserito dentro un header di tipo Authorization preceduto dalla dicitura "bearer". Se l'operazione avviene correttamente, il file di configurazione viene ottenuto in formato JSON. I campi contenuti nel file di configurazione sono:

- Dump mode: variabile booleana che indica se lo sniffer deve funzionare in modalità dump o parsed, cioè effettuare l'elaborazione dei pacchetti prima dell'invio o no
- Privacy mode: variabile booleana che indica se effettuare o no la pseudonimizzazione dei MAC adress rilevati
- Broker Address: stringa contenente l'indirizzo del broker MQTT
- Broker Port: intero corrispondente alla porta di ascolto del broker MQTT
- Topic: stringa contenete il nome del topic su cui andranno pubblicati i dati
- Power thrashold: intero che indica il valore minimo di RSSI che un pacchetto deve avere per non essere scartato

L'RSSI è un valore che indica la potenza con cui lo sniffer ha ricevuto un pacchetto. Il valore varia da -100 a 0, che corrisponde alla potenza maggiore. Il valore di default è -100 che indica che tutti i pacchetti non vengono scartati, utilizzando valori più prossimi allo 0 si andrà a selezionare solo i pacchetti ricevuti con potenza maggiore e quindi provenienti da device più vicini.

4.4.4 Inizializzazione e configurazione del client MQTT

Il framework ESP-IDF, come indicato precedentemente, dispone di un client MQTT nativo. Il client va configurato tramite la struct `esp_mqtt_client_config_t` all'interno della quale possiamo indicare l'indirizzo e la porta usati dal broker, eventualmente un livello di qualità del servizio, l'intervallo di keepAlive, un client_id che permette di identificare il dispositivo connesso e le credenziali di accesso al broker. Nel nostro caso, questi ultimi tre valori sono il deviceName definito precedentemente sia come client_id che come username e la devicePassword. All'interno della configurazione viene indicata anche la funzione che avrà il compito di rispondere agli eventi lanciati dal client.
Successivamente la configurazione viene passata come parametro alla funzione `esp_mqtt_client_init()`.
La chiamata ritorna una handle di tipo `esp_mqtt_client_handle_t` che rappresenta un puntatore ad una struct di tipo `esp_mqtt_client` che viene passata a sua volta come parametro alla funzione `esp_mqtt_client_start()`. Da questo momento è possibile iscriversi a topic, pubblicare e ricevere messaggi chiamando le opportune funzioni.
Quando viene eseguita l'inizializzazione del client, questo tenta ad intervalli regolari di collegarsi con il broker. Stabilita la connessione viene lanciato l'evento `MQTT_EVENT_CONNECT`. In reazione a questo evento, la funzione che lo gestisce iscrive i client ai topic "commands" e "commands
snifferName", avvia la modalità promiscua in modo da iniziare la raccolta dei pacchetti e registra la funzione `wifi_sniffer_cb()` come callback de chiamare in seguito alla ricezione di un pacchetto. Il client mqtt viene iscritto ai topic "commands" e "commands/snifferName" in modo da poter ricevere comandi dal back end. Per ora l'unico utilizzo di questa funzione è inviare un messaggio contenente la stringa "reset" che indica allo sniffer di riavviarsi. Questa funzionalità è stata aggiunta all'interno della handler dell'evento `MQTT_EVENT_DATA` che viene scatenato alla ricezione di dati su un topic a cui è iscritto il client, per permettere il riavvio remoto dello sniffer quando viene cambiata la configurazione così che riavviandosi venga scaricata quella nuova. Pubblicando il messaggio sul topic "commands" vengono riavviati tutti i sniffer connessi mentre solo quello specificato se viene indicato uno snifferName come subtopic. Per l'invio di questo messaggio viene usato QoS 2 in modo da garantire la ricezione del messaggio da parte di tutti i client una ed una sola volta.

4.4.5 Cattura dei pacchetti

La funzione `wifi_sniffer_cb` ha il principale compito di identificare il tipo di pacchetto ricevuto e, nel caso fosse un probe request, analizzarlo in base alla configurazione ottenuta. La callback riceve come parametro un puntatore a void che referenzia il buffer in cui verrà inserito il pacchetto catturato. Il puntatore subisce un cast come puntatore ad una struct di tipo `wifi_promiscous_pkt_t` che ha come campi una struct del tipo `wifi_pkt_rx_ctrl_t` che contiene i metadati del pacchetto e un puntatore a `uint8_t` che punta al payload del pacchetto.
Dalla prima struct che, come detto prima, contiene principalmente i metadati relativi al pacchetto e le informazioni radio, è possibile prelevare diversi dati di interesse tra cui il RSSI (received signal strength indicator), e la lunghezza del payload del pacchetto.
Il puntatore al payload subisce a sua volta un cast al tipo `sniffer_payload_t *`. Questo

puntatore fa riferimento all'omonima struct che contiene la parte 802.11 del pacchetto, indicando l'header, il MAC di destinazione, il MAC sorgente, il BSSID, ed un puntatore al payload del pacchetto. Leggendo l'header è possibile individuare il tipo di pacchetto che, nel nostro caso, probe request, è identificato dal codice esadecimale `0x04`. I pacchetti contenenti un codice diverso vengono immediatamente scartati.
Se nella configurazione è stata indicata una potenza minima del segnale, vengono scartati anche i pacchetti che presentano una potenza inferiore. Questo è stato fatto in quanto è possibile in certi frangenti voler solo raccogliere pacchetti a potenza elevata per garantire, seppure con una certa incertezza, che siano stati raccolti da dispositivi non troppo distanti.
Sono stati previsti due metodi di funzionamento: dump mode e parsed mode, la cui scelta viene indicata all'interno del file di configurazione ottenuto dal servizio apposito. Nel primo caso il pacchetto probe request subisce un processamento minimo in modo da poter avere il payload a disposizione per poter fare successive analisi. Tuttavia la quantità di dati inviati al broker è paragonabile alla grandezza del pacchetto ricevuto, pertanto molto tempo viene sprecato per l'invio dei dati permettendo una raccolta di un minor numero di pacchetti.
In questa modalità il pacchetto inviato al server è composto come da figura 4.3.

Figura 4.3: Struttura del messaggio inviato sul topic "dump"

In questo caso il pacchetto non viene inserito in una coda ma inviato direttamente al broker sul topic "dump".
Nella modalità di funzionamento parsed, il pacchetto viene processato in modo da inviare al server soltanto l'insieme di dati necessari alle nostre analisi. Una prima distinzione viene fatta in base al fatto che il pacchetto indichi un indirizzo locally administered o globally administered. Nel primo caso siamo di fronte ad un pacchetto generato da un device che non si annuncia con il suo MAC definito di fabbrica. Sarà necessario, quindi, come descritto nei capitoli precedenti, calcolare il fingerprint del pacchetto. Per la composizione della fingerprint vengono usati, come descritto precedentemente, i seguenti parametri:

- La lunghezza totale del pacchetto al netto della lunghezza di un eventuale SSID
- La stringa contenente i tag ordinati nell'ordine in cui appaiono
- Il value associato al tag nel caso sia interessante per differenziare i vari device, in particolare sono stati considerati i tag DD, 01, 32, 7F, 2D, BF

I dati vengono inseriti in un buffer e ne viene calcolato l'hash usando l'algoritmo MD5.
I dati estratti dal pacchetto vengono usati per inizializzare un oggetto della classe `ProbeRequestData` che viene inserito in una coda thread safe, implememntata dalla classe generica `SharedQueue`. La classe generica incapsula una normale `std::queue<T>` ma è stata resa thread safe tramite un mutex che permette la sincronizzazione delle letture e delle scritture tra più thread. La classe mette a disposizioni i seguenti metodi:

- `size()`: ritorna il numero di elementi contenuti nella coda

- `popFront()`: ritorna l'elemento in cima alla coda
- `pushBack()`: inserisce un elemento in fondo alla coda
- `checkEmpty()`: ritorna true se la coda è vuota

Il thread principale è responsabile dell'inserimento nella coda dei pacchetti raccolti ed analizzati. A seguito della connessione con il broker viene avviato un secondo thread che esegue la funzione `mqtt_task()` che sarà responsabile di prelevare i pacchetti dalla coda in cui sono stati inseriti e inviarli al broker per il dispatch, come descritto in figura 4.6. Nel caso la coda fosse vuota, il thread attende 100 ms prima di riprovare ad estrarre un elemento. Negli schemi 4.4 e 4.5 è riportata la struttura dei messaggi inviati al broker rispettivamente per i pacchetti con indirizzo MAC globally administered e locally administered. Nel primo caso (figura 4.4) il messaggio inviato è così formato:

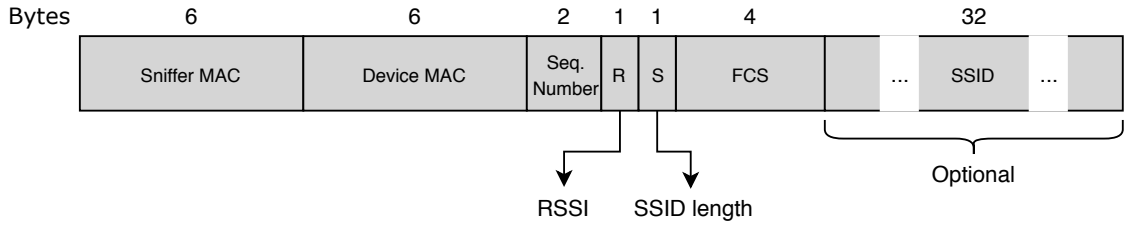


Figura 4.4: Schema del contenuto del messaggio originato da pacchetto con indirizzo MAC globally administered

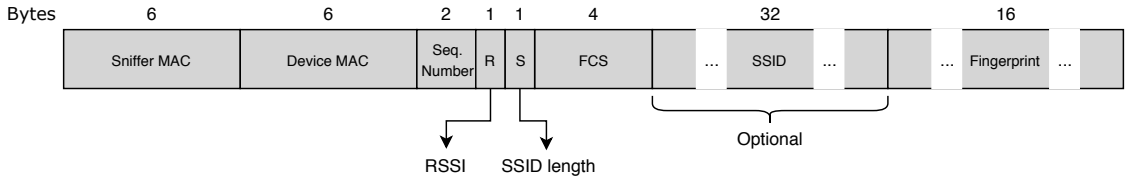


Figura 4.5: Schema del contenuto del messaggio originato da pacchetto con indirizzo MAC locally administered

- MAC address dello sniffer che ha catturato il pacchetto
- MAC address del device che ha emesso il pacchetto
- numero di sequenza del pacchetto
- RSSI
- lunghezza dell'eventuale SSID presente all'interno dei tagged parameters
- FCS (frame check sequence) del pacchetto
- eventuale SSID presente all'interno dei tagged parameters

Nel caso il pacchetto sia local, ai campi descritti precedentemente verrà aggiunto in coda il fingerprint calcolato.
Nel primo caso la grandezza massima del pacchetto inviato sarà pari a 52 byte mentre nel secondo caso 68.
A parte gli indirizzi MAC dei device ed il fingerpint, gli ulteriori dati inseriti hanno le seguenti finalità:

- FCS: individuare eventuali pacchetti catturati contemporaneamente da più sniffer
- lunghezza dell'eventuale SSID: permettere la lettura corretta del SSID eventualmente presente nel messaggio
- RSSI: stimare la distanza del device dallo sniffer

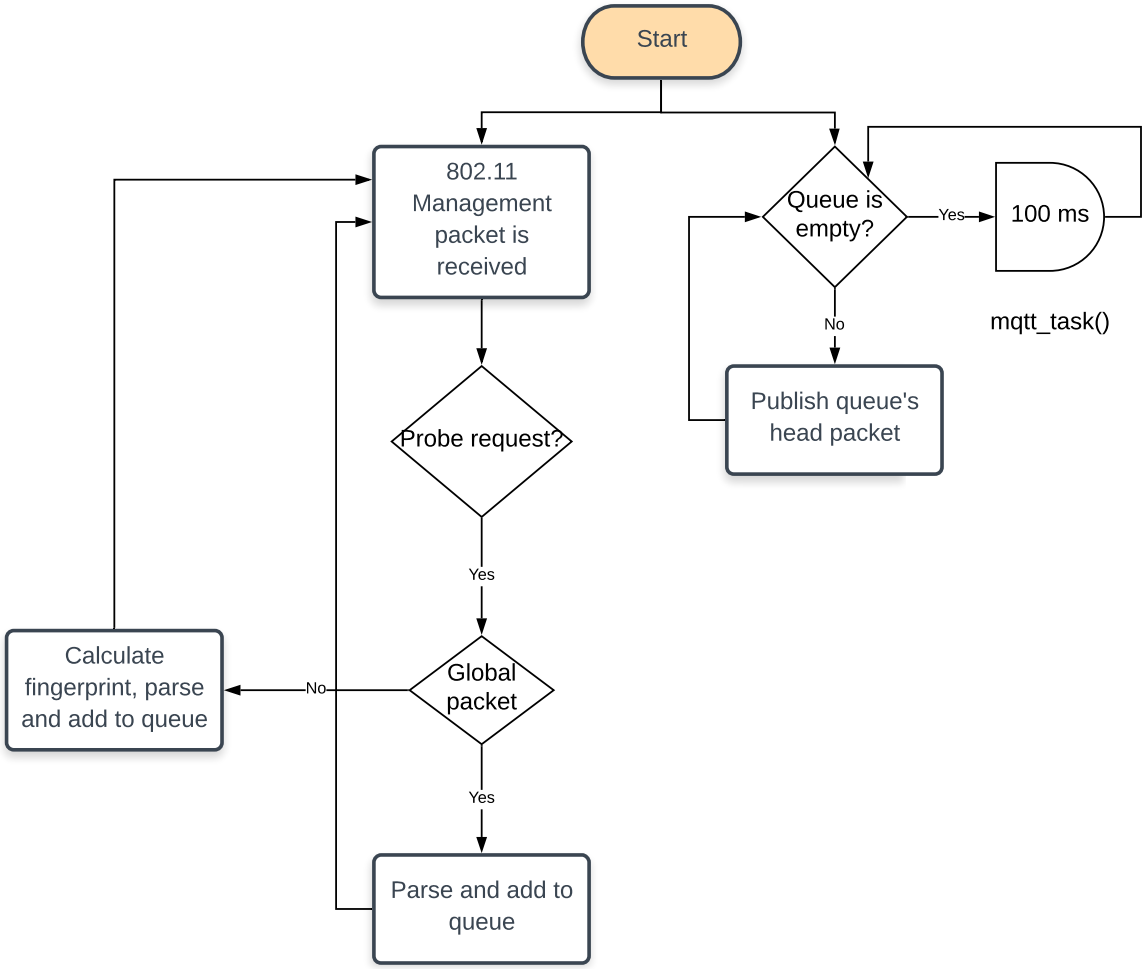


Figura 4.6: Flow chart del funzionamento dello sniffer durante la fase di raccolta ed invio dei pacchetti

4.5 Gestione degli errori e recovery

Sono stati implementati diversi sistemi per far fronte ad eventuali problemi di connessione sia con la rete wifi sia con il broker. Possiamo individuare due tipi di errore: errori critici che non permettono un funzionamento regolare del device e non sono recuperabili ed errori non critici che possono essere recuperati. Tra gli errori critici possiamo individuare due principali classi: quelli che prevedono l'hard reset immediato del dispositivo e quelli che prevedono un reset ritardato in quanto causati con buona probabilità da errori di configurazione. Il ritardo nel reset ha lo scopo di permettere a chi sta effettuando la configirazione di aver abbastanza tempo per individuare la causa dell'errore ed eventualmente correggerlo.
Nella prima classe ricadono gli errori verificatisi durante la lettura dei file nella partizione SPIFFS e l'inizializzazione del server http. In questi casi è impossibile un funzionamento corretto ed il device si riavvia istantaneamente. Tuttavia è molto difficile che si verifichino questi tipi di errori se non per errori durante la fase di programmazione.
Della seconda classe fanno parte errori verificatisi durante la fase di fetch del token JWT o del file di configurazione.
Nel primo caso è impossibile continuare nello svolgimento del programma. Viene settato un messaggio specifico nella pagina di configurazione per permettere di identificare la ragione del malfunzionamento ed intervenire per correggere l'errore. Passati 3 minuti il device si resetterà e proverà di nuovo ad ottenere il token. Questo errore può presentarsi in diversi casi:

- Credenziali sbagliate
- Indirizzo o porta del server di autenticazione sbagliati
- Server di autenticazione irraggiungibile

Nel caso l'errore si verifichi durante la fase di fetch del file di configurazione e possibile in alcuni casi continuare lo svolgimento del programma. Se nel file mancassero dei campi o i valori inseriti non fossero corretti verranno inseriti dei valori di default, tuttavia questo non vale per l'indirizzo e la porta del broker che non possono avere un valore di default. In questo caso e nel caso in cui fosse impossibile reperire il file JSON contenente la configurazione, il device si resetterà passati 3 minuti, similarmente al caso precedente. Questo errore può presentarsi in diversi casi:

- Credenziali sbagliate
- Indirizzo o porta del server di configurazione sbagliati
- Server di configurazione irraggiungibile

Nella classe degli errori recuperabili ricadono le disconnessioni dalla rete wifi e dal broker mqtt. Nel primo caso possiamo individuare due eventualità:

- Impossibilità di connettersi a nessuna delle reti salvate
- Disconnessione avvenuta in seguito ad una prima connessione, che implica che le credenziali di accesso sono corrette

Nel primo caso, prima di avviare il server http viene fatto un tentativo di connessione a tutte le reti salvate fino a che non avviene la connessione ad una di queste. In caso contrario viene settato un messaggio di errore nella pagina di configurazione e viene ritentata la connessione con tutte le reti salvate un numero arbitrario di volte.
Si è scelto questo approccio in quanto, se la ricezione è particolarmente debole, può capitare che alcuni tentativi di connessione non vadano a buon fine quando comunque le credenziali sono corrette. Se la connessione avviene, viene rimosso il messaggio di errore e si prosegue con la configurazione. in caso contrario, superato il numero di tentativi di connessione stabiliti, viene riavviato il device. Le principali causa di questo errore possono essere

- Credenziali errate
- Ricezione del segnale WiFi insufficiente

Nel secondo caso, la connessione viene persa dopo un primo collegamento avvenuto con successo. In questo caso è possibile tentare di riconnettere il dispositivo alle rete. La funzione implementata che gestisce gli eventi collegati al WiFi reagisce ai seguenti eventi nel modo descritto:

- `SYSTEM_EVENT_STA_START` che viene lanciato dopo l'inizializzazione della modalità station e chiama la funzione `esp_wifi_connect()` per inizializzare la connessione
- `SYSTEM_EVENT_STA_GOT_IP` che viene lanciato a connessione avvenuta
- `SYSTEM_EVENT_STA_DISCONNECTED` che viene lanciato quando viene persa la connessione con la rete, quando viene chiamata la funzione `esp_wifi_disconnect()` o `esp_wifi_stop()` e quando la chiamata alla funzione `esp_wifi_connect()` non viene conclusa con successo

Questo ultimo evento è stato usato per rilevare e gestire le disconnessioni. Dato che questo evento può essere lanciato anche durante le fasi precedenti, è stata usata una variabile booleana chiamata `connection_established` che viene settata a true nella handler dell'evento `SYSTEM_EVENT_STA_GOT_IP` in modo da identificare l'avvenuta connessione. Quando viene lanciato l'evento di disconnessione e la variabile connection established è a true, viene lanciato un thread che esegue la funzione `wifiDisconnectionTask()` che fa avviare un timer di 30 secondi e, passato quel tempo , viene testata una variabile che indica se è stata ripristinata la connessione, in caso positivo il thread termina altrimenti il dispositivo viene riavviato.
E' stato usato questo approccio per evitare il completo riavvio del dispositivo nel caso di disconnessioni spurie e momentanee ma per garantire comunque il riavvio in caso di disconnessione prolungata così da poter ritentare dal principio la procedura di connessione usando tutte le reti salvate.
Il thread principale intanto distacca il thread appena generato e tenta nuovamente una connessione che se andrà a buon fine risulterà nella continuazione dell'esecuzione, altrimenti verrà nuovamente lanciato l'evento `SYSTEM_EVENT_STA_DISCONNECTED`. Questo meccanismo è realizzato tramite due variabili booleane thread safe, implementate tramite il costrutto `std::atomic`. La prima è chiamata `wifi_disconnection_thread_running` e identifica se è stato già lanciato il task di gestione della disconnessione. La funzione di questa variabile è evitare di lanciare un nuovo thread ogni volta che viene lanciato l'evento

`SYSTEM_EVENT_STA_DISCONNECTED` venendo settata a true quando viene lanciato il thread `wifiDisconnectionTask` e settata a false quando questo termina.
La seconda è `wifi_connection_lost` che indica se è stata persa la connessione. Passati 30 secondi all'interno di `wifiDisconnectionTask`, questa è la variabile che viene testata. Viene settata a false quando è lanciato l'evento `SYSTEM_EVENT_STA_GOT_IP` che indica l'avvenuta connessione e settata a true in seguito all'evento `SYSTEM_EVENT_STA_DISCONNECTED` che indica una disconnessione.
Le gestione della disconnessione dal broker MQTT viene gestita nello stesso modo, l'unica differenza sta nel fatto che in questo caso viene disattivata la modalità promiscua per evitare di saturare la memoria del dispositivo con pacchetti che non vengono inviati, e riattivata quando viene ristabilita la connessione.

4.6 Deploy

Il sistema è stato concepito con l'intento di rendere la messa in opera un'operazione semplice e veloce. Per quanto riguarda gli sniffer è necessario che sia disponibile una rete Wi-Fi che permetta il collegamento ad internet o ad una rete Lan che permetta di raggiungere il broker e i vari servizi necessari al funzionamento. Al momento dell'accensione, se sono ancora presenti in memoria delle credenziali di accesso ad una rete, verrà tentata la connessione ed in caso di esito negativo si avvieranno i processi di recovery descritti precedentemente. In questo lasso di tempo un operatore più accedere alla pagina di configurazione degli sniffer collegandosi all'access point messo a disposizione dello sniffer che avrà come SSID il mac dello sniffer seguito dalla dicitura "_CONFIG". La password è definita all'interno del firmware e al momento non è possibile cambiarla. Una volta collegato è necessario raggiungere l'indirizzo 192.168.4.1 per accedere alla pagina di configurazione ed impostare i valori richiesti. A questo punto, sempre tramite l'interfaccia web è possibile riavviare il device che da questo momento, salvo malfunzionamenti o impossibilità di comunicazione di rete, può operare in maniera totalmente autonoma.

Capitolo 5

Implementazione del back end

5.1 Il framework Spring e Spring Boot

Da alcuni anni lo standard de facto per la realizzazione di applicativi web è rappresentato dal framework Java Spring. Spring è stato creato principalmente come container per Dependency Injection, cioè un pattern secondo il quale le varie dipendenze di un oggetto vengono soddisfatte non delegando il compito all'oggetto stesso ma attraverso l'iniezione della dipendenza dall'esterno. Il framework è delegato alla gestione del ciclo di vita di questi componenti e quindi anche a soddisfarne le dipendenze.
Questo approccio, dove è lo stesso framework a farsi carico della gestione degli oggetti e di parti del programma è chiamato Inversion of Control. In Spring l'Inversion of Control è gestita dall'ApplicationContext che è responsabile di istanziare, configurare ed assemblare gli oggetti, soprannominati beans, e gestirne il ciclo di vita. Le specifiche dei singoli beans possono venire definite in diversi modi: file XML, Java annotations o attraverso classi di configurazione. Per ottimizzare e velocizzare lo sviluppo di applicativi web, il team di sviluppo di Spring ha creato una versione preconfigurata del framework che provvede a fornire al programmatore un insieme di bean già configurati e pronti all'uso secondo il principio "Convention over Configuration", integrando il tutto con l'application server Tomcat in modo da rendere l'applicativo deployabile ovunque come un semplice file jar, dando origine a Spring Boot. Gli applicativi Spring Boot fanno uso di alcune annotazioni specifiche. A partire dalla classe di bootstrap, cioè quella che contiene la chiamata al metodo `SpringApplication.run()` che ha il compito ed avviare l'applicativo, possiamo trovare `@SpringBootApplication`. Questa annotazione non fa altro che riunire altre tre annotazioni: `@Confguratio`, `@EnableAutoConfiguration` e `@ComponentScan`. Queste tre annotazioni hanno il compito di attivare la configurazione automatica di SpringBoot descritta precedentemente e rilevare gli altri componenti dichiarati all'interno dell'applicativo.
Per supportare lo sviluppo di microservizi sfruttando l'ecosistema Spring, è nato il progetto Spring Cloud, che offre soluzioni ai più comuni problemi che si presentano durante lo sviluppo di sistemi distribuiti. I componenti di Spring Cloud che sono stati usati sono Spring Cloud Config ed Eureka, Zuul e Ribbon che fanno parte del progetto Spring Cloud Netflix che verranno descritti in seguito.

5.1.1 Le annotazioni del framework Spring

Per creare classi gestite dal framework, è possibile usare l'annotazione `@Component`. Una classe così annotata verrà automaticamente rilevata grazie all'uso di `@ComponentScan` e gestita dal framework. In certi casi, tuttavia, non è possibile sfruttare i meccanismi automatici di Spring ed è necessario dichiarare i bean manualmente tramite `@Bean`. In questo modo possiamo anche separare la dichiarazione della classe da quella del bean. Infatti `@Component` è un annotazione a livello di classe mentre `@Bean` viene usata a livello del metodo che istanzia e ritorna l'oggetto che andrà a costituire il bean. Come detto precedentemente, uno degli aspetti di maggiore interesse di Spring è di fungere da container che gestisce vari componenti secondo il principio dell' Inversion of Control. Per gestire la dipendenze con i bean gestiti dal framework, viene utilizzata l'annotazione `@Autowired` che indica al framework che vogliamo che venga risolta una certa dipendenza. Ad esempio all'interno della classe A abbiamo bisogno di un bean della classe B. A livello di codice potremo indicare la dipendenza in vari modi tra cui a livello di attributo della classe o a livello di costruttore.

```
class A {
    @Autowired
    B b;
}

class A {
    B b;
    @Autowired
    a( B b ){
        this.b = b;
    }
}
```

Il framework mette anche a disposizione l'annotazione `@Value` che permette di iniettare all'interno dei bean dei valori provenienti da risorse esterne, solitamente un file di configurazione. Ad esempio, nel caso volessimo assegnare ad una chiave di cifratura simmetrica il valore presente nel file di configurazione nel campo `security.key` andremo ad indicare a livello dell'attributo della classe di nostro interesse la mappatura del valore nel seguente modo:

```
@Value("${security.key}")
String signingKey;
```

Tramite questa annotazione è anche possibile indicare valori (o array di valori separati da una virgola) di default nel seguente modo:

```
@Value("${security.key:secretKey}")
String signingKey;
```

5.2 Spring MVC

5.2.1 Il pattern MVC

Tra le varie librerie che vengono messe a disposizione da Spring, Spring MVC è quella di cui è stato fatto un uso maggiore per la realizzazione del back end. Questo framework è utilizzato per realizzare applicativi secondo il pattern MVC.
MVC sta per Model View Controller e consiste in un pattern architatturale che prevede una suddivisione dei vari componenti software che compongono un programma per ottenere una distinzione tra la logica di presentazione e la logica di business. Il model si occupa della gestione dei dati, il controller riceve comandi dall'utente (potenzialmente tramite la view) e agisce di conseguenza sugli altri due moduli, modificandone lo stato e la view si occupa della presentazione dei dati come evidenziato nell'immagine 5.1.

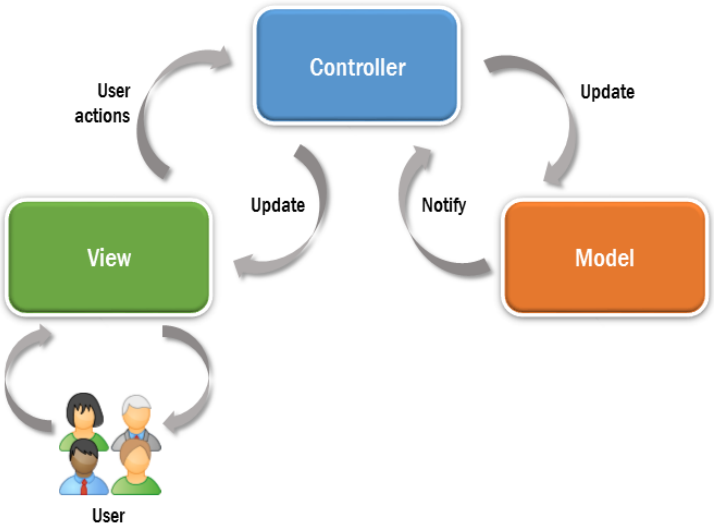


Figura 5.1: Il pattern MVC

5.2.2 MVC nel contesto di Spring

Nel contesto di Java Spring questo pattern è espresso nella seguente declinazione:

- Il model è rappresentato dalle classi POJO che rappresentano le entità gestite dall'applicativo.
- II controller sono le classi con il compito di rispondere alle richieste dell'utente attraverso metodi che "stanno in ascolto" su specifiche url. Ulteriori classi chiamate services implementano la logica effettiva, questo per creare un'ulteriore distinzione logica tra i metodi che gestiscono le richieste e la logica sottostante.
- Infine le view sono solitamente rappresentate tramite pagine JSP che vengono compilate in file Html al momento della richiesta e che vengono inviate come risposta in seguito all'invocazione di uno specifico controller

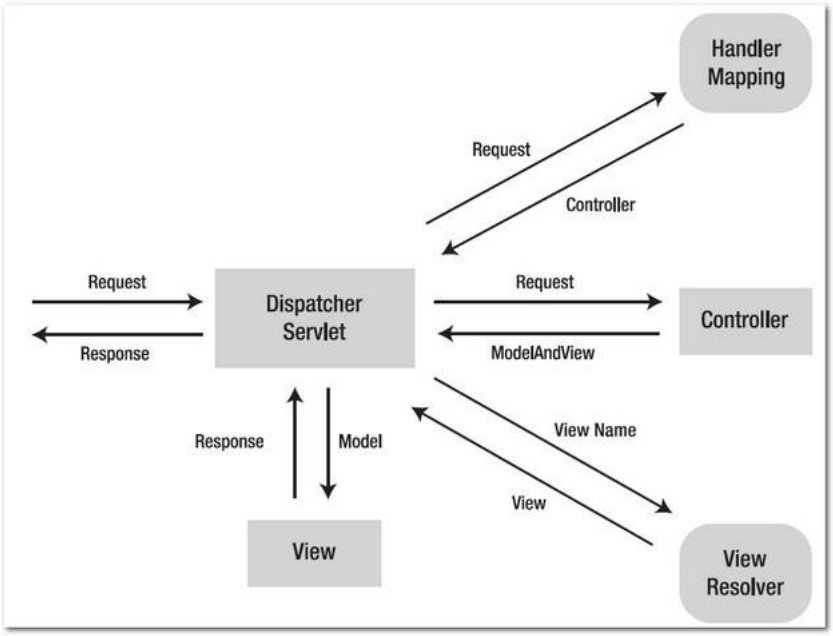


Figura 5.2: Il pattern MVC nel contesto di Spring

Nell'immagine 5.2 è descritto il ciclo di vita di una richiesta:

- La richiesta pervenuta all'application server su cui è in esecuzione l'applicativo viene inviata al Dispatcher Servlet che ha il compito di indirizzare la richiesta al controller corretto consultando l'Handler Mapping
- Selezionato il giusto controller in grado di soddisfare la richiesta, il dispatcher servlet attende che questo ritorni fornendo i dati richiesti.
- Viene invocato il View Resolver che ha il compito di mappare il nome della view che è indicata per visualizzare i dati ritornati dal controller con la pagina JSP corrispondente.
- Selezionata la view corretta, viene compilata la pagina JSP ed inviata all'utente.

Nel nostro caso, è stato usato un approccio leggermente diverso in quanto i dati che vogliamo che siano ritornati in seguito ad una richiesta non sono pagine web ma file JSON che contengono i dati richiesti. Questo è dovuto al fatto che l'applicativo che farà da front end si occuperà della logica di presentazione, necessitando soltanto di ricevere i dati da visualizzare.

5.2.3 Le annotazioni di Spring MVC

Di seguito verranno descritte alcune delle annotazioni che sono state maggiormente utilizzate ed il loro scopo all'interno dell'applicativo. Le classi controller vengono annotate con `@RestControllerche` a sua volta integra le annotazioni `@Controller` e `@ResponseBody`. La

prima è una versione specializzata dell'annotazione `@Component`descritta precedentemente, la seconda permette di automatizzare la serializzazione dell'oggetto ritornato dai metodi che compongono il controller, solitamente in formato JSON.
Un'altra annotazione utilizzabile sia a livello di classe che di metodo è `@RequestMapping` che indica al framework quale sarà la url mappata ed il tipo di metodo http utilizzato dalla richiesta. Espresso a livello di casse implica che tutte le ulteriori url specificate a livello di metodo saranno relative a quella specificata a livello di classe. Ad esempio avendo `@RequestMapping("/a")` a livello di classe, se vogliamo raggiungere il metodo con `@RequestMapping("/b")` dovremo contattare la url `/a/b`.
Possiamo anche attuare meccanismi di autorizzazione a livello di classe o metodo tramite l'annotazione `@PreAuthorize` ed esprimere quale authority o insieme di esse possa effettivamente chiamare un metodo del controller. Questi aspetti verranno approfonditi nel capitolo dedicato all'implementazione del servizio che gestisce la sicurezza dell'applicativo.
Il framework Spring MVC mette a disposizione anche diverse annotazioni utili a recepire dati provenienti dal client, a livello di Url e come parametri della richiesta Get e del body in caso di POST. Come descritto precedentemente, il paradigma REST prevede una suddivisione logica delle Url in modo tale da descrivere la risorsa richiesta. Ad esempio se volessimo accedere ai dati dell'utente 123 contatteremmo con una richiesta Get la url `/utenti/123`. Grazie all'annotazione `@RequestMapping` è possibile indicare 123 come parametro che potrà essere utilizzato dal metodo chiamato successivamente mappando la variabile su uno dei parametri della funzione grazie all'annotazione `@PathVariable` che verrà posta prima della dichiarazione del parametro che avrà lo stesso nome indicato nella url. In questo caso avremo il metodo così annotato:

```
@RequestMapping("/utenti/{id}", method = requestMethod.GET)
User getUser(@PathVariable String id){
...
}
```

Per recuperare le variabili contenute nel body della richiesta, nel caso di POST o PUT, viene usata la notazione `@RequesBody`, posta prima della dichiarazione del parametro del metodo che dovrà avere lo stesso nome del parametro contenuto nel body.

```
@RequestMapping("/utenti", method = requestMethod.POST)
void addUser(@BodyVariable User user){
    ...
}
```

Ultimo modo per passare parametri alle funzioni del controller è attraverso i parametri di una richiesta GET. In questo caso viene posta, sempre prima della dichiarazione del parametro, l'annotazione `@RequestParameter`. Come negli altri casi il parametro dovrà avere lo stesso nome della variabile presente nella url per poter permettere la mappatura automatica. Nel seguente esempio viene descritto il prototipo di una ipotetica funzione che ritorna gli utenti con una certa età, con il seguente url: `/users?age=20`.

```
@RequestMapping("/utenti", method = requestMethod.GET)
List<User> getUsers(@RequestParam int age){
    ...
}
```

Come detto precedentemente, le classi controller hanno il compito di ricevere le richieste dell'utente e rispondere con l'oggetto o la lista di oggetti richiesti. Tuttavia per mantenere una maggiore leggibilità e mantenibilità del codice è preferibiile implementare l'effettiva logica applicativa in classi separate, denominate servizi che, sfruttando i meccanismi di wiring di Spring, vengono iniettate dentro le classi controller. Queste classi, per ragioni di convenzione e norme di buona scrittura, vengono descritte come interfacce ed implementate separatamente.
Le classi che implementano queste interfacce vengono annotate con `@Service` che rappresenta una specializzazione dell'annotazione `@Component` come nel caso di `@Controller`. All'interno di queste classi viene implementata la logica applicativa. Tramite i meccanismi di wiring descritti precedentemente vengono iniettate all'interno di queste classi altre classi specializzate chiamate repository che hanno il compito di fungere da interfaccia tra i servizi e il database implementando quello che è il layer view e model. La descrizione di queste classi verrà effettuata in seguito.

5.3 Spring Data

Spring mette a disposizione un potente insieme di librerie che permettono l'interazione con diversi sistemi di basi di dati tramite un insieme di api consistenti e familiare con il modello di programmazione di Spring. Queste librerie rientrano sotto il progetto Spring Data, e nel nostro caso è stato fatto uso di Spring Data MongoDB. Spring Data permette di creare implementazioni specifiche di classi DAO tramite la descrizione di interfacce secondo precisi criteri. Queste interfacce prendono il nome di repository e permettono solitamente di ottenere in modo dichiarativo implementazioni di metodi CRUD, sistemi di paginazione e di ordinamento rispettivamente estendendo `CrudRepository` e `PagingAndSortingRepository`. Ad esempio la seguente interfaccia permette di reperire nel database quanti utenti hanno il cognome specificato.

```
interface UserRepository extends CrudRepository<User, Long> {
    long countByLastName(String lastname);
}
```

Il framework è in grado, interpretando il nome del metodo e di creare un'implementazione che soddisfi le condizioni indicate nell'interfaccia. In questo modo è possibile creare query anche di discreta complessità, combinando condizioni sugli attributi degli oggetti.

5.3.1 I repository Spring Data MongoDB

Nel caso di Spring Data MongoDB, viene estesa l'interfaccia `MongoRepository`, che a sua volta è una specializzazione di quelle indicate precedentemente. La configurazione necessaria all'accesso al database è indicata all'interno del file di configurazione del singolo servizio. L'interfaccia che descrive il repository deve essere annotata con `@Repository` per permettere al framework di capirne la funzione. Di seguito è riportata come esempio la classe `UserRepository` utilizzata da User-Service.

```
@Repository
public interface UsersRepository extends MongoRepository<User, String> {
```

```
    Optional<User> findByUsername(String username);
    Boolean existsByUsername(String username);
    Boolean existsByMail(String email);
    Long deleteByUsername(String username);
}
```

5.3.2 MongoTemplate

Per quanto tramite repository sia possibile effettuare la maggior parte delle query, per poter accedere alla funzionalità fornite dall'aggregation framework è necessario utilizzare un altro strumento messo a disposizione da Spring Data, la classe `MongoTemplate`. Questa classe fornisce alcune feature estese rispetto ai repository e più vicine alle API native di MongoDB.
Per utilizzare mongo template all'interno dei servizi è necessario dichiarare una variabile di istanza della classe con tipo Mongotemplate ed annotare con `@Autowired`. La funzione `aggregate()` del template accetta come parametri un oggetto `Aggregation` che a sua volta contiene un insieme variabile di oggetti che rappresentano i vari stadi della pipeline nell'ordine in cui devono essere effettuati, il nome della collection su cui effettuare l'aggregazione, ed una classe su cui verranno mappati i risultati. La funziona ritorna un oggetto `AggregationResult` che contiene al suo interno un campo `rawResults` che rappresenta i dati grezzi ritornati dal database e `mappedResult` che corrisponde ai rawResult mappati sulla classe indicata precedentemente.
Per permettere il funzionamento di tutto questo, anche le classi POJO che rappresentano gli oggetti trattati dall'applicativo devono essere opportunamento annotate tramite `@Document` che permette di indicare come parametro a quale collection appartengono quegli oggetti. Se il parametro rimane vuoto viene usato il nome della classe.
Ogni oggetto inserito all'interno del database deve avere un campo `_id` che rappresenta un identificativo univoco di quel documento. Pertanto nella classe che rappresenta l'oggetto è necessario inserire anche questo attributo e annotare con `@Id`.
E' importante sottolineare che per permettere una mappatura automatica dei campi del documento prelevato dal db e l'oggetto java, stessi attributi dovranno avere lo stesso nome. In caso contrario è possibile usare l'annotazione `@Field` che indicherà come parametro il nome usato per quella variabile nel db. E' utile sottolineare a questo punto come è stata gestita la validazione dei dati prima di renderli persistenti nel db. Spring Boot usa il framework di validazione di Hibernate e pertanto necessario aggiungere tra le dipendenze del progetto anche quella a `spring-boot-starter-jpa`. Seguendo la filosofia di Spring Boot, i criteri di validazione di un dato vengono descritti da alcune annotazioni apposte nella definizione dei parametri di ogni classe che ne necessita. Possono essere imposti vari criteri ad esempio la lunghezza massima o minima di una stringa tramite `@Size`, o la non nullità di un valore tramite `@NotNull` o che una stringa non sia volta tramite `@NotEmpty`. Può essere anche indicato un messaggio che verrà inviato al client per specificare quale condizione o insieme di tali non è stata rispettata. Per imporre questi criteri è necessario aggiungere l'annotazione `@Valid` a livello dei metodi del controllore che riceve i dati da validare.

5.4 Configuration Service

Prima dell'avvio di ognuno dei servizi, è necessario reperire la relativa configurazione. Solitamente un applicativo creato con Spring Boot viene configurato attraverso un file YAML chiamato application.properties.yml mentre la configurazione necessaria per il bootstrap del servizio è fornita dal file application.bootstrap.yml. Nel caso si volesse cambiare alcuni dei parametri sarà necessario andare a modificare il file e ricreare il file .jar che contiene la nostra applicazione.
Tuttavia questo approccio è poco scalabile quando si ha a che fare con tante applicazioni e per questo il framework Spring Cloud mette a disposizione un componente che funge da server di configurazione. Questo componente permette di centralizzare la configurazione di tutti i servizi che fanno parte dell'applicazione e di distribuire tali configurazioni in modo automatico ogni volta che un servizio viene avviato.
Come molti altri servizi facenti parte del framework Spring Cloud, per realizzare il Configuration Server è sufficiente inserire nella classe di bootstrap del servizio l'annotazione `@EnableConfigServer`.
Nel file di configurazione del servizio è possibile indicare il repository che conterrà le configurazioni dei vari servizi. Questo repository può essere remoto o locale e versionato con git. Nel nostro caso il repository git è in locale e acceduto dal container docker in cui è in esecuzione il servizio tramite un volume condiviso con l'host del client Docker. All'interno del repository sono presenti i file di configurazione corrispondenti a `configuration.properties` di ogni servizio, ma per permetterne l'identificazione da parte del server, il nome del file segue lo schema `nomeDelServizio-profilo.yml` dove profilo indica la versione della configurazione desiderata, ad esempio quella di produzione (prod) o quella di sviluppo (dev). All'interno dei file `application.bootstrap` dei singoli servizi è indicato l'indirizzo a cui è reperibile il server di configurazione in modo tale che all'avvio venga richiesta la configurazione specificata dall'opzione `spring.profiles.active` che può anche essere indicata come variabile d'ambiente dall'interno del file docker-compose. Dato che senza l'opportuna configurazione è impossibile continuare l'esecuzione di un servizio, nel caso in cui il server fosse irraggiungibile o non fosse presente la configurazione richiesta, il servizio verrebbe terminato. Pertanto è necessario implementare dei meccanismi che prevengano questi casi che verranno approfonditi successivamente.

5.5 Eureka

Eureka è il componente che funge da service registry, cioè un registro in cui i vari servizi si registrano indicando il loro indirizzo fisico, la porta su cui sono in ascolto ed un service ID, rendendo così possibile ad altri servizi di interagire con questi senza dover conoscere a prescindere la loro posizione. Questo approccio prende il nome di service discovery ed è di estrema importanza nelle architetture distribuite: dal momento che per utilizzare un servizio non è necessario conoscerne la posizione effettiva ma solo il nome, è possibile aggiungere o togliere istanze di un certo servizio pur mantenendolo sempre raggiungibile, in questo modo aumentando la flessibilità e la scalabilità e permettendo di aumentare la resilienza dell'applicazione in caso di malfunzionamento di un servizio dato che la richiesta fallita può essere dirottata su di una replica funzionante dello stesso servizio.

Per creare un'istanza di Eureka, come per molti altri componenti del framework, è sufficiente inserire l'annotazione `@EnableEurekaServer` all'interno della classe di bootstrap del nostro servizio. In questo modo, alla porta selezionata (solitamente 8761), sarà possibile accedere ad un'interfaccia grafica che contiene tutte le informazioni relative ai servizi registrati. La registrazione di un servizio con Eureka viene effettuato attraverso il file di configurazione.

5.5.1 Registrare i servizi con Eureka

Per registrare un servizio nel registry è necessario impostare alcuni campi all'interno del file di configurazione, come indicato nel seguente listato.

```
eureka:
    instance:
        preferIpAdress: true
    client:
        registerWithEureka: true
        fetchRegistry: true
        serviceUrl:
            defaultZone: http://eureka-service:8761/eureka/
```

I campi di maggiore importanza sono:

- `registerWithEureka`: indica che intendiamo registrare il servizio.
- `serviceUrl.defaultZone`: contiene una lista di istanze di eureka che verranno usate per individuare la posizione del servizio richiesto. Nel nostro caso corrisponde anche all'istanza in cui si registrerà il servizio.

Il nome con cui sarà possibile individuare il servizio richiesto sarà quello indicato nel campo `spring.application.name` del file di configurazione.
Dal momento che è stato usato Docker, nel file di configurazione di ogni servizio che intende registrarsi con eureka, è stato necessario inserire come indirizzo dell'istanza di eureka il nome del servizio eureka indicato nel file docker-compose che istanzia l'applicazione. Infatti docker è in grado di risolvere questo nome e indirizzare le richieste provenienti dagli altri container dal momento che questi sono connessi alla stessa sottorete.

5.5.2 Interrogare il service registry

Per effettuare il lookup di un altro servizio, il framework mette disposizione diverse librerie che permettono di interagire in diversi modi con il service registry: SpringDiscoveryClient e SpringRestTemplate (con o senza Ribbon, per bilanciare le richieste) Netflix Feign che implementa un client REST dichiarativo, che permette di effettuare richieste semplicemente annotando delle interfacce Java.
Dato che non sono presenti particolari necessità di comunicazione tra i vari servizi, è stato usato il primo metodo. Ciò consiste nell'annotare la classe di bootstrap del nostro servizio con `@EnableDiscoveryClient`. A questo punto è possibile invocare nei servizi desiderati il bean DiscoveryClient che tramite il metodo `getInstances()` permette di ottenere la lista delle istanza di un certo servizio (sottoforma di oggetti ServiceInstance) il cui nome viene

indicato come parametro della funzione citata precedentemente.
L'oggetto ServiceInstance ottenuto permette di ricavare tutte le informazioni necessarie a contattare il servizio prescelto, come l'indirizzo e la porta su cui è in ascolto il servizio.

5.6 Zuul Proxy

L'utilizzo di architetture distribuite implica la necessità di dover utilizzare un punto di ingresso comune che funga da facciata nascondendo l'insieme di microservizi sottostanti, ricevendo tutte le richieste e inoltrandole ai servizi richiesti. Questo pattern prende il nome di Api Gateway. Tra i componenti del progetto Spring Cloud Netflix è presente un servizio che si occupa proprio di questo: Zuul Gateway. Come gli altri servizi resi disponibili dal framework, avviare un proxy Zuul richiede semplicemente un annotazione `@EnableZuulProxy` nella classe che implementa il main del nostro servizio. In questo modo per poter interagire con il servizio è sufficiente modificare i file di configurazione.

5.6.1 Configurazione del servizio

Oltre ad indicare la porta su cui sarà in ascolto il servizio, è necessario indicare l'indirizzo a cui sarà reperibile il server Eureka in modo che Zuul possa conoscere gli indirizzi dei vari servizi. Nel file di configurazione è possibile indicare come ogni servizio venga mappato su una specifica url secondo la sintassi `nomeServizio: /urlRelativaDelServizio/**`. Ad esempio se volessimo richiedere il token JWT sarà necessario fare una richiesta a `localhost:5555/auth/oauth/token`. Il proxy (che in questo caso è ospitato in locale) vedendo `/auth/` all'interno della url, dirotterà la richiesta. Di default, se non viene indicata nessuna mappatura manualmente, Zuul provvedere a mappare le url basandosi sul nome con cui i vari servizi sono registrati in Eureka. Se si volesse usare dei nomi diversi è necessario indicarli nel file di configurazione ma questo disattiva la mappatura automatica dei servizi, pertanto se si andrà a richiedere un servizio non specificato manualmente verrà lanciata un'eccezione.
Zuul implementa il client side load balancing grazie a Ribbon e meccanismi di circuit breaker usando Hystrix, entrambi facenti parti del progetto Spring Cloud Netflix. Il primo permette di implementare meccanismi di load balancing, cioè suddividere le richieste provenienti dall'esterno tra più repliche dello stesso servizio per garantire una suddivisione del carico. Hystrix implementa un circuit breaker, cioè un componente che ha il compito di rilevare eventuali problemi con un servizio e "tagliare il circuito" dirottando le richieste verso un servizio alternativo e verificare se quello precedente è ritornato disponibile.

5.6.2 Gestione delle richieste CORS

CORS (Cross-Origin Request Sharing) è un meccanismo che permette ad un applicativo web in esecuzione su di un certo dominio di richiedere risorse provenienti da un'origine (protocollo, porta o indirizzo) differente dalla propria. Questo meccanismo serve a limitare le richieste generate all'interno di script ed indirizzate a risorse esterne all'origine di caricamento dell'applicazione che solitamente vengono bloccate dai browser, a meno che il server a cui viene fatta una richiesta preliminare includa all'interno della risposta i corretti header CORS. Questo meccanismo viene solitamente implementato effettuando una chiamata http

OPTIONS preliminare alla richiesta vera e propria chiamata preflight request, che richiede i metodi supportati dal server. All'interno della risposta è indicato l'insieme di origini che sono autorizzate ad accedere alle risorse richieste.
Dato che il gateway riceve tutte le richieste è stato necessario implementare una classe in grado di gestire correttamente questo tipo di richieste. Il framework mette a disposizione l'interfaccia Filter per implementare classi in grado di intercettare e modificare i parametri delle richieste ricevute.
ll filtro CORS implementato utilizza le annotazioni `@Component` e `@Order` che perme, ricevendo come parametro `Ordered.HIGHEST_PRECEDENCE`, di applicare questo filtro prima di ogni altro. Questa classe, come detto precedentemente, deve implementare l'interfaccia Filter, specialmente il metodo `doFilter()` che implementa la logica del filtro. Innanzi tutto vengono settati alcuni header del tipo `Access-Control`. Nello specifico l'header `Access-Control-Allow-Origin` indica l'insieme di origini con cui il server può condividere le sue risorse. Nel nostro caso si è lasciato il valore '*' che indica che il server condivide le sue risorse con qualsiasi dominio. Altri header sono `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers`, `Access-Control-Max-Age` che indicano rispettivamente i metodi e gli header supportati per la richiesta vera e propria conseguente e quanto a lungo il risultato della preflight request può essere mantenuto in cache.
Il filtro intercetta qualsiasi richiesta pervenuta e verifica il metodo utilizzato, se è OPTIONS invia la risposta contenente gli headers indicati precedentemente mentre se il metodo usato è diverso, la richiesta viene inoltrata ai filtri successivi ed indirizzata al servizio destinatario.

5.7 Security Service

Authentication è il servizio che si occupa di fornire e verificare i token JWT. Questo servizio è stato costruito usando il framework Spring Security e funziona secondo lo standard Oauth2 fungendo da Authorization Server. Questo prevede un sistema di autenticazione ed autorizzazione basato su token (nel nostro caso secondo lo standard JWT) rilasciati da un trusted third party, l'Authorization Service. Quando un client si autentica con successo, l'authorization server gli fornisce un token che verrà usato per autenticare ogni richiesta successiva effettuata ad un resource server, in questo modo sarà possibile effettuare delle richieste senza dover presentare di volta in volta le credenziali ma solo il token. Il resource server a sua volta potrà richiedere all authorization server la verifica del token.
Le possibilità offerte da Spring Security sono moltissime e permettono una gestione molto sofisticata ed approfondita della sicurezza delle nostre applicazioni.
Il cuore del funzionamento del servizio risiede all'interno di alcune classi di configurazione che permettono di definire i principali aspetti del funzionamento di Spring Security. All'interno di queste classi vengono anche definiti i bean che implementeranno i vari componenti del sistema di autenticazione come il password encoder, token store e l'authentication manager.
Dato che il servizio, secono lo standard Oauth2, funge sia da authorization server che da resource server, la classe di bootstrap del servizio sarà annotata con `@EnableResourceServer` e `@EnableAuthorizationServer`.

5.7.1 La classe AuthorizationServerConfigurer

La classe `AuthorizationServerConfigurer` estende `AuthorizationServerConfigurer-Adapter`. All'interno della classe sono presenti due metodi `configure()` che effettuano l'override dei relativi metodi della classe estesa.
Il primo accetta un parametro di tipo `AuthorizationServerEndpointsConfigurer` e serve a definire i componenti utilizzati dall'`AuthenticationServerConfigurer` cioè il token service, l'authentication manager, il token converter e lo user details service che vengono iniettati all'interno del costruttore della classe.
Il secondo metodo `configure()` accetta invece un parametro di tipo ClientDetailServiceConfigurer e permette di indicare quali saranno i tipi di client che potranno accedere alla nostra applicazione. Nel nostro caso il l'unico client che può accedere all'applicazione è il relativo frontend che quindi dovrà fornire un client id ed una password per poter accedere all'endpoint di richiesta del token. E' utili sottolineare che queste credenziali fornite non hanno nulla a che fare con quelle relative ad un utente che cerca di autenticarsi ma sono solo necessarie ad autenticare il client da cui provengono le richieste.

5.7.2 La classe WebSecurityConfigurer

La classe `WebSecurityConfigurer` estende `WebSecurityConfigurerAdapter` ed è usata per definire come gli utenti saranno autenticati e come avverrà l'autenticazione del client. All'interno della classe vengono istanziati tre bean: `AuthenticationManager`, `Password-Encoder` e `UserDetailService`.
`AuthenticationManager` è un'interfaccia usata per effettuare l'autenticazione dell'utente ed è necessario per configurare l'end point `auth/oauth/token` che è utilizzato per il reperimento del token. Nel nostro caso si è utilizzato l'implementazione nativa di Spring: una volta ricevute le credenziali, l'Authentication Manager le passerà all'Authentication Provider che sfrutterà lo User Datail Service per reperire nel database le credenziali dell'utente e verificarne la correttezza. `PasswordEncoder` è il bean in cui viene definito il tipo di codificatore che verrà usato per ottenere la password nel formato in cui verrà salvata nel database. Nel nostro caso è stato usato `BcryptPasswordEncoder` che sfrutta la funzione di hashing bcrypt, basata sull'algoritmo di cifratura a blocchi Blowfish. La stringa risultante include sia l'hash della password che il sale utilizzato, come da esempio:

$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy

La stringa di output risulta composta dai seguenti campi:

- Il prefisso $2a$ indica il che la stringa è nel formato bcrypt
- 10 indica il parametro di costo
- I seguenti 22 caratteri corrispondono al sale di 128 bit codificato in formato base64
- Gli ultimi 31 caratteri corrispondono all'hash della password codificata in formato base64

All'interno del database viene salvata questa stringa e la stessa viene usata per testare le credenziali inserite dell'utente che cerca di ottenere il token.
`UserDatailService` è un'interfaccia che viene usata dal framework per gestire le informazioni

dell'utente. Va implementata in base al database che si è scelto di utilizzare e permette di ottenere le informazioni attraverso un oggetto di tipo `UserDetails` chiamando il metodo `loadUserByUsername()` che accetta come parametro una stringa contenente lo username. Tramite i repository di Spring Data, la funzione reperisce dal database il documento contenente i dati dell'utente e li inserisce nell'oggetto `userDetails` che ha come attributi due stringhe che rappresentano lo username e la password in formato bcrypt e una lista contenente le authorities dell'utente. Nel caso non ci fosse un riscontro con gli utenti registrati nel database verrà lanciata un'eccezione del tipo `UsernameNotFoundException`. All'interno della classe sono implementati due metodi `configure()`. Il primo accetta come parametro un oggetto del tipo `AuthenticationManagerBuilder` che ha il compito di configurare l'`AuthenticationManager` indicando lo `UserDetailsService` utilizzato e il `PasswordEncoder`. Il secondo accetta un oggetto del tipo `HttpSecurity` e permette di definire come il client dovrà autenticarsi per poter richiedere il token. In questo caso è stata usata la http basic autentication.
La classe `JWTTokenStoreConfig` è deputata alla definizione di come Spring genererà, firmerà e verificherà i token JWT. All'interno di questa classe il metodo di maggiore importanze è sicuramente `jwtAccessTokenConverter()` che definisce come il token verrà tradotto, e quale sarà la chiave di cifratura simmetrica che verrà usata per firmare i token e consentirne la verifica. La chiave è una stringa casuale definita nel file di configurazione del servizio e che dovrà essere condivisa con tutti i servizi che riceveranno il token.
Dato che, oltre a fornire il token di autenticazione, il servizio ospita alcuni endpoint che verranno descritti in seguito, il servizio stesso rientra nella categoria di resource server e pertanto bisogna definire le regole di sicurezza che intendiamo usare per accedere al servizio.

5.7.3 La classe ResorceServerConfigurer

L'ultima classe di configurazione è `ResourceServerConfigurer` che estende `ResourceServerConfigurerAdapter` (questa classe è molto simile a quella che troveremo negli altri resource server, cioè idealmente tutti i servizi che espongono degli endpoint protetti). La classe implementa il metodo `configure()` che riceve come parametro un oggetto `HttpSecurity`. All'interno di questo metodo verranno definite le uri degli endpoint e come intendiamo proteggere tale risorsa. Il servizio espone l'endpoint `/sniffer` che potrà essere acceduto senza autenticazione mentre tutti gli altri necessitano di autenticazione.

5.7.4 Endpoint esposti

Dato che la sicurezza dell'applicazione è stata centralizzata in questo servizio, quando un altro servizio riceverà una richiesta ed un token sarà solamente in grado di verificarne la firma ma non di interpretare il contenuto del token. Pertanto è stato necessario esporre un endpoint che ricevesse il token e restituisse le informazioni relative all'utente e le sue authorities. Tale endpoint è definito nella classe `UserController` e permette di ottenere in risposta una mappa contenente lo username associato al token e la lista delle sue authorities. Questi dati verranno utilizzati dal servizio che ne ha fatto richiesta per verificare che l'utente esista e che abbia l'autorizzazione di compiere la richiesta effettuata.
Le authorities previste sono tre: User, Admin e Sniffer.

User è l'authority associata all'utente base, che può accedere ad un insieme ridotto di funzionalità, legate principalmente alla visualizzazione dei dati raccolti e dei propri dati personali.
Admin è l'authority con privilegio di accesso più elevato e può chiamare i metodi esposti da tutti gli endpoint.
Sniffer è l'authority associata all'utente che rappresenta uno sniffer e viene usato proteggere l'endpoint che permette di reperire la configurazione dello sniffer. L'ultimo endpoint esposto è `/sniffers` e viene usato dal broker MQTT tramite POST per verificare che il client che sta tentando di effettuare la connessione è registrato tra quelli abilitati. In caso le credenziali contenute nei parametri della richiesta siano corrette, il server invierà una risposta contenente il codice http 200 (Ok). Nel caso contrario la risposta sarà 401 (Unauthorized) ed il broker rifiuterà la connessione del nuovo client. A completare il servizio troviamo alcune classi che fungono da service e repository per implementare i meccanismi descritti in precedenza ma che non necessitano di un'ulteriore discussione.

5.8 Moquette Broker

MoquetteBroker è il servizio che implementa un broker MQTT sfruttando la libreria Moquette[9]. Questa libreria consente di integrare all'interno dei propri progetti un broker MQTT semplicemente aggiungendo le opportune dipendenze nel file `pom.xml`.
Il broker viene istanziato come un normale oggetto Java e la sua configurazione avviene attraverso un oggetto del tipo `MemoryConfig`. La classe `MoquetteBroker` è responsabile dell'inizializzazione e della gestione del ciclo di vita del broker facendone da wrapper. Per poter sfruttare le potenzialità del framework Spring, la classe è stata annotata con `@Component` ed implementa le interfacce `DisposableBean` e `InitializingBean` che, attraverso i metodo `afterPropertiesSet()` e `destroy()`, vengono usate per effettuare alcune operazioni in specifici punti del ciclo di vita del componente, rispettivamente dopo che le proprietà del bean saranno settate e prima che il bean venga distrutto.
`AfterPropertiesSet()` ha il compito di chiamare la funzione `start()` che verifica tramite una variabile booleana d'istanza, `started`, settata di default a false se il server è già attivo o no. In caso affermativo la funzione semplicemente ritorna mentre nel caso contrario viene iniziato il processo di configurazione del server. Il broker viene settato in modo tale da non permettere client non autenticati e viene indicata la classe `AuthenticationWrapper`. Per gli altri parametri vengono utilizzati i valori di default, principalmente la porta su cui sarà in ascolto il server è quella standard del protocollo, la 1883. Successivamente viene istanziato l'oggetto `Server` e inizializzato chiamando la funzione `startServer()` che accetta come parametro la configurazione definita in precedenza e viene settata a true la variabile che indica lo stato di attività del server. Da questo momento sarà possibile per i client registrarsi col broker ed iniziare ad inviare dati.

5.8.1 Avvio ed arresto del broker

All'interno della classe sono ancora presenti alcuni metodi che permettono di gestire il broker. Cercando di trovare un modo semplice ed efficace per interrompere la raccolta dei dati, è stata creato il metodo `stopServer()` che in caso `started` sia vera e quindi

il server sia attivo, chiama la funzione `stopServer()` dell'oggetto Server, fermando il broker e rilasciando tutte le risorse allocate, settando nuovamente a false la variabile `started`. I metodi `start()` e `stopServer()` sono raggiungibili tramite due endpoint, rispettivamente `/start` e `/stop` e permettono di avviare e disattivare il server secondo la necessità dell'utente. La variabile started serve ad impedire di istanziare più volte il server e evitare di chiamare il metodo `stopServer()` su un oggetto che corrisponde a `null`.
É presente anche il metodo `getServer()` che ritorna l'oggetto `Server`, la cui utilità verrà descritta in seguito.

5.8.2 Autenticazione dei client

Per gestire l'autenticazione dei client che tentano la connessione con il broker, la libreria mette a disposizione l'interfaccia `IAuthenticator`. La classe `AuthenticationWrapper`,che implementa tale interfaccia, deve fornire un metodo `checkValid()` che accetti come parametri l'Id del client MQTT ed una password in formato array di byte e ritorni una variabile booleana che valga true se il client può connettersi e false altrimenti.
La funzione inizialmente converte la password in una stringa UTF_8 e testa se l'id e la password non corrispondono ad una coppia di valori di default che viene usata per auteticare i servizi che implementano un client MQTT. In caso contrario, viene effettuata una richiesta al servizio Security tramite l'edpoint `/sniffers` che indicherà se le credenziali sono accettate o no. A tale scopo la classe `AuthenticationWrapper` mette a disposizione il metodo `getHttpRequestClient` che ritorna il bean `HttpRequestClient` richiedendolo direttamente allo SpingContext. Questo è necessario in quanto la classe `AuthenticationWrapper` non è gestita dal framework, e quindi è impossibile iniettare il componente `HttpRequestClient`.
La classe `HttpRequestClient` sfrutta il `DiscveryClient` per richiedere al service registry l'istanza di Security ed effettuare la richiesta all'endpoint necessario tramite la funzione `sendRequest()`. Nel caso l'operazione andasse a buon fine viene ritornato true, false altrimenti.

5.8.3 Individuazione dei client connessi

La classe `MoquetteController` espone gli endpoint necessari all'avvio e all'arresto del broker ed un terzo endpoint, mappato sulla url `/clients`. É possibile accedere, tramite la funzione `getServer()`, all'istanza dell'oggetto `Server` ed ottenere la lista di tutti i client connessi tramite il metodo `listConnectedClients()`. Il metodo ritorna una lista di oggetti `ClientDescriptor` che, tramite le Stream Api di Java, viene convertita in una lista contenente gli id dei client connessi. In questo modo è possibile visualizzare sull'applicazione client quali sono gli sniffer connessi e quindi funzionanti.

5.9 I servizi Subsriber Parsed e Subscriber Dump

I servizi subscriber parsed e subscriber dump hanno il compito di acquisire, elaborare e salvare nel database i dati ricevuti dagli sniffer.
Dal momento che questi due servizi utilizzano MQTT per ricevere i dati dagli sniffer, è stato necessario individuare una libreria che permettesse di implementare tale protocollo

all'interno del servizio. La scelta è ricaduta sulla libreria Eclipse Paho[9], un progetto open source che implementa un client MQTT secondo la versione 3.1.1 dello standard. Per includere la libreria è sufficiente aggiungere la relativa dipendenza al file pom.xml.
Dal momento che il client MQTT non è un componente di Spring, è necessario creare una classe gestita dal framework che faccia da wrapper, quindi procediamo similmente a quanto fatto per Moquette, inserendo la chiamata al metodo `context.getBean()` all'interno della classe di bootstrap.
Dato che verrà usato il discovery client, è necessario aggiungere anche l'annotazione `@EnableDiscoveryClient`.

5.9.1 Configurazione del client MQTT

La classe MQTTSubscriber è il componente che gestisce il ciclo di vito del client MQTT. La classe è annotata con `@Component` per far in modo tale che venga gestita dal framework ed implementa le interfacce `MqttCallback`, `DisposableBean` e `InitializingBean`. I parametri necessari alla configurazione del client MQTT sono prelevati dalla configurazione ricevuta dall'apposito servizio al momento dell'avvio e sono mappate sui membri della classe attraverso l'annotazione `@Value`.
I parametri sono:

- `String clientId`: l'identificativo con cui si registrerà il client
- `String topic`: il topic su cui si iscriverà il client
- `Boolean auto-reconnect`: indica se si intende o no che il client tenti la riconnessione in caso avvenga una disconnessione dal broker (tuttavia, sembra che questo meccanismo non funzioni o comunque non è stato possibile renderlo funzionante e si è preferito implementare meccanismi di recovery della connessione in altri modi)
- `Integer port`: la porta usata dal broker (questa va indicata manualmente in quanto il Eureka fornisce l'indirizzo della porta di ascolto di tomcat, non quella del broker)
- `Boolean userCredentials`: indica se si intende usare delle credenziali per connettersi al broker
- `String username`
- `String password`
- `Integer KeepAliveInterval`: secondi segnalati al broker per il keep alive

Infine la classe contiene un oggetto di tipo MqttClient che costituisce il vero e proprio client MQTT.
Dopo che il framework ha inizializzato il bean con i valori reperiti dal file di configurazione viene chiamato il metodo `afterPropertiesSet()` che al suo interno indica i passi successivi necessari all'inizializzazione del client, che consistono nella chiamata dei metodi `getBrokerInstance()`, `config()`, `connect()` e il metodo `subscribe()` della classe MqttClient.
`getBrokerInstance()` utilizza il discovery client per ottenere l'indirizzo del broker e, in caso il discovery client non fosse in grado di reperire l'indirizzo dal service registry, il thread

attende 2,5 secondi e ritenta finché non ottiene un indirizzo valido. In questo caso, infatti, è necessario bloccare l'esecuzione del servizio finché non viene reperito l'indirizzo in quanto la mancanza di questo rende impossibile il completamento dei passi successivi.
Successivamente viene chiamata la funzione `config()` che ha il compito di istanziare l'oggetto MqttClient, passando al costruttore l'url del broker, l'id del client e un oggetto MemoryPersistence, che viene usato per salvare eventuali messaggi ricevuti o inviati con qos 1 o 2 e l'oggetto ConnectionOptions che racchiude al suo interno alcuni parametri come l'username, la password e il keepAliveInterval. All'interno di questa funzione viene anche settato la classe che implmenterà l'interfaccia MqttCallback, che sarà l'istanza della classe stessa passata come parametro della funzione `setCallback()` dell'oggetto MqttClient.
All'interno della funzione `connect()` viene chiamato il metodo `connect()` dell'oggetto MqttClient, che accetta come parametro l'oggetto ConnectionOptions definito precedentemente. Questa funzione lancia un'eccezione MqttExeption nel caso la chiamata alla funzione `connect()` dovesse fallire, ad esempio nel caso l'indirizzo fosse sbagliato o il broker non fosse funzionante. Questo può avvenire, ad esempio, quando viene effettuata l'inizializzazione dei servizi e il broker non è ancora stato istanziato quando invece il client è già pronto.
Nel caso venga lanciata l'eccezione il thread entra in attesa per 30 secondi e ritenta la connessione finché questa non avviene con successo.

5.9.2 Ricezione e manipolazione dei messaggi

Una volta eseguita correttamente la connessione con il broker, viene chiamata la funzione `subscribe()` del MqttClient che accetta come parametro il nome del topic a cui iscriversi e il livello di qualità del servizio richiesto, in questo caso 0 visto che è tollerabile la perdita di qualche pacchetto.
L'interfaccia MqttCallback mette a disposizione vari metodi che permettono la gestione delle disconnessioni e la ricezione di messaggi pubblicati sui topic sottoscritti rispettivamente tramite i metodi `connectionLost()` e `messageArrived()`.
`ConnectionLost()` viene chiamato quando viene rilevata una disconnessione dal broker. In questo caso viene semplicemente richiamata la funzione `connect()` nella speranza che sia possibile ristabilire la connessione. In ogni caso la connessione verrà ritentata finché non avverrà con successo.
La funzione `messageArrived()` viene chiamata quando viene ricevuto un messaggio su uno dei topic a cui si è iscritto il client. La funzione accetta due parametri, una stringa che rappresenta il topic su cui è stato pubblicato il messaggio ricevuto, ed il messaggio stesso come oggetto MqttMessage. Dal momento che l'implementazione del metodo differisce per alcuni aspetti nei due servizi parser, verrà descritto successivamente.
Dal momento che può essere utile conoscere il produttore di un dispositivo per effettuare analisi sulla diffusione delle varie marche e riconoscere se l'OUI di un indirizzo mac è registrato o no, nel database è stata create una collection che contiene tutti gli OUI registrati dall'IEEE fino al mese di marzo del 2019. La lista è stata prelevata dal source code di wireshark, pubblicato su git hub all'indirizzo https://github.com/wireshark/wireshark. Tramite uno script realizzato in Python sono stati estratti dal file le diverse decine di migliaia di oui registrati e inseriti nel database in un'apposita tabella.
La classe Oui rappresenta questi dati e possiede i seguenti parametri:

- `String id`: l'id dell'elemento salvato nel database
- `String oui`: la stringa di sei caratteri esadecimali suddivisi a due a due dal carattere ':' che rappresenta la parte OUI del Mac Address
- `String shortName` e `String completeName`: rappresentano la versione contratta ed estesa dell'organizzazione a cui appartiene l'OUI

Questi dati sono reperibili tramite l'apposito repository che presenta anche un metodo `findByOui()` che accetta come parametro una stringa che rappresenta un OUI e restituisce un oggetto Optional contenente il corrispettivo oggetto Oui se presente nel database.
L'ultimo aspetto comune ai due servizi sono i metodi statici `bytesToHex()` e `HexToAscii()` della classe HelperFunctions.
Il primo metodo accetta come parametro un array di byte e restituisce una stringa rappresentante l'array in caratteri esadecimali e viene usato per convertire i dati binari provenienti dagli sniffer in una stringa comprensibile su cui è possibile effettuare analisi.
Il secondo metodo accetta come parametro una stringa in esadecimale e la converte nella corrispondente stringa in caratteri ASCII. Questo è necessario in quanto alcuni valori presenti all'interno del payload dei pacchetti sono in plain text, ad esempio il nome degli SSID annunciati. Queste stringhe vanno convertite in formato leggibile attraverso questa funzione.
Nel caso di subscriber parsed, questa classe presenta un metodo statico in più chiamato `parseParameters()` che ha il compito di estrarre dal payload del messaggio MQTT i vari campi che compongono i tagged parameters all'interno del pacchetto probe request. La funzione accetta come parametro la stringa che rappresenta il payload del messaggio MQTT in formato esadecimale e ritorna una lista contenente i dati come istanze di oggetti TaggedParameter.
A questo punto è necessario separare la descrizione dei due servizi in quanto iniziano a rendersi evidenti le differenze implementative di alcuni metodi ed i tipi di dati trattati.

5.9.3 Il servizio Subscriber Dump

Il servizio Subscriber Dump è nato principalmente per raccogliere i pacchetti probe request e salvarli nel database mantenendo il maggior numero di informazioni, concentrando l'attenzione sul contenuto dei pacchetti piuttosto che da che sniffer fossere stati raccolti in modo da effettuare analisi del contenuto dei singoli tag. Pertanto il pacchetto probe request deve essere salvato nella sua interezza e ogni tag deve essere accessibile così da poter essere analizzato. Lo scopo principale della funzione `messageReceived()` è estrarre dal messaggio MQTT ricevuto tutti i valori di interesse e manipolarli per poterli inserire in un oggetto della classe Packet che presenta i seguenti parametri:

- `String id`: l'id con inserito da mongo al momento dell'inserimento del db e permette di identificare univocamente il pacchetto una volta salvato
- `long timestamp`: indica il momento in cui è stato ricevuto il pacchetto da parte del subscriber secondo il timestamp unix
- `String snifferMac`: MAC address dello sniffer che ha catturato il pacchetto probe request

- `String deviceMac`: l'indirizzo mac del device che ha inviato il pacchetto probe request
- `String oui` e `String completeOui`: rappresentano il nome breve e completo associati al oui del deviceMac
- `String ssid` e `int ssidLen`: rappresentano rispettivamente l'eventuale ssid e la sua lunghezza
- `boolean global`: indica se il deviceMac è un indirizzo globally administered o no
- `String rawData`: contiene il payload del pacchetto probe request come stringa di caratteri esadecimali
- `int sequenceNumber`: il sequence number prelevato dal pacchetto probe request
- `List<TaggedParameter> taggedParameters`: lista contenente tutti i tagged parameters del pacchetto
- I campi year, month,weekOfYear, dayOfMonth, DayOfWeek, hour, quarter, tenMinute, fiveMinute e minute sono tutti interi e vengono usati per effettuare analisi aggregando i dati secondi i rispettivi frame temporali
- I campi fingerprint rappresentano vari modi di calcolare il fingerprint del device, nel tentativo di trovare l'insieme di attributi che garantiva il minor errore di conteggio

Il campo taggedParameters contiene una lista di oggetti Tagged Parameters che rappresentano gli elementi aggiuntivi all'interno del frame probe request e che vengono usati per la costruzione del fingerprint del device.
Per ogni tag sono presenti i campi `String tag`, `String value` e `int length` che rappresentano rispettivamente il codice del tag, il valore corrispondente e la lunghezza in caratteri ascii del contenuto del tag.
Nel caso dei tag DD, sono presenti anche `String oui` e `String completeOui` che rappresentano il nome compatto e completo del produttore associato a quel particolare tag.
Alla fine dell'elaborazione l'oggetto packet verrà salvato chiamando la funzione `save()` dell'apposito repository.

5.9.4 Il servizio Subscriber Parsed

Il servizio Subscriber Parsed è stato concepito per gestire la raccolta dei pacchetti già processati dai singoli sniffer, che sono quelli di effettivo interesse, che verranno usati per effettuare le stime sul numero e sulla distribuzione dei device.
La principale differenza rispetto a subscriber dump è nell'implementazione del metodo `messageReceived()` della classe MqttClient che riflette i diversi tipi di oggetti che vengono trattati da questo servizio. Infatti la classe Packet vista precedentemente presenta degli attributi diversi:

- `String snifferName`: nome comune con cui viene identificato uno sniffer dentro una room

- `String snifferBuildingId`, `String snifferBuilding`, `String snifferRoomId`,-`String snifferRoom`: permettono di identificare il nome e l'id della stanza e del building in cui è stato raccolto il pacchetto
- `int rssi`: indica la potenza del segnale ricevuto
- `String fcs`: frame check sequence del pacchetto utilizzato per individuare stessi pacchetti catturati allo stesso tempo da sniffer diversi

Alla classe Packet si aggiunge la classe LocalPacket che eredita dalla precedente ed aggiunge il campo `String fingerprint`. Ciò è stato fatto per evitare di avere `null` come possibile valore del fingerprint all'interno del database.
Al momento gli sniffer non hanno conoscenza della loro posizione quindi non possono includere all'interno dei messaggi MQTT dei valori che permettano di identificarla quindi per risolvere questo problema è stato necessario creare un apposito servizio che sia in grado di recuperare le informazioni sulla posizione degli sniffer effettuando una richiesta tramite il discovery client allo Sniffer Service per ottenere la lista della posizione di tutti gli sniffer. Queste richieste sono gestite dalla classe SnifferLocationService, che viene annotata con `@Service` ed iniettata all'interno della classe MqttClient. Le informazioni sulla posizione degli sniffer sono contenute in una mappa che associa l'id dello sniffer con un oggetto SnifferLocation che contiene i campi necessari alla localizzazione dello sniffer descritti precedentemente. Per reperire la posizione di uno sniffer è possibile chiamare la funzione `getSnifferLocation()` che accetta come parametro l'id dello sniffer e ritorna il corrispondente oggetto SnifferLocation. Il metodo `update()` effettua una richiesta allo Sniffer Service per ottenere la lista delle posizioni degli sniffer e le inserisce nella mappa come oggetti SnifferLocation.
La funzione `update()` viene chiamata all'interno di `afterPropertiesSet()` all'interno di MqttClient prima che il client venga istanziato e inizi la raccolta dei pacchetti. E' stato reso disponibile anche un end point REST che, effettuando una richiesta GET all'indirizzo /update, effettua il refresh delle informazioni.

5.10 Sniffers Service

Sniffer service ha il compito di implementare la gestione degli sniffer e permette l'inserimento di nuovi sniffer,la modifica e la lettura dei dati relativi a quelli salvati e la gestione della configurazione dei singoli sniffer. Il servizio, che è stato realizzato rispettando i principi del pattern MVC descritto in precedenza mette a disposizione vari endpoint REST che permettono di interagire con le informazioni salvate.
Il model è rappresentato dalle classi Sniffer, Room e Building: la prima permette di rappresentare gli sniffer fisici e contiene i dati che permettono di identificarli e localizzarli all'interno dello spazio considerato.
Dato che il sistema è stato concepito per funzionare principalmente all'interno di edifici e quindi senza considerare le effettive coordinate GPS dello sniffer, le classi Room e Building permettono di rappresentare la posizione degli sniffer sfruttando un'organizzazione gerarchica: ogni Building contiene al suo interno delle Room e queste a loro volta contengono gli Sniffer.
Ovviamente Building diversi possono contenere room con lo stesso nome ed e presente anche

un vincolo di unicità sul nome con cui uno Sniffer è identificato all'interno di una Room. Per contestualizzare questa suddivisione possiamo utilizzare il Politecnico come esempio: come Building si potrebbe avere Sede Centrale, Valentino, Edificio Aule P, Edificio Aule I e come Rooms le singole aule, i corridoi e gli uffici. Sta all'amministratore che esegue l'impostazione iniziale di questi dati andarli a definire in modo coerente e rappresentativo.
Quando si deve realizzare questo genere di gerarchie usando MongoDB come sistema di database, sono possibili due tipi di implementazione: tramite nested documents o tramite riferimenti.
Nel primo caso il documento di gerarchia inferiore è inserito all'interno di quello di gerarchia maggiore. Questo approccio da un lato ha la comodità che con una sola richiesta vengono ottenuti tutti i dati relativi all'oggetto di gerarchia maggiore e di gerarchia minore ma nel caso ci fossero molti documenti annidati potrebbero venire ritornati documenti non necessari e questo rappresenta uno spreco sia in termini di banda, in quanto vengono trasferiti dei dati che poi verranno scartati, che di tempo in quanto documenti più grandi impiegano più tempo per essere trasmessi.
Nel secondo caso invece, all'interno del documento di gerarchia maggiore viene inserito un riferimento al documento di gerarchia inferiore. Dato che ogni documento salvato all'interno delle collection è provvisto di un attributo id univoco, questo può essere utilizzato come riferimento. Se da un lato questo approccio permette di risolvere i problemi evidenziati precedentemente, dall'altro obbliga ad effettuare diverse richieste al database per ottenere tutti i dati desiderati. Nel nostro caso, avendo a che fare con una gerarchia a tre livelli, ho ritenuto che utilizzare la prima soluzione non fosse la scelta migliore in quanto, in contesti di deploy realistico, dove si potrebbero decine di stanze per edificio e alcuni sniffer per stanza, avremmo ottenuto una rappresentazione troppo confusionaria. Utilizzando invece i riferimenti, avremo che all'interno di ogni building sarà presenta una lista di riferimenti alle rooms contenute ed in ogni room una lista di riferimenti agli sniffer.
La classe Sniffer ha i seguenti attributi:

- `String id`
- `String mac`: l'indirizzo mac dello sniffer utilizzato
- `String macId`: l'indirizzo mac senza il carattere ':' a dividere i byte, per renderlo più adatto all'utilizzo all'interno di url.
- `String name`: il nome comune utilizzato per individuare lo sniffer
- `String roomId`, `String roomName`, `String buildingId`, `String buildingName`: permettono di localizzare la posizione dello sniffer
- `String status`
- `Configuration configuration`

Questi ultimi due attributi necessitano di un approfondimento.
L'oggetto sniffer rappresentato da questo classe va considerato separato dall'effettivo device utilizzato come sensore dal momento che questo può essere sostituito in caso di malfunzionamenti o disattivato. Possiamo individuare una suddivisione in sniffer "fisico" e sniffer "logico", quest'ultimo rappresentato dall'oggetto della classe sniffer, mentre il primo è

rappresentato da uno user con authority sniffer che rappresenta l'effettivo device utilizzato. Per quanto lo sniffer logico una volta definito rimane persistente in quanto necessario come riferimento per effettuare ricerche sul database, lo sniffer fisico associato può cambiare pertanto è stato implementato un meccanismo che permette di "distaccare" lo sniffer logico da quello fisico così da permetterne la sostituzione.
Il campo status indica questa evenienza se il suo valore è impostato a "detached". In questo caso l'oggetto sniffer ha le variabili `mac` e `macId` impostate a null e saranno nuovamente definite quando verrà impostato un nuovo sniffer fisico.
La classe Configuration rappresenta i parametri di configurazione degli sniffer e viene richiesta da questi al momento dell'inizializzazione.
Gli attributi della classe sono:

- `boolean dumpMode`: indica la modalità di funzionamento dello sniffer, se true i pacchetti vengono inviati secondo la modalità dump
- `boolean privacyMode`: indica se lo sniffer deve funzionare in modalità privacy, attualmente questa funzionalità non è stata implementata negli sniffer.
- `String brokerAddress`: l'indirizzo del broker MQTT
- `String topic`: il topic su cui verranno pubblicati i messaggi contenenti i dati dei pacchetti intercettati
- `int powerThreshold`: indica la potenza minima che un pacchetto deve avere per essere catturato. Il range di valori accettati varia da -100 (potenza minima) a 0 (potenza massima). Di default questo valore è -100 e significa che tutti i pacchetti sono catturati. Scegliendo valori maggiori verranno esclusi i pacchetti di potenza inferiore e quindi, idealmente generati da device più lontani. Questo meccanismo può essere utile per esempio, per catturare i pacchetti generati da device all'interno di una stanza.

Le classi Room e Building hanno lo stesso schema:

- `String id`
- `String name`
- `List<String> Rooms` nel caso dei buildings e `List<String> Sniffers` nel caso di Rooms

Per quanto riguarda la parte controller, l'implementazione è stata fatta nel modo descritto precedentemente, cioè classi controller che ricevono come dipendenza un servizio che implementa i metodi chiamati dalle funzioni che rispondono alle richieste http.
Sono presenti endpoint che permettono di ottenere tramite GET le liste di sniffers, rooms e buildings registrati. In particolare sono presenti endpoint che ritornano le rooms avendo specificato un building e gli sniffer specificando un building o un building ed una room.
Tra i vari metodi che permettono di gestire le rooms ed i buildings, manca la possibilità di modificare dopo che una di queste entità è stata creata. Questa scelta implementativa è dovuta al fatto che, nel caso venisse modificato l'attributo, sarebbe necessario andare a modificare potenzialmente milioni di record all'interno del database. E' evidente che questa

soluzione sia solo un palliativo ed andrebbe trovata una soluzione più efficace e valutato l'impatto di una modifica del genere. Attualmente quindi chi inserisce questi dati deve verificarne la correttezza prima di confermare (anche se va considerato che l'amministratore che inserisce questi dati ha la capacità di accedere al database e modificarli direttamente da li in caso di necessità).
Per quanto riguarda gli sniffer è possibile invece modificare l'indirizzo mac dello sniffer fisico associato e la configurazione dello sniffer per permettere di sostituire lo sniffer fisico in caso di malfunzionamenti. In generale questi dati vanno considerati non più modificabili una volta creati, anche perché vengono usati per effettuare ricerche sui pacchetti raccolti ed un'eventuale modifica renderebbe i pacchetti catturati in precedenza non più ottenibili.
Va sottolineato che la procedura di creazione di uno sniffer può avvenire in due modi, indicando o non indicando un indirizzo mac associato allo sniffer.Nel primo caso, se non è ancora registrato uno sniffer fisico con l'indirizzo MAC indicato, verrà chiamato il servizio apposito e creata la risorsa. Nel secondo caso si dovrà successivamente associare lo sniffer logico con uno fisico non ancora utilizzato da altri sniffer logici.

5.11 Users Service

Users Service è il servizio che implementa le funzionalità di gestione degli utenti come la registrazione, la modifica e la cancellazione degli utenti registrati.

5.11.1 Endpoint esposti

Soltant gli utenti con authority Admin possono accedere alle funzionalità di cancellazione, modifica e creazione di nuovi utenti mentre gli utenti con authority User possono solamente accedere alle proprie informazioni e modificare la password. Gli User hanno l'accesso agli endpoint `/users{id}` su cui possono effettuare GET per ottenere i dati relativi all'utente e PUT per effettuare l'update della password. Il campo id rappresente l'identificativo dell'utente su cui si vuole operare che viene ottenuto come path variable. Per evitare che un utente operi delle modifiche o visualizzi dati altrui, viene effettuata la verifica che l'utente che effettua la richiesta corrisponda a quello di cui sono stati richiesti i dati.
Gli admin accedono agli endpoint che iniziano con la dicitura `/restricted`. Tramite `/restricted/users/{id}` l'Admin può leggere i dati relativi all'utente tramite GET, modificare i dati salvati tramite PUT e calcellare l'utente tramite DELETE. Ovviamente, in tutti i casi di POST o PUT, la richiesta deve contenere un oggetto che contiene i dati dell'utente che si va a creare o modificare.
In fine, tramite gli endpoint `/restricted/users`, `/restricted/admins` e `restricted/-sniffers` e possibile, effettuando una POST, creare un nuovo utente con il privilegio indicato dall'url. Durante la procedura di aggiunta di un nuovo sniffer, viene invocato l'endpoint `restricted/sniffers` tramite GET per verificare che lo sniffer creato non corrisponda ad uno già esistente ed in caso contrario viene creato il nuovo utente che rappresenta lo sniffer. Nel caso di utenti con authority Sniffer, non è presenta la mail e l'username corrisponde al MacId descritto nel capitolo precedente, cioè l'indirizzo MAC dello sniffer senza i caratteri ":" e la password è ottenuta sempre secondo il metodo descritto precedentemente.

Effettuando una GET su `/restricted/users/` è possibile ottenere la lista di tutti gli utenti registrati, compresi Admin e Sniffer.

5.12 Data Enricher Service

Inizialmente questo servizio era nato con l'intento di aggiungere informazioni ai dati salvati sul database, ma col tempo si è preferito inserire tutti i dati direttamente al momento del salvataggio nel database dei pacchetti ricevuti. Al momento questo servizio implementa i meccanismi di conteggio dei pacchetti per permettere di effettuare stime sul numero dei device. Queste stime si basano sui concetti descritti precedentemente e permettono di individuare una stima dei device che hanno emesso pacchetti probe request all'interno di blocchi temporali di 5 minuti.
Dal momento che i conteggi vanno effettuati con una cadenza regolare, è stata usata una funzionalità di Spring che permette di programmare l'esecuzione di funzioni. Per abilitare questa funzionalità è necessario annotare la classe di bootstrap con `@EnableScheduling`.
Per automatizzare l'esecuzione di un metodo, che deve essere inserito in una classe annotata con `@component` è necessario annotarlo con `@Scheduled`.
L'annotazione accetta alcuni parametri:

- `fixedDelay` permette di indicare o millisecondi che devono trascorrere prima che il metodo venga nuovamente invocato
- `fixedRate` indica il tempo che deve trascorrere tra un invocazione e l'altra partendo dal momento della prima invocazione, quindi non attendendo il termine della precedente

Questi due parametri esistono anche nella versione `fixedDelayString` e `fixedRateString` che permettono di usare una stringa rappresentante l'intervallo invece che un intero.

5.12.1 Conteggio dei dispositivi

La classe che implementa il conteggio è `CountProbeRequestTask`. Questa classe viene annotata con `@Component` e al suo interno vengono iniettate le varie dipendenze di cui necessita, tra cui MongoTemplate, usato per accedere alle funzioni di Mongo Aggregation ed i repository che implementano i sistemi di accesso al database dei vari tipi di dato gestito. Il metodo `count()` è quello che implementa effettivamente il conteggio dei device e viene eseguito usando ogni 10 minuti a partire da quando è terminata l'esecuzione precedente, usando il parametro `fixedDelayString` indicato precedentemente. La durata dell'intervallo di attesa è definita nel file di configurazione del servizio, che di per sé è un intero, viene estratto con la seguente espressione che diventa il parametro dell'annotazione `$tasks.count.delay`.
Si è deciso di effettuare i conteggi dei pacchetti considerando finestre temporali di 5 minuti dato che questo valore permette di ottenere un errore di stima ragionevole, pur restando un intervallo di tempo in cui è probabile che un device emetta dei pacchetti.Questi intervalli di 5 minuti sono assoluti e non relativi ad un momento particolare quindi per ogni ora avremo 12 finestre o frame che conterranno tutti i dati raccolti, nel caso della prima dal minuto 0 a 4 e 59 secondi, nel secondo da 5 a 9 e 59 e così via. Il risultato dei conteggi ci permetterà di ottenere per ogni frame temporale di 5 minuti il numero totale di pacchetti

catturati, quanti hanno un mac globally administered e quanti no, ed il numero di device stimati in base agli indirizzi globali univoci e fingerprint univoci.
Per semplicità, da questo momento con la nomenclatura *pacchetti global* si intenderanno i pacchetti generati da un device che si annuncia con un indirizzo mac globally administered mentre *pacchetti local* nel caso di indirizzi mac locally administered.
La funzione `count()` è di per sé molto complessa e può essere suddivisa in diversi step:

1. Inizialmente viene reperito tramite l'apposito repository, iniettato nella classe, un timestamp che rappresenta l'istante corrispondente all'inizio dell'ultimo frame temporale analizzato. Conoscendo questo dato possiamo generare il timestamp corrispondente al blocco di 5 precedente a quello attuale al momento dell'avvio del task, ad esempio se l'ultimo conteggio comprende i dati raccolti fino a . Nel caso il timestamp di partenza non fosse reperibile all'interno del database, significa che siamo al primo conteggio, in questo caso viene preso il primo pacchetto raccolto in assoluto e viene estratto il timestamp da cui viene costruito quello corrispondente all'inizio del time frame di cui quel timestamp fa parte e il processo prosegue normalmente.

2. Vengono richiesti allo sniffer service i dati relativi a tutti gli sniffer e vengono salvati in una mappa che associa l'id dello sniffer all'oggetto sniffer stesso. Questo è necessario in quanto al termine dell'aggregazione l'unico dato che permette di identificare lo sniffer che ha raccolto i pacchetti considerati è l'id. In questo modo è possibile ottenere i dati completi che permettono di individuare la posizione ed il nome dello sniffer. Questo dati vengono usati anche per effettuare aggregazioni basate su luoghi, ad esempio se si volesse ottenere la stima media di presenza in una stanza o edificio.

3. Vengono definiti i vari step della pipeline di elaborazione di Mongo Aggregation. Vengono definite due pipeline diverse, una per il conteggio dei pacchetti global e la seconda per i local. Gli step dell'aggregazione sono:

   - Match che permette di restringere le successive operazioni di aggregazione ai soli pacchetti catturati all'interno del frame temporale di interesse e che siano rispettivamente global o local
   - Aggregation dei pacchetti intervalli di 5 minuti e per snifferId. Vengono contati tramite l'operatore count() i singoli pacchetti raccolti e con addToSet() viene create una lista dei mac univoci nel caso dei pacchetti global e dei fingerprint nel caso di quelli local.
   - Projection per conteggiare gli elementi presenti nel set popolato nello step precedente
   - Order dei risultati in ordine temporale ascendente

4. Tramite mongoTemplate viene effettuata la richiesta al database. L'operazione ritorna una lista di oggetti di tipo CountResult (una lista per i pacchetti global ed una per quelli local) che hanno i seguenti attributi:

   - `CountResultId timeframe`: i campi secondo cui viene effettuata l'aggregazione vengono inclusi in un unico campo _id, durante la fase di projection questo campo viene rinominato in timeFrame e mappato sull'oggetto CountResultId. Questo'oggetto contiene i dati che permettono di identificre il frame temporale ed l'id dello sniffer che ha raccolto i pacchetti:

  - `int year`
  - `int month`
  - `int dayOfMonth`
  - `int hour`
  - `int fiveMinute`: l'intervallo di 5 minuti in cui è stato raccolto il pacchetto, ad esempio i pacchetti raccolti tra il tempo 0 minuti e 0 secondi di un ora e 4 minuti e 59 secondi rientreranno nel blocco fiveMinute 1
  - `String snifferId`
  - `int totPackets`: il numero di pacchetti catturati dallo sniffer all'interno del frame temporale
  - `int totMacs`: il numero di mac distinti conteggiati o il numero di fingerprint distinti. Il nome usato è lo stesso per semplicità.

Gli oggetti che verranno salvati sul database devono includere i due conteggi per ogni time frame quindi è necessario effettuare il merge delle due liste.
L'oggetto inserito nel database ha quindi il seguente schema ed è definito dalla classe `CountedPackets`:

- `String id`: l'id utilizzato da mongo come identificativo nel database
- `String snifferMac, String snifferId, String snifferName, String buildingName, String buildingId, String roomName, String roomId`: permettono di identificare lo sniffer che ha raccolto quei pacchetti e dove è localizzato
- `int totalPackets`:il numero totale di pacchetti catturati dallo sniffer nel frame di 5 minuti
- `int globalPackets` e `int localPackets`: sono rispettivamente il numero di pacchetti globali e pacchetti locali catturati
- `int totalDistinctMacAdresses` e `int totalDistinctFingerprints`: sono il numero di mac e fingerprint distinti catturati in quel time frame
- `long startTimestamp`: il timestamp unix corrispondente all'inizio del timeframe
- `int year, int month, int weekOfYear, int dayOfMonth, int dayOfWeek, int hor, int twoHour, int fourHour, int sixHour, int twelveHour, int thirtyMinute, int fifteenMinute, intfiveMinute e int minute`: per effettuare aggragazioni in base temporale

Questo ultimo insieme di valori permette di identificare con precisione il momento della raccolta e permette di effettuare aggregazioni più complesse per individuare trend con una granularità variabile. Ad esempio se si volesse individuare quale giorno della settimana presenta il numero massimo di conteggi si può effettuare l'aggregazione usando il termine dayOfWeek. Tutti questi valori partono da 1, ad esempio fifteenMinute, che indica il frame di 15 minuti in cui sono stati catturati i pacchetti conteggiati, varrà 1 per tutti i time frame che avranno valore fiveMinute compreso tra 1 e 3, 2 da 4 a 6, 3 da 7 a 9 e 4 da 10 a 12. A partire dagli elementi delle due liste ritornate, vengono generati gli oggetti CountedPackets e vengono fusi i conteggi appartenenti allo stesso frame sfruttando una mappa che utilizza come

chiave un intero generato dalla funzione `hashCode()` della classe `CountedPackets`, che utilizza i campi che identificano il time frame e lo sniffer per generare l'hash e come valore l'oggetto CountedPackets stesso. Inizialmente viene scandita la lista contenente i conteggi dei pacchetti global: per ogni elemento viene inserito nella mappa l'oggetto CountedPackets corrispondente i cui valori sono stati estratti o calcolati a partire dal contenuto dell'elemento in considerazione. Scanditi tutti gli elementi si passa a quelli local: similmente, per ogni elemento, viene creato un oggetto `CountedPackets` e viene testata la presenza di un elemento della mappa avente la stessa chiave, in caso positivo significa che è già stato inserito un elemento e quindi questo viene completato con i dati mancanti, in caso contrario viene inserito nella mappa.
Questo secondo caso è molto raro in quanto la maggior parte dei pacchetti raccolti è global e questo caso si avrebbe solo se in un frame temporale non fossero stati raccolti pacchetti global, eventualità per ora mai osservata ma possibile.
Alla fine del processo viene salvato nel database il timestamp corrispondente all'inizio dell'ultimo timeframe considerato e gli elementi presenti all'interno della mappa vengono salvati in blocco nel database.

5.13 Data Explorer Service

Data Explorer Service è il componente del back end che espone gli endpoint che vengono richiamati per ottenere i dati relativi alle stime di posizione e conteggio dei device. Questo servizio include anche un client MQTT tramite il quale è possibile inviare i messaggi di reset agli sniffer. Il servizio presenta quattro classi controller che forniscono gli endpoint rispettivamente per il conteggio dei pacchetti, il tracking di un device, il tracking di device multipli per visualizzare le heatmap, informazioni generali sul numero di pacchetti raccolti e conteggi effettuati ed una quinta classe controller che espone i metodi necessari a comunicare con il client MQTT. Come per tutti gli esempi precedenti, questo servizio è stato sviluppato sempre nello stesso modo: classi controller che espongono metodi che chiamano metodi di servizi che interagiscono con il database attraverso repository. I dati ritornati sono sempre file JSON che rappresentano i dati richiesti.

5.13.1 La classe CountedPacketsController

La classe `CountedPacketsController` espone i metodi che permettono di interrogare la tabella `countedPackets` del database che contiene i risultati dei conteggi effettuati da Data Enricher Service.
La classe espone tre metodi che permettono di effettuare le ricerche filtrando i risultati rispettivamente per building, room o sniffer. Gli altri parametri che devono essere forniti sono due timestamp che limitano l'intervallo di risultati ritornati e la risoluzione con cui si vuole ottenere i risultati: infatti i dati nel database sono aggregati per intervalli di 5 minuti, indicando una risoluzione temporale più bassa, questi verranno aggregati su intervalli di tempo maggiori e verrà fatta la media dei valori.
I risultati ritornati comprendono, per ogni intervallo di tempo relativo alla risoluzione desiderata, tutti i conteggi relativi ad un singolo sniffer, agli sniffer in una room o direttamente a livello di building per l'intervallo di tempo desiderato. I conteggi comprendono

oltre ai riferimenti temporali, il numero totale di pacchetti raccolti nel tempo considerato, quanti di questi siano local o global, il numero di MAC address distinti, il numero di fingerprint distinte ed il numero finale di device stimati risultante dalla somma dei due parametri precedenti.
Anche nel caso vengano fatte aggregazioni per room o building, viene fatta la media dei singoli valori.
É stato implementato anche un meccanismo che permette di riempire eventuali buchi all'interno dei dati, che potrebbero derivare da momentanei malfunzionamenti della rete o degli sniffer che impediscono di raccogliere dati per un intervallo di tempo superiore ai 5 minuti. In questi casi viene inserito un record contanente il conteggio di 0 device in modo da garantire una corretta visualizzazione successivamente. Questi dati vengono generati solo su richiesta e non salvati nel database in quanto occuperebbero spazio inutilmente.

5.13.2 La classe DeviceTrackingController

La classe `DeviceTrackingController` espone i metodi necessari a reperire dal databse i dati necessari alla visualizzazione del tracciato che mette in relazione la posizione di rilevamento di un device ed il tempo. Questi dati vengono anche usati per creare heathmap che rappresentano le zone con una maggiore probabilità di trovare un dispositivo. I calcoli necessari ad effettuare queste stime vengono eseguite dal front end e sono descritte nel capitolo 7. Gli endpoint esposti permettono rispettivamente di ottenere la lista di tutti i dispositivi rilevati in un certo lasso di tempo. Per ogni dispositivo sono riporatati il MAC address e le informazioni sul vendor in modo tale da permettere di filtrare le informazioni a livello del client. Un altro endpoint permette di richiedere, dato un MAC address selezionato tra quelli ottenuti precedentemente, la lista di tutti i rilevamenti del device stesso, ordinati per tempo di osservazione. I dati ritornati contengono anche le informazioni necessarie all'identificazione dello sniffer che ha rilevato il pacchetto ed il valore di RSSI associato al pacchetto. Questi valori sono utilizzati a livello di front end per permettere le visualizzazioni descritte nel prossimo capitolo.

5.13.3 La classe GeneralDataController

Tramite gli endpoint esposti da questa classe, possiamo ottenere dati relativi al numero di pacchetti raccolti in generale o per singolo sniffer a la relativa percentuale di pacchetti global e local, l'ultima stima rilevata per sniffer e la media relativa all'ora, giorno della settimana e mese in modo tale da poter fornire un valore di riferimento per quanto riguarda il numero di device stimati. Questi dati vengono utilizzati dal front end per realzzare alcuni grafici.

5.13.4 La classe FlowController

Questa classe espone l'endpoint tramite il quale vengono ottenuti i dati necessari alla realizzazione di heatmap che dovrebbero rappresentare le aree con un afflusso maggiore all'interno di un edificio. É necessario fornire come parametro un intervallo di tempo per limitare l'insieme dei dati trattati. Successivamente, per ogni device rilevato nell'intervallo di tempo, verranno aggregati per ogni sniffer e per intervalli di un minuto tutti i pacchetti

raccolti e verrà effettuata la media degli RSSI in questo modo per ogni device, per ogni sniffer e per ogni minuto si potrà calcolare la distanza di ogni device dallo sniffer.

5.13.5 La classe MQTTController

La classe MQTTController espone gli end point che permettono di ordinare l'invio da parte del client MQTT dei messaggi di reset ad un singolo sniffer o a tutti quelli registrati con il broker e raggiungibili. Il client MQTT è stato implementato nello stesso modo dei servizi Subscriber. Invocando l'endpoint `/reset` viene pubblicato sul topic commands il comando reset che riavvierà tutti gli sniffer mentre aggiungendo alla URL il MACid di uno sniffer, il messaggio verrà pubblicato su `/sniffer/MACid` e si riavvierà solo quello indicato.

5.14 Deploy

5.14.1 Definizione dei Dockerfile

Come detto in precedenza, i vari servizi sono eseguiti all'interno di container Docker. Per ogni servizio è definito un dockerfile che contiene le indicazioni necessarie alla creazione dell'immagine del servizio. I dockerfile sono uguali per ogni servizio, differisce soltanto il nome del file che viene scritto nell'immagine in quantto riflette il nome del file jar che verrà deployato. Di seguito viene riportato ,a titolo di esempio, il contenuto di uno dei dockerfile:

```
FROM openjdk:8-jdk-alpine
ADD target/dataexplorer-0.0.1-SNAPSHOT.jar data-explorer.jar
ENTRYPOINT ["java", "-jar", "/data-explorer.jar"]
```

Il comando `FROM` indica l'immagine di partenza su cui verrà basata quella in creazione. In questo caso l'immagine contenente OpenJDK in versione 8 basata sulla distribuzione Linux Alpine che è caratterizzata da una dimensione inferiore rispetto ad altre immagini basate su distribuzioni Linux più note come Ubuntu.
Il comando `ADD` permette di effettuare la copia del file jar all'interno dell'immagine, a livello della root del file system del container.
Infine il comando `ENTRYPOINT` indica il comando che verrà eseguito una volta avviato il container, in questo caso `java -jar /data-explorer.jar` che avvierà l'applicativo.

5.14.2 Utilizzo di Docker Compose

Ovviamente gestire questi container singolarmente non è la soluzione migliore, in quanto i diversi container hanno bisogno di configurazioni leggermente differenti e impartire i comandi manualmente espone ad errori e dimenticanze.
Tramite Docker Compose è possibile definire un file yml che contiene le definizioni dei parametri dei singoli servizi ed avvarli tutti insieme attraverso il comando `up`. Nel giro di pochi minuti i servizi saranno attivi ed inizierà la raccolta dei dati.
L'unico aspetto da considerare è modificare il file docker-compose.yml conseguentemente ad eventuali differenze nella struttura di cartelle in cui sono inseriti i file .jar dei vari servizi. Infatti tra i parametri definibili all'interno del file di configurazione di Docker Compose è possibile indicare anche la posizione del Dockerfile del servizio e aggiornare l'immagine in

caso siano state apportate delle modifiche ai file sorgente.
Nel seguente esempio è riportata la porzione del file di configurazione relativo ad uno dei servizi:

```
security-service:
    container_name: security
    build:
      context: ./security
      dockerfile: Dockerfile
    image: security:latest
    expose:
      - 8764
    networks:
      - my-net
    environment:
      SPRING_CLOUD_CONFIG_URI: http://configuration-service:8888
    depends_on:
      - configuration-service
      - eureka-service
      - users-service
      - mongodb
    logging:
      driver: "json-file"
      options:
        max-size: "500k"
        max-file: "10"
```

Tra i parametri è possibile indicare le porte esposte e quelle mappate verso l'esterno e quindi raggiungibili sulla rete locale, la rete virtuale creata da Docker su cui connettere il container, eventuali variabili d'ambiente, anche relative al contesto di Spring e la lista di servizi da cui dipende il servizio considerato e che quindi dovranno essere avviati precedentemente.
Tuttavia questo non garantisce che il servizio sia effettivamente attivo e bisogna attuare altri meccanismi per essere sicuri che ogni servizio sia in grado di avviarsi correttamente. Ad esempio se il server di configurazione non fosse ancora attivo al momento dell'avvio di uno dei servizi, quest'ultimo non sarebbe in grado di ottenere la configurazione e terminerebbe. Fortunatamente in questi casi è possibile utilizzare le funzionalità messe a disposizione da Spring Retry che permettono di definire nel file `bootstrap.properties` quante volte il server di configurazione deve essere contattato in modo fallimentare prima di terminare il programma.
Alcuni servizi però non dipendono solo da quello di configurazione ma, ad esempio da Eureka o dal servizio che ospita il broker MQTT. In questi casi è stato necessario attuare metodologie ad hoc in modo tale da prevenire l'arresto dei servizi.

Capitolo 6

Implemetazione del front end

6.1 Introduzione

Il front end permette di visualizzare i risultati delle analisi effettuate sui dati raccolti, effettuare operazioni sugli utenti e gestire gli sniffer e le loro configurazioni. Questo componente è stato sviluppato come una web app, accessibile come una semplice pagina web tramite un qualsiasi browser web. Per realizzarlo è stato utilizzato Angular, un potente framework MVC che permette la creazione di Single Page Application (SPA).
In seguito ad una introduzione in cui verranno esposti alcuni aspetti tecnici che caratterizzano questo framework e che sono stati utilizzati per la realizzazione del front end, varranno discusse le funzionalità dell'applicativo, suddivise secondo le pagine che lo compongono.

6.2 Il framework Angular e le Single Page Application

Angular [18] è un progetto open source svilppato da Google e rilasciato a partire dal 2016 e negli anni si è imposto come uno dei framework più utilizzati per lo sviluppo di Single Page Application. Questo tipo di applicativi web consistono in una pagina web in grado di renderizzare in modo dinamico i suoi elementi senza necessità di ricaricare la pagina, grazie a codice Javascript che si occupa di gestire la composizione della pagina e l'aggiornamento dei dati visualizzati, infatti, una volta effettuata la prima richiesta al web server che ospita la SPA richiesta, tutti i file che la implementano vengono scaricati sul client e non sono neccessarie altre richieste. L'applicazione è infatti in grado di gestire la navigazione tra diverse pagine, la richiesta di dati ad API esterne tramite chiamate XHR (Xml Http Request) per aggiornare i dati visualizzati. E' evidente come questo approccio si sposi perfettamente con quanto realizzato nel back end: una volta definite le viste che componegono la nostra applicazione e la logica di gestione dei dati, tramite richieste http è possibile interagire con i vari microservizi per ottenere i dati desiderati e permetterne la manipolazione o la visualizzazione.
Uno dei principali vantaggi di Angular rispetto ad altri framework di questo tipo è rappresentato dall'utilizzo del linguaggio Typescript al posto di Javascript. Typescript è un superset di Javascript e permette di introdurre costrutti tipici dei linguaggi ad oggetti come

classi ed interfacce e l'utilizzo dei tipi di dato, andando a rendere questo linguaggio più adatto alla costruzione di robuste e complesse web application.

6.2.1 I componenti

L'unità fondamentale di un applicativo Angular è rappresentato dai componenti. Questi elementi vanno a definire le unità funzionali e grafiche che, composte, permettono la creazione delle pagine visualizzate e l'implementazione delle funzioanlità.

```
@Component({
    selector:    'app-hero-list',
    templateUrl: './hero-list.component.html',
    styleUrls: ['./hero-list.component.css']
    providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
    /* . . . */
}
```

Nell'esempio precedente, sono presenti gli elementi che determinano un componente. Il decoratore `@component` permette di identificare la classe sottostante come una classe che descrive un componente e permette di specidficare alcuni metadati necessari alla corretta visualizzazione della view associata al componente:

- `templateUrl` indica il riferimento al template del componente, un file `html` che verrà usato come modello per il rendering della view del component
- `styleUlrs` indica una lista di fogli di stile `CSS` che verranno usati per personalizzare l'estetica del componente
- `selector` è un selettore `CSS` che permetterà di inserire il componente all'interno del template di altri componenti tramite, in questo caso, l'utilizzo del tag `<app-hero--list></app-hero-list>`
- `providers` indica una lista di servizi che il componente richiede per poter visualizzare i dati richiesti

Nella classe sottostante e possibile definire variabili e funzioni, che avranno principalmente il compito di manipolare i dati per renderli visualizzabili. Il componente può essere richiamato all'interno del template di altri componenti andando così a costituire un albero di componenti che determina le pagine dell'applicazione. Il componente chiamato prende il nome di componente figlio mentre il chiamante componente padre.

6.2.2 Il Data Binding

Una delle funzionalità più interessanti fornite dal framework è la possibilità di legare variabili del componente ad elementi della view permettendo di sincronizzare il valore di un dato con quello visualizzato. Questo meccanismo prendo il nome di data binding ed in Angular avviene in entambe le direzioni, dalla view al componente e viceversa,

prendendo il nome di two-way data binding. All'interno del template di un componente è possibile vincolare il valore di un particolare elemento con quello di una variabile della classe in maniera unidirezionale, dal componente verso il template attraverso l'utilizzo nel template della sintassi `{{variabile}}` dove all'interno delle parentesi è presente il nome della variabile definita all'intreno della classe. In questo modo quanto il valore della variabile verrà modificato, il framework si accorgerà della modifica e anche l'elemento del template verrà modificato. Questo metodo prende il nome di *interpolation*. Il framework mette a disposizione altre tre forme di comunicazione tra template e classe. Il *property binding* permette di settare valori di variabili di un altro componente (componente figlio), richiamato all'interno del template del componente considerato (componente padre), vincolandoli come prima al valore di una variabile del componente padre, che verrà iniettata nel componente figlio e verrà aggiornata dal framework in caso venisse modificata.
Questo meccanismo usa la seguente sintassi, inserita all'interno del template del componente padre:

```
<componente-figlio [variabileFiglio]="variabilePadre"></componente-figlio>
```

L'*event binding* pemette di legare metodi della classe del componente agli eventi lanciati dagli alementi del template in seguito, ad esempio, alla pressione di un pulsante o al transito del mouse su una parte della view. Questo meccanismo permette di emettere dati dal componente figlio al componente padre.
Anche in questo caso viene usata una sintassi peculiare all'interno de template del componente padre:

```
<componente-figlio (click)="funzioneDelPadre()"></componente-figlio>
```

In questo caso viene chiamata la funzione `funzioneDelPadre()` quando viene lanciato l'evento click da parte del componente figlio. Il property binding e l'event binding possono anche essere usati su elementi del template che non sono necessariamente altri componenti, ma semplicemente elementi della pagina come i vari tag html che possono essere utilizzati. Infine possiamo combinare questi due metodi andando ad ottenere il two way data binding che permette di mantenere sempre sincronizzato il valore di un dato visualizzato nella view anche quando questo viene direttamente modificato, ottenendo un data binding bidirezionale. Questo tipo di data binding usa la seguente sintassi:

```
<input [(ngModel)]="variabilePadre">
```

Nell'esempio, il two way data binding viene usato all'interno di un tag input di un form, in questo modo eventuali modifiche manuali del dato da parte dell'utente si rifletteranno immediatamente sul valore della variabile `variabilePadre` della classe e viceversa.

6.2.3 Direttive strutturali

Il framework mette a disposizione alcuni particolari classi, chiamate direttive che hanno il compito di rendere dinamica la composizione del template, permettendo di modificarlo in base al valore delle variabili del componente. Tra queste direttive quelle più utilizzate sono:

- `*ngFor`: permette di ripetere un particolare elemento del template per ogni elemento di una lista

- `*ngIf`: permette di inserire nel DOM un'elemento del template secondo il valore di una variabile booleana o di un espressione che ne ritorna una

6.2.4 I servizi e la dependency injection

E' buona norma separare la logica di business dall'accesso ai dati in modo da aumentare la modularità e la riusabilità del codice.
Così come con Spring i services non accedevano direttamente al database ma utilizzavano i repository messi a disposizione da Spring Data, anche Angular prevede questa separazione ma utilizza una nomenclatura diversa: in questo caso con servizi vengono individuate tutte quelle classi che implementano particolari funzinalità che non rientrano tra quelle proprie dei componenti (che dovrebbero ridursi alla sola manipolazione dei dati al fine di prepararli per la visualizzazione) tra cui la richiesta di dati ad un server tramite REST.
All'interno dei servizi sono stati implementati tutti i metodi che permettono di chiamare gli endpoint esposti dai vari microservizi disponibili.
Come Spring, anche Angular implementa la dipendency injection che viene usata per iniettare all'interno dei componenti i servizi di cui hanno bisgno. La dependency injection non viene usata solamente per i servizi ma permette di iniettare come dipendenze anche funzioni e variabili. Per rendere una classe iniettabile è necessario utilizzare il decoratore `@Injectable`. Quando il framework rileverà la presenza di una dipendenza all'interno di un componente cercherà di risolverla prima di chiamare il costruttore del componente. Quando un componente ha bisogno di recuperare dei dati dal back end, viene chiamata la relativa funzione del servizio che reperirà le informazioni e le passerà al componente.

6.2.5 HttpClient e programmazione reattiva

Angular mette a disposizione un potente client Http che sfrutta l'interfaccia XHR esposta dai browser. Uno degli aspetti di maggiore interesse di questo client è che il risultato della richiesta effettuata al server è inserito all'interno di un *Observable*, un'oggetto messo a disposizione della libreria RxJS, specializzata nel trattare flussi di dati asincroni con un approccio funzionale. Questo stile di programmazione prende il nome di *programmazione reattiva*.
Un observable è l'oggetto che media il passaggio di dati tra un produttore, in questo caso il client Http, e un consumatore, il componente. Quando un componente intende ricevere i dati emessi dall'observable chiama la funzione `subscribe()`. All'interno della funzione è possibile indicare la callback che verrà chiamata quando un nuovo dato verrà emesso e un'ulteriore funzione che verrà chimata in caso di errore. Nell'observable infatti, il producer può pubblicare i dati chiamando la funzione `next()` che scatena l'emissione di un nuovo dato e allo stesso modo chiama la funzione `error()` in caso di errori.
Nel caso del client Http, che viene iniettato dentro la classe ch eimplementa il servizio, quando viene fatta una richiesta Http, viene ritornato un observable che emetterà il dato ricevuto dal back end una volta che qualche componente si sarà iscritto e che il dato sarà disponibile.
All'interno del servizio avremo il seguente metodo che interroga l'endpoint REST:

```
getData(): Observable<any> {
    return this.http
```

```
      .get('http://' + host + ':' + port + '/data')
      .pipe(
        timeout(7500)
      );
  }
```

La funzione `getData()` ritorna un observable che contiene una variabile di tipo any che intende un oggetto noon precisato. In seguito alla `get()`, che ritorna un observable, è chiamata la funzione `pipe()` che permette di concatenare gli operatori funzionali della libreria RxJs, che ricevono un observable come parametro e ritornano un observable, permettendo la concatenazione. In questo caso l'operatore `timeout()` chiama `error()` sull'observable in caso non vengano emessi valori per il tempo indicato come parametro in millisecondi. In questo modo è possibile gestire gli errori di trasmissione dei dati a livello dei componenti che li desiderano ricevere.
All'interno del componente che intende utilizzare i dati ottenuti, viene iniettato il servizo e chiamata la funzione `getData()`.

```
fetchData(): void {
    this.service.getData().subscribe(
        data => {
            this.data = data;
        },
        error => {
            console.log('Error!')
        }
    )
}
```

Quando l'observable emetterà un dato, il valore ritornato verrà assegnato alla variabile `data` della classe mentre se venisse chiamata la funzione `error()` sull'observable, verrà stampato sulla console un messaggio di errore.
Quando un componente non intende più ricevere dati tramiete l'observable, chiama la funzione `unsubscribe()`. Solitamente questa operazione viene effettuata all'interno del metoto `onDestroy()` della classe che si è iscritta all'observable, che viene chiamato al momento della distruzione del componente.
Con il client Http di Angular questo passaggio non è necessario in quanto quando viene ricevuto il dato richiesto, viene chiamata sull'observable la funzione `complete()` che disiscrive gli observer ancora iscritti.

6.2.6 Il Routing

Dal momento che tutta l'applicazione viene eseguita in un'unica pagina, il framework ha il compito di gestire la navigazione tra le viste e riflettere lo stato dell'applicazione nel relativo URL. Il componente che gestisce questo aspetto in Angular è chiamato Router. All'interno del Router vengono definite le associazioni tra URL e componente che implementa la pagina. Navigando da un indirizzo all'altro all'interno dellapplicazione sarà possire passare alla visualizzazione della vista associata al componente desiderato.

All'interno delle URL è anche possibile definire dei parametri che potranno essere utilizzati all'interno dei componenti invocati. Questo aspetto risulta particolarmente comodo, ad esempio, volendo accedere ad un componente che mostra il dettaglio di un elemento di una certa collezione. In questo caso l'identificativo dell'oggetto verrebbe inserito nella URL e recuperata dal componente desiderato per richiedere i dettagli di quell'oggetto.
Un altro aspetto importante del Router è la capacità di imporre condizioni all'accesso a particolari URL tramite le Route Guards. Le Route Guards sono semplici classi Typescript che implementano l'interfaccia `CanActivate`.
Quest'interfaccia utilizza il metodo `canActivate()` per determinare preventivamente alla navigazione verso la pagina richiesta se l'utente che sta tentando di accedere alla URL detiene i permessi necessari. In caso affermativo la funzione torna `true` e viene visualizzata la pagina richiesta mentre in caso contrario è possibile dirottare l'utente su un'altra pagina o semplicemente negare la navigazione.

6.3 Gestione dell'autenticazione ed autorizzazione

Di seguito verranno descritto come sono stati implementati a livello di front end sistemi di autenticazione e autorizzazione dell'utente.

6.3.1 Autenticazione dell'utente

Al caricamento della web app, viene presentata la pagina di login. All'interno dei campi vanno fornite le credenziali di accesso con cui l'utente è stato registrato in modo tale che venga richiesto a Security Service il token JWT necessario all'autorizzazione delle successive richieste. Alla conferma viene chiamato il metodo `login()` dell'apposito servizio che riceve le credenziali inserite ed effettua la chiamata al servizio del back end autenticandosi tramite HTTP Basic Authentication.
Nella risposta è contenuto il token JWT che viene inserito all'interno di un oggetto LoginUser. Il token viene decodificato e vengono estratti l'username e le authorities del soggetto autenticato. L'oggetto viene salvato all'interno del Local Storage del browser in modo da risultare persistente per tutta la sessione di utilizzo. Il servizio di autenticazione fornisce due metodi ulteriori:

- `logout()` che cancella dal Local Storage l'oggetto LoginUser
- `getCurrentUserValue()` che fornisce come Observable l'oggetto LoginUser

6.3.2 Autenticazione delle richieste al back end

Una volta ottenuto il token JWT è necessario che questo venga inserito in ogni richiesta che viene effettuata al back end. Il framework Angular mette a disposizione un meccanismo che permette di intercettare le richieste HTTP effettuate e di manipolarne il contenuto prima che la richiesta sia effettivamente inviata. Classi che implementano questo meccanismo vengono chiamate Interceptor. Per implementare una classe di questo tipo è necessario implementare l'interfaccia HttpInterceptor.
Nel metodo `intercept()` vengono definite le operazioni effettuate preventivamente sulla

richiesta. Nel nostro caso nella richiesta viene inserito l'header Authorizazion contenente il token.
Questo meccanismo viene anche utilizzato per intercettare l'errore HTTP 401 Unauthorized. Un errore di questo tipo indica che l'utente ha tentato di accedere ad una risorsa a cui non è autorizzato. Pertanto viene effettuato il logout dell'utente e la redirezione alla pagina di login.

6.3.3 Protezione delle routes

In base all'authority del soggetto autenticato, sarà possibile accedere solo a certe pagine dell'applicativo. Gli Admin avranno un accesso completo e potranno apportare modifiche aggiungendo o rimuovendo utenti, modificando le configurazioni degli sniffer, aggiungendo rooms o buildings. Gli utenti normali potranno soltanto visualizzare i report.
Questo meccanismo viene definito a livello di routes tramite una classe che funge da Route Guard. A livello di route viene definita tramite il parametro `canActivate` una lista di classi che implementano l'interfaccia `Can Activate`.
Quest'interfaccia ha un metodo `canActivate()` all'interno del quale è possibile definire i criteri di accesso alla route. Nel nostro caso che l'utente descritto nell'oggetto LoginUser sia Admin.
Questo meccanismo viene anche usato per effettuare la redirezione alla pagina di login quando viene caricato l'applicativo. Se non si richiede direttamente la pagina `/login` viene caricato di default la home page, corrispondente alla route `/`. Applicando una Route Guard che prevede l'esistenza di un oggetto LoginUser nel Local Storage a questa route, è possibile forzare la navigazione verso la pagina di login nel caso questa condizione non venga rispettata.

6.4 Funzionalità del front end

Di seguito verranno descritte le funzionalità fornite dall'applicativo.
Una prima distinzione può essere effettuata tra le pagine che permettono di visualizzare report sui dati raccolti e quelle che permettono di operare su sniffer, building, room e utenti. La descrizione sarà suddivisa in base alle pagine che compongono l'applicativo e non i singoli componenti in quanto una descrizione dettagliata di ognuno (più di 30) risulterebbe inutilmente ripetitiva. In linea generale, il funzionamento dei componenti e dei servizi avviene secondo la descrizione generale riportata all'inizio del capitolo. La descrizione delle funzionalità è svolta secondo il punto di vista di un utente con Authority Admin. Nel caso di un utente normale si potrà accedere unicamente alle pagine relative ai report.

6.4.1 La dashboard

La dashboard 6.1 è il punto di ingresso dell'applicativo e permette di visualizzare alcune informazioni generali sullo stato di funzionamento del sistema. Il pulsante in alto a sinistra permette di rivelare una barra di navigazione dalla quale è possibile accedere alle pagine di gestione relative a sniffer, utenti e locations oppure tornare sulla dashboard.
Nella parte centrale della pagina è possibile visualizzare l'ultima stima del numero di device

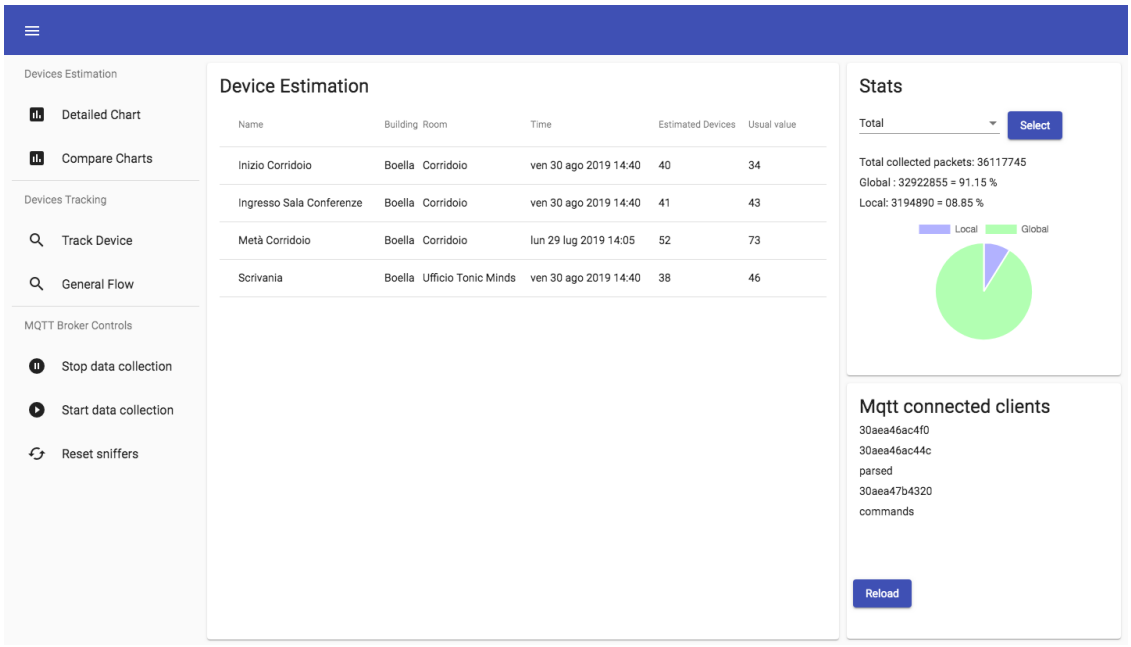


Figura 6.1: La dashboard

per ogni sniffer, insieme all'orario relativo alla stima ed al numero medio di device stimati relativi all'ora, al giorno della settimana e del mese considerato in modo da poter dare un valore di riferimento su cui basarsi per valutare un conteggio anomalo o nella norma.
Sulla destra, in alto, è presente una scheda in cui viene riportato il numero di pacchetti probe request raccolti e la percentuale tra essi di pacchetti contenenti indirizzi globally administered (soprannominati pacchetti global) e pacchetti contenenti indirizzi locally administered (soprannominati pacchetti local).
In basso è riportata la lista di client connessi al broker MQTT.
Sulla sinistra è presente una barra di navigazione tramite la quale è possibile accedere alle pagine di visualizzazione delle stime del numero di dispositivi e del tracciamento di questi. Nella sezione MQTT Broker Controls sono presenti alcuni pulsanti che permettono di avviare ed arrestare la raccolti di pacchetti (contattando gli end point del servizio apposito) e impartire a tutti gli sniffer connessi il comando di reset, effettuando una richiesta all'end point esposto dal servizio Data Explorer.

6.4.2 Pagine di gestione

Le pagine di gestione permettono di operare sui dati relativi ad utenti, sniffer e locations. Nei primi due casi la pagina è organizzata come una tabella in cui e possibile visualizzare le informazioni relative ai dati richiesti. Nel caso degli utenti (figura 6.3), in ogni riga della tabella sono riportati lo username, un indirizzo mail (se presente) e l'authority dello user: User, Sniffer o Admin. In ogni riga sono presenti dei pulsanti che permettono di visualizzare in una pagina dedicata le informazioni dettagliate dell'utente, effettuare modifiche o cancellare l'utente. Tramite il pulsante New User è possibile accedere alla

pagina di creazione di un nuovo utente. Nella pagina di gestione degli sniffer (figura 6.3)

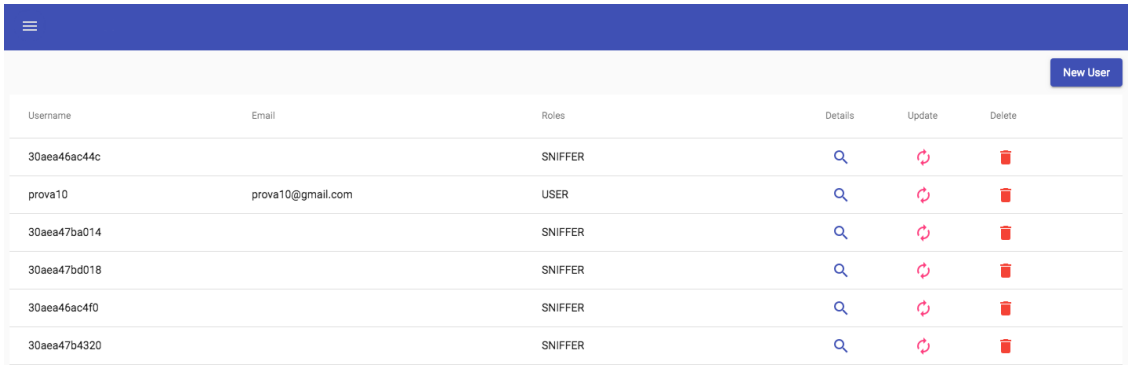


Figura 6.2: Pagina di gestione degli users

sono riportati, per ogni sniffer, il nome, il MAC address del device fisico utilizzato, la posizione dello sniffer espressa tramite room e building e lo stato dello sniffer che può essere Connected in caso lo sniffer sia correttamente connesso al broker e funzionante, Disconnected in caso non presente tra i client connessi al broker e Detached in caso si sia effettuata la procedura di distacco descritta nal capitolo precedente. Per ogni sniffer è possibile accedere alla pagina che mostra il dettaglio dei dati dello sniffer compresa la configurazione (che viene inviata allo stesso in fase di avvio dello sniffer), modificare la configurazione, effettuare la procedura di distacco o resettare lo sniffer, nel caso in cui questo sia connesso. Questa funzione è utilizzata quando viene effettuato lìupdate della configurazione dello sniffer che, riavviandosi, viene configurato con i nuovi dati inseriti. Sopra la tabella sono presenti alcuni pulsanti che hanno le seguenti funzioni:

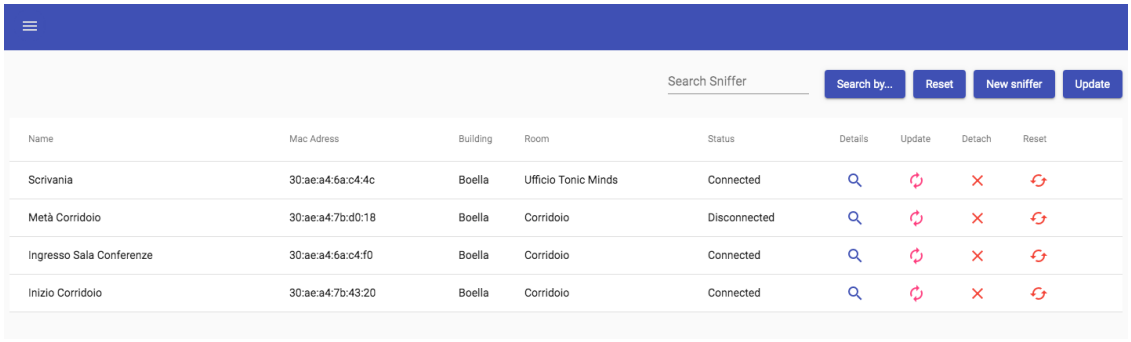


Figura 6.3: Pagina di gestione degli sniffers

- New Sniffer permette di accedere alla pagina di creazione di un nuovo sniffer. In questa fase lo sniffer può essere o non essere associato ad un device fisico, nel secondo caso risulterebbe in stato detached.
- Update forza l'aggiornamento dei dati visualizzati nella tabella
- Search By... permette di filtrare la lista degli sniffer per nome, MAC address, building e room indicando il criterio di ricerca nel campo di testo adiacente

- Reset permette di annullare l'operazione di filtraggio, ritornando alla visualizzazione di tutti i dati

6.4.3 Pagine di visualizzazione dei report

Le pagine di visualizzazione dei report permettono la visualizzazione dei grafici relativi al conteggio dei dispositivi ed al tracciamento dei dispositivi in forma aggregata ed isolata. Nella barra di navigazione della dashboard sono presenti quattro pulsanti che permettono di accedere alla visualizzazione di queste pagine, suddivisi nelle categorie Device Estimation e Device Tracking. Il primo pulsante di Device Estimation, denotato dalla scritta Detailed Chart permette di accedere alla visualizzazione dettagliata di un report (figura 6.4).
La pagina presenta sulla sinistra un form in cui è possibile definire i parametri della ricerca. É possibile aggregare i conteggi per building o room oppure visualizzare i conteggi relativi ai pacchetti raccolti da uno specifico sniffer. É possibile indicare una data ed un orario su cui restringere la visualizzazione o un intervallo di date. In fine è possibile indicare il livello di aggregazione dei dati visualizzati (chiamata in questo caso "risoluzione") che varia da 5 minuti, che è l'intervallo di conteggio standard, a 15 minuti, 30 minuti, 1 ora, 2 ore, 4 ore, 6 ore e 12 ore. Indicando un intervallo di aggregazione maggiore di 5 minuti viene effettuata la media dei conteggi che rientrano nell'intervallo indicato. Il conteggio

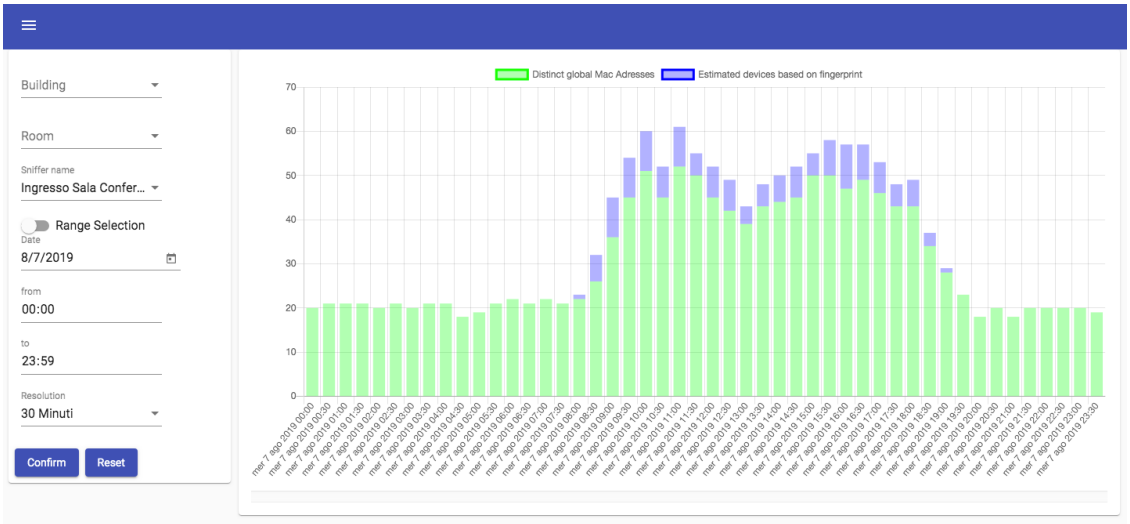


Figura 6.4: Visualizzazione dei report sui conteggi

dei device viene rappresentato tramite un istogramma che presenta sulle ascisse l'intervallo di tempo considerato e sulle ordinate il numero di device stimati. In verde è riportata la parte corrispondente a device che si annunciano con il loro indirizzo globally administered mentre, in blu, la parte dei device stimata a partire dal conteggio dei fingerprint univoci.
Il secondo pulsante, compare Chart, permette di accedere ad una pagina in cui, a differenza del caso precedente, è possibile indicare più sniffer in modo da poter comparare il tracciato visualizzato.
Il pulsante Track Device permette di accedere alla pagina in cui vengono visualizzate li informazioni relative al tracciamento di un device. Questa pagina presenta due tab.

Nel primo (figura 6.5) è possibile visualizzare le informazioni sulla posizione stimata del device tramite heatmap mentre nel secondo tab (figura 6.6) è possibile visualizzare su uno scatter plot, per ogni pacchetto probe request emesso da un device , lo sniffer che l'ha raccolto (quindi più prossimo al device) ed il riferimento temporale relativo al momento della cattura. É necessario sottolineare che queste visualizzazioni si basano soltanto sui pacchetti che contengono indirizzi globally administered e che quindi risultano tracciabili nel tempo. Come verrà spiegato nel prossimo capitolo, non è stato possibile individuare una metodologia che permettesse di effettuare il tracciamento di un dispositivo che maschera il proprio indirizzo MAC. Dal momento che in una giornata possono essere raccolti migliaia di

Figura 6.5: Posizione stimata di un device tramite heatmap. Le tre zone evidenziate denotano lo spostamento del device nell'intervallo di tempo considerato

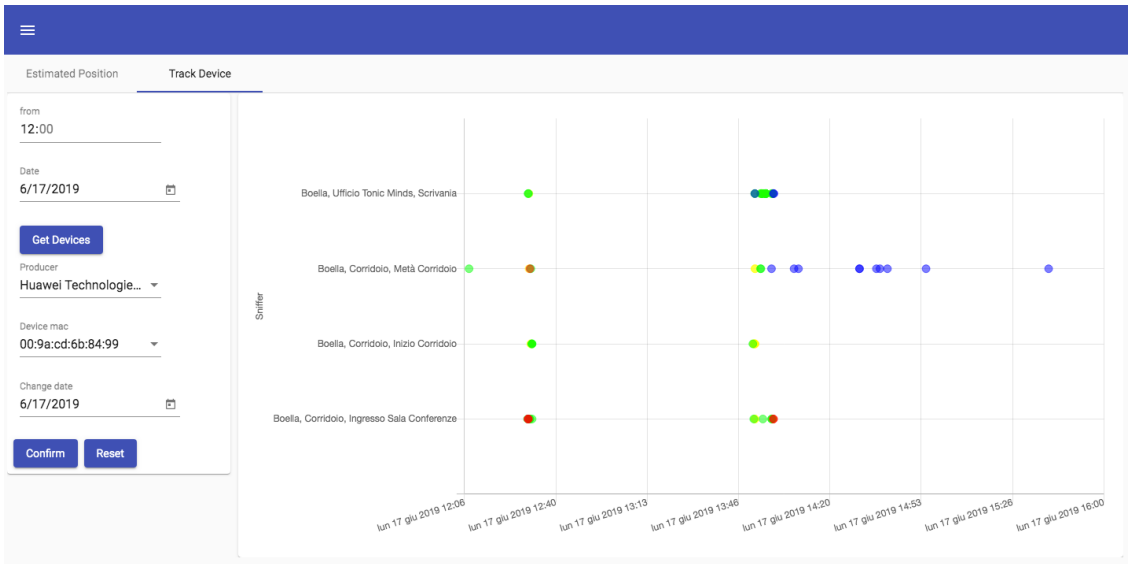


Figura 6.6: Tracciamento di un device

MAC address distinti, è stata implementata una funzione di ricerca che permette di limitare la ricerca di un dispositivo su un intervallo di un ora. Selezionando una data ed un ora,

premendo il pulsante Get Devices verrà ottenuta la lista di tutti i MAC address dei device rilevati in quell'intervallo di tempo suddivisi per produttore in modo da permettere una ricerca più efficace. Ad esempio selezionando come producer un produttore di smartphone saremo sicuri di isualizzare dati proenienti effettivamente da uno smartphone.
Una volta selezionato un indirizzo mac è possibile variare la data per visualizzare i dati relativ ad altre giornate.
Nello scatter plot vengono visualizzati tutti i pacchetti provenienti da un certo device raccolti in 24 ore. Ovviamente verranno la visualizzazione verrà ridotta al solo intervallo di tempo che contiene effettivamente dei pacchetti. Ad esempio se il primo pacchetto è stato catturato alle ore 10:00, avremo che la visualizzazione dei dati partirà da questo momento.
Nel caso della heatmap i pacchetti raccolti vengono aggregati su intervalli di un minuto e tramite le frecce sulla destra è possibile muoversi in avanti ed indietro nel tempo. Vengono visualizzati soltanto i minuti all'interno dei quali sono stati effettivamente raccolti dei pacchetti. Sotto le frecce è riportata l'ora a cui fanno riferimento i dati visualizzati. Nel caso dell'esempio sono evidenti tre aree con una colorazione più intensa. Questo caso può essere spiegato col fatto che il device che emetteva i pacchetti si è spostato lungo il corridoio e all'interno del minuto considerato è stato rilevato con un segnale di buona intensità da tre sniffers.
L'ultimo pulsante permette di accedere alla visualizzazione che aggrega dati raccolti da tutti i device presenti in un certo intervallo di tempo e visualizza una heatmap che dovrebbe indicare le aree con una concentrazione di dispositivi maggiore (figura 6.7). In questo

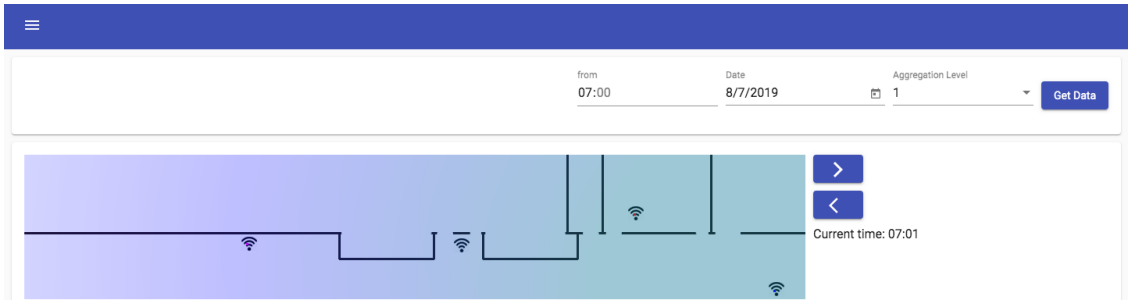


Figura 6.7: Heatmap generale che mostra le aree con un numero maggiore di device

caso i dati utilizzati per la visualizzazione della heatmap sono stati limitati ad un ora per volta per poter avere buone prestazioni. Indicando la data e l'ora è possibile visualizzare le zone con presumibilmente una concentrazione di device maggiore. Come nel caso dei singoli device, i dati sono raccolti secondo finestre della durata di un minuto. É possibile aggregare questi dati per intervalli di tempo maggiori indicando nel form un intervallo di aggregazione maggiore.
Nel prossimo capitolo verrà discussa l'efficacia di questi sistemi di visualizzazione.

6.5 Deploy

Angular mette a disposizione un'interfaccia a riga di comando chiamata Angular CLI, richiamabile tramite il comando `ng`, che permette di gestire le varie fasi di vita di un progetto Angular, dall'inizializzazione alla creazione di servizi e componenti, dall'avvio di un server di test in locale alla pacchettizzazione dei file in modo da permetterne in deploy. Tramite il comando `ng build` si otterrà una cartella contenente tutti i file necessari al funzionamento dell'applicativo web. Tramite Docker è stata creata un'immagine costruita a partire da quella di Nginx, un comune server web, per poter ospitare l'applicazione.
Nel Dockerfile sono stati definiti i comandi necessari alla copia della suddetta cartella e dei file di configurazione del server web all'interno dell'immagine risultante. Tramite il comando `docker run` è stata avviata l'istanza del server che rimane in ascolto di nuove richieste sulla porta 8082. Il container, a differenza di quelli che costituiscono il back end, non è collegato alla stessa rete virtuale pertanto le chiamate al back end passano dal localhost. In caso di esecuzione su un'altra maccchina sarà sufficiente variare l'indirizzo e la porta dell'api gateway definiti nelle variabili d'ambiente nel file environment.ts.

Capitolo 7

Analisi dei risultati

7.1 Introduzione

Nel seguente capitolo verranno analizzati i risultati ottenuti per quanto riguarda il conteggio, il tracciamento e l'individuazione delle aree più affollate tramite heatmap oltre ad alcune considerazione più generali riguardanti i risultati ottenuti.

7.2 Setup dell'esperimento

La raccolta dei dati è iniziata nel mese di marzo del 2019 all'interno dell'Istituto Mario Boella. Sono stati utilizzati quattro sniffer disposti nei punti indicati nello schema di seguito (figura 7.1). I due punti verdi indicano la posizione degli access point utilizzati per

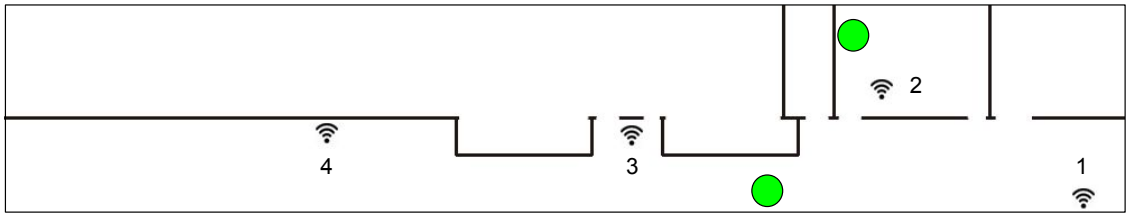


Figura 7.1: Schema del posizionamento degli sniffer

fonrire connettività agli sniffer.
Di seguito è riportata l'associazione tra il numero che indica lo sniffer nella figura e lo sniffer registrato:

- Sniffer 1: Building: Boella, Room: Corridoio, Name: Ingresso Sala Conferenze
- Sniffer 2: Building: Boella, Room: Ufficio Tonic Minds, Name: Scrivania
- Sniffer 3: Building: Boella, Room: Corridoio, Name: Metà Corridoio
- Sniffer 4: Building: Boella, Room: Corridoio, Name: Inizio Corridoio

Sfortunatamente, già a pochi metri dall'access point, gli sniffer non sono stati in grado di mantenere una connessione stabile e pertanto è stato necessario l'acquisto di un range extender per permettere di posizionare gli sniffer ad una distanza significativa. Nonostante la relativa vicinanza tra gli sniffer, è stato comunque possibile evidenziare una buona differenza tra i dati raccolti dagli sniffer nello stesso momento ma è comunque comune che lo stesso pacchetto venga catturato da diversi sniffer.
Sono stati raccolti dati in modalità dump nel mese di marzo e di luglio. Le prime osservazioni hanno permesso di modificare il sistema di fingerprinting per migliorare il conteggio dei dispositivi.

7.3 Dataset utilizzati

Purtoppo sono difficilemente reperibili dataset pubblici, data comunque la natura sensibile dei dati contenuti all'interno dei frame probe request, e quelli reperibili sono vecchi di alcuni anni, rendendoli meno rappresentativi della situazione odierna, dato che negli ultimi anni è cresciuta la quantità di dispositivi che utilizzano sistemi di mascheramento dell'indirizzo MAC. Per effettuare le analisi descritte nelle pagine successive sono stati utilizzati due dataset raccolti nel periodo di marzo (da ora D1) e luglio del 2019 (da ora D2) sfruttando uno sniffer in modalità dump.
Un terzo dataset è composto dai pacchetti raccolti in modalità parsed che sono stati raccolti a cavallo di diversi mesi. Questi ultimi dati sono stati usati per realizzare i grafici dei conteggi ed il tracciamento dei dispositivi. Il contenuto risultante della cattura è riportato nella seguente tabella 7.2:

Dataset	# pacchetti raccolti	% pacchetti global	% pacchetti local	# indirizzi global distinti	# indirizzi local distinti	giorni di raccolta
D1	403597	87,6 %	12,4 %	2095	29589	8
D2	895594	96,3 %	3,7 %	913	13456	10
Parsed	36048928	91,15 %	8,85 %	30307	1135052	da 7/05 al 9/08 con 4 sniffer

Figura 7.2: Composizione dai dataset

7.4 Osservazioni sui dati raccolti

Analizzando il contenuto del dataset è possibile giungere ad alcune interessanti conclusioni. Innanzi tutto è possibile notare come il D2 comprenda un numero di indirizzi MAC sia globali che locali dimezzato rispetto a D1 data la fine delle lezioni ed il minor numero di gente in generale. Analizzando la distribuzione degli indirizzi global si può notare come in entrambi i dataset la maggior parte dei pacchetti local provenga da un insieme ristretto di dispositivi. In D1 dieci dispositivi sono l'origine di 195013 pacchetti corrispondenti a circa

il 48 % dei pacchetti raccolti. Gli OUI associati agli indirizzi MAC di questi device non sono riconducibili necessariamente a marche di smartphone ma rientrano soprattutto tra dispositivi Raspberry Pi, telecamere di sicurezza ed altri dispositivi con funzionalità WiFi. Tra questi dieci device, da un dispositivo Apple sono stati catturati 14032 pacchetti, mentre guardando le posizioni successive possiamo iniziare a trovare degli indirizzi riconducibili a smartphone delle più diffuse marche, da HTC a Samsung, da Huawei a OnePlus.
Questo dimostra come l'adozione di sistemi di mascheramento degli indirizzi MAC non sia ancora implementata su un grande numero di device o che comunque questi usano anche il loro indirizzo MAC globally administered.
Per quanto riguarda il dataset D2 abbiamo una situazione similare anche se in questo caso un dispositivo HP ha emesso ben 290747 corrispondenti a circa il 32 % dei pacchetti raccolti ed un dispositivo Espressif che ne ha emessi 231647 corispondenti a circa il 26 %. In entrambi i dataset, il numero di pacchetti raccolti per ogni dispositivo decresce velocemente fino ad ottenere che la maggior parte dei dispositivi osservati hanno emesso meno di 10 pacchetti. In D1 ricadono in questa categoria 1346 dispositivi, corrispondenti a circa il 64 %, mentre in D2 abbiamo 686 device che corrispondono al 75 %. Questo è dovuto al fatto che molti di questi pacchetti sono stati catturati in modo spurio da dispositivi di passaggio.
I pacchetti contenenti un indirizzo MAC locally administered presentano due tipi di OUI: Google e Unknown.
Nel primo caso ricadono i dispositivi Android che implementano il mascheramento del MAC, come da linee guida utilizzando l'OUI DA:A1:19 mentre nel secondo caso ricadono dispositivi che randomizzano il loro indirizzo MAC ma che non generano collisioni con altri OUI già registrati. Analizzando questi pacchetti è possibile notare che alcuni contengono un tag vendor specific registrato con un OUI della Apple pertanto si può concludere con certezza che siano dispositivi di questa marca. In D1 abbiamo 24759 pacchetti raccolti con OUI Unknown ed il 72 % circa contengono il tag Apple. In D2 invece 23335 pacchetti e 40 % di tag Apple. Un ultima osservazione va fatta a riguardo del contenuto di SSID all'interno dei pacchetti probe request.
Dalle analisi effettuate risulta che in D1 ben 78026 pacchetti global e 5691 pacchetti local contengono un SSID per un totale di 642 SSID distinti mentre in in D2 abbiamo 580716, 8276 e 281 SSID distinti. Questo dimostra che è comunque comune inserire gli SSID anche nei pacchetti local e questo espone ad una serie di rischi. Dal momento che un device emette queste informazioni è possibile inferire molti dati sul possessore del dispositivo andando a localizzare l'origine delle reti annunciate utilizzando servizi di localizzazione basati su WiFi come Wigle o anche solo osservando SSID di locali pubblici, palestre o università. Analizzando i dati è stato possibile individuare che molti device che mascherano il loro MAC emettono all'interno di un burst degli SSID che possono essere usati per tracciarlo in quanto quest'insieme di reti annunciate può essere molto specifico.
Di seguito sono riporatti degli esempi:

```
{'mac': ['82:91:c1:78:20:a5', '2e:de:ed:43:64:52', '6a:c5:ef:04:56:55'],
  'ssid': ['FASTWEB-1-0021966FD59C',
           '4487411846050991',
           'presidenza',
           'WOW FI - FASTWEB',
```

```
            'sitipianoterra',
            'ospiti@ismb.it']}

{'mac': ['5e:49:da:74:15:8a', 'b6:f2:08:29:69:bb', '9a:77:ee:d8:2e:3c'],
  'ssid': ['BWAtLakePowell',
              'BW Grand Canyon Squire Inn',
              'Serrano Hotel']}
```

Questo approccio potrebbe rivelarsi interessante come modalità di tracciamento di dispositivi e andrebbe ulteriormente approfondita.
Analizzando la composizione dei tag all'interno dei pacchetti global e local è evidente che questa varia in base al tipo di pacchetto, come evidenziato in figura 7.3. Per entrambi i dataset è evidente l'assenza del tag WPS, identificato dalla sigla dd0050f204, all'interno dei pacchetti local. Questo tag è molto utile al fine della deanonimizzazione di un dispositivo [11]. All'interno del tag è contenuto un identificativo chiamato UUID (Universally Unique Identifier) che è costruito a partire dal MAC address del dispositivo. É stato dimostrato che tale UUID può essere utilizzato per derivare il MAC del device rendendo inutile l'eventuale randomizzazione del MAC. La completa assenza di questo tag all'interno dei pacchetti local denota che sono stati presi provvedimenti a riguardo da parte dei produttori dei dispositivi.

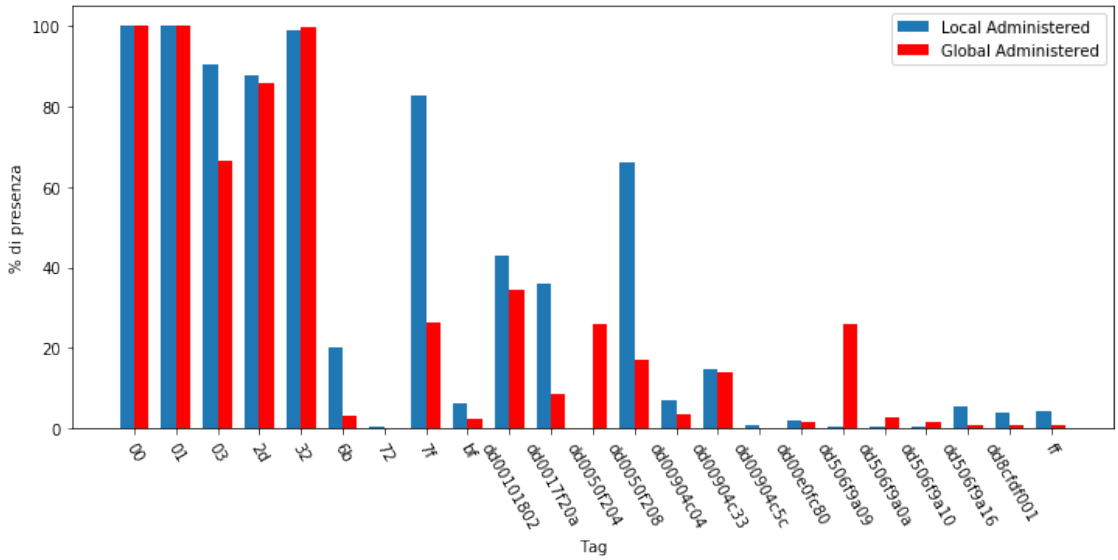
7.5 Capacità di tracciamento dei dispositivi

7.5.1 Metodo d'analisi

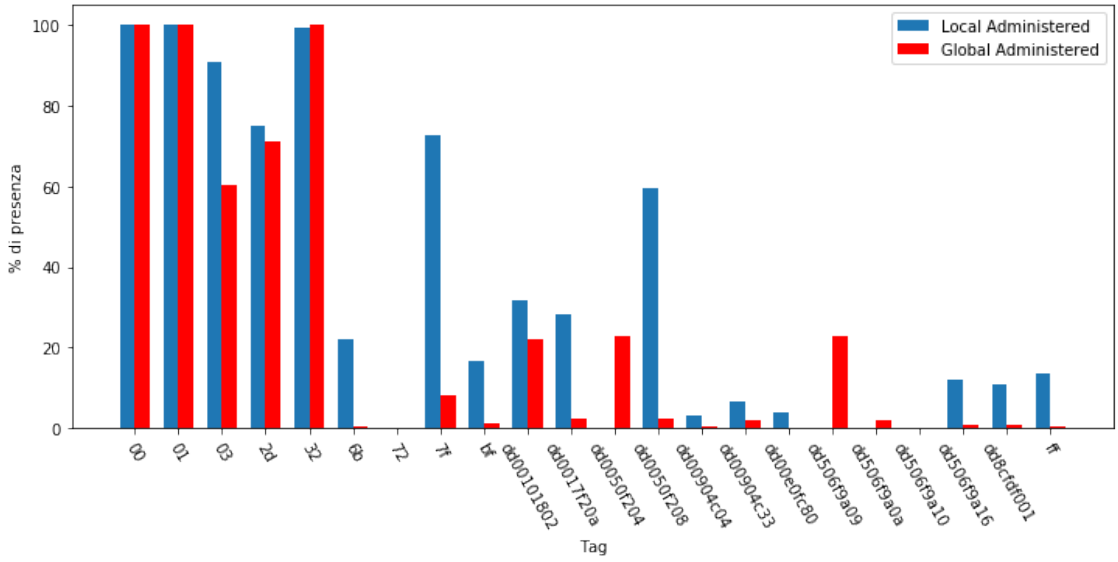
Come descritto nei capitoli precedenti, identificare una metodologia in grado di deanonimizzare un dispositivo che maschera il proprio indirizzo MAC per renderlo tracciabile è tutt'altro che triviale. Non potendo identificare con sicurezza un dispositivo che utilizza queste tecniche, dal momento che l'indirizzo MAC viene cambiato all'incirca ad ogni invio di pacchetti, si è deciso di testare la tecnica di fingerprinting basata su information elements almeno sui pacchetti provenienti da dispositivi che usano l'indirizzo MAC globally administered, in modo da poter avere un valore di riferimento. All'intreno di D1 e D2 è stato calcolato il fingerprint anche per i pacchetti global in modo tale da verificare in quanti casi un dispositivo è univocamente identificato da un fingerprint. Per ogni dataset si è applicato il metodo basato su anonimity set [13]:

- Per ogni fingerprint presente nel database e stato creato un set contenente tutti gli indirizzi mac presenti nei pacchetti probe request che li generavano
- É stata calcolata la grandezza di ogniuno di questi set e aggregato i risultati secondo questo valore
- Tutti gli indirizzi mac che facevano parte di un anonimity set della stessa grandezza sono stati inseriti a loro volta in un set, eliminando così eventuali ripetizioni
- É stata calcolata la grandezza di questi set

Alla fine di questo processo si ottiene il numero di device che fa parte di anonimity set di dimensione n.



(a) D1



(b) D2

Figura 7.3: Comparazione tra percentuale di presenza dei Tagged Parameters

Più device fanno parte di un anonimity set di dimensione 1 più device vengono identificato univocamente da un fingerprint. I risultati ottenuti sono presenti nei grafici 7.4 e 7.5.
L'andamento dei grafici è comparabile al comportamento descritto negli articoli citati precedentemente: il primo picco evidenzia molti device identificati da solo un fingerprint o comunque un numero ristretto di essi.
Spostandoci più avanti nel grafico otteniamo un secondo picco che è spiegato dalla presenza di molti device che condividono lo stesso fingerprint. Nel caso di D2 è presente un anonimity

set di dimensione 304 che è composto da 304 dispositivi. Questo significa che tutti questi dispositivi condividono lo stesso fingerprint e pertanto non sono distinguibili usando questa tecnica. Verificando l'OUI di questi device si è appreso che sono tutti prodotti da Motorola. Nonostante la grande diffusione di questo brand è comunque singolare la presenza di un così grosso numero di device di questa marca. É possibile che questi device implementino uno schema di mascheramento dell'indirizzo MAC non standard che porta ad avere un numero di indirizzi MAC maggiore del numero di device effettivi nonostante l'indirizzo risulti globally administered.
É evidente la principale limitazione di questo metodo per effettuare il tracciamento di un dispositivo in quanto il fingerprint è calcolato su dati che sono specifici del modello e del software del dispositivo. Un dispositivo particolarmente diffuso sarà difficilmente tracciabile con questo approccio in quanto sarà impossibile distinguerlo da altri modelli. Pertanto anche se un device appare con un fingerprint univoco, col passare del tempo la prbabiltà che si presenti un altro device che genera pacchetti con lo stesso fingerprint aumenta rendendolo indistinguibile da quello precedente.

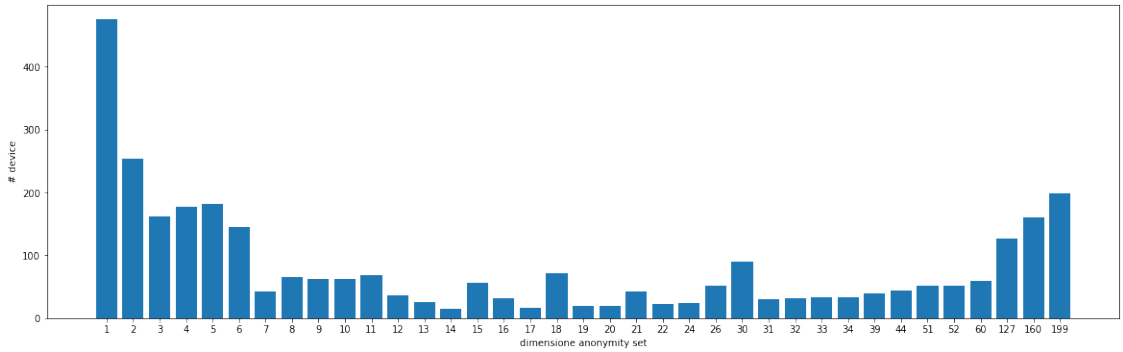


Figura 7.4: Anonimity set D1

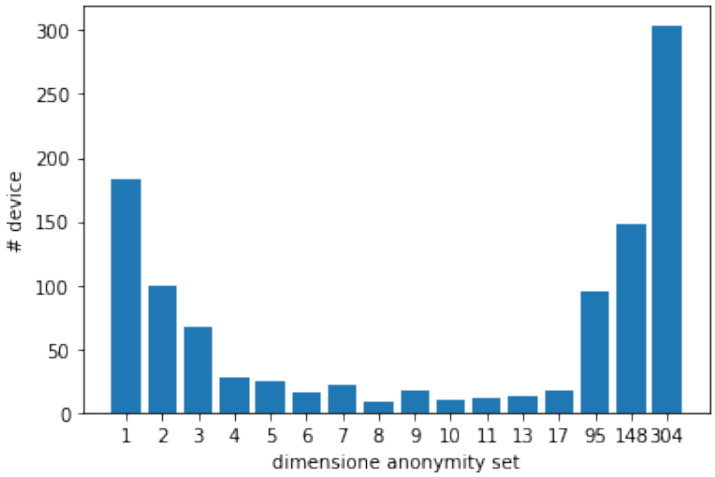


Figura 7.5: Anonimity set D2

7.5.2 Risultati relativi al tracciamento dei dispositivi con indirizzo MAC locally administered

I risultati descritti precedentemente denotano come solo una piccola percentuale dei fingerprint calcolati possono essere utilizzati per tracciare in modo univoco un device. Per quanto riguarda device che emettono pacchetti con indirizzi locally administered, non possiamo supporre di ottenere risultati migliori di questi.
Pertanto è evidente che questo metodo, che già mostra la sua debolezza nel caso degli indirizzi globally administered, non sia utilizzabile per la deanonimizzazione di un device che randomizza il proprio indirizzo MAC, per effettuarne un tracciamento continuativo nel tempo.

7.5.3 Risultati relativi al tracciamento dei dispositivi con indirizzo MAC globally administered

Come affermato nel capitolo precedente, è comunque presente all'interno dei dataset raccolti in fase di test e in fase di funzionamento normale un gran numero di device che si annunciano usando il loro indirizzo mac globally administered. Questo permette di tracciare il loro spostamento all'interno dell'area di osservazione semplicemente ricercando il loro indirizzo MAC all'interno del dataset.
Per visualizzare questi dati, tramite il client è possibile accedere ad una pagina che utilizza uno scatter plot, dove sulle ascisse è presente l'orario di raccolta del pacchetto mentre sulle ordinate è presente il riferimento allo sniffer che ha catturato quel pacchetto. Il colore del dato nel grafico indica l'intensità del segnale percepito. I pacchetti raccolti che presentano un rssi inferiore -90 dBm vengono indicati con un colore blu, quelli compresi tra -90 dBm e -75 dBm con il verde, tra -75 dBm e -60 dBm con il giallo e oltre -60 dBm con il rosso.
Il principale problema riscontrato analizzando i dati raccolti è che alcuni dei pacchetti vengono raccolti contemporaneamente da diversi sniffer. Questo è dovuto principalmente al fatto che gli sniffer sono posizionati ad una distanza troppo piccola tra loro. Nel grafico 7.6 è riportato a titolo di esempio il grafico generato dai pacchetti raccolti da un particolare dispositivo che si annuncia con indirizzo globally administered. Dal grafico si evince la posizione in cui è stazionato il dispositivo e gli spostamenti effettuati, come nell'intervallo di tempo tra circa le 11 e le 11 e 30, dove è evidente che il dispositivo ha stazionato in una posizione diversa che ha permesso ad altri sniffer di catturare dei pacchetti.
Durante le osservazioni è stato possibile individuare alcuni device che emettono frame Probe Request in continuazione. Si presume che questi device siano principalmente stampanti, telecamere di sorveglianza o dispositivi embedded come Raspberry Pi. Nell'immagine 7.7 che riporta il caso di uno di questi dispositivi, è possibile notare la differenza di comportamento rispetto al device precedente.

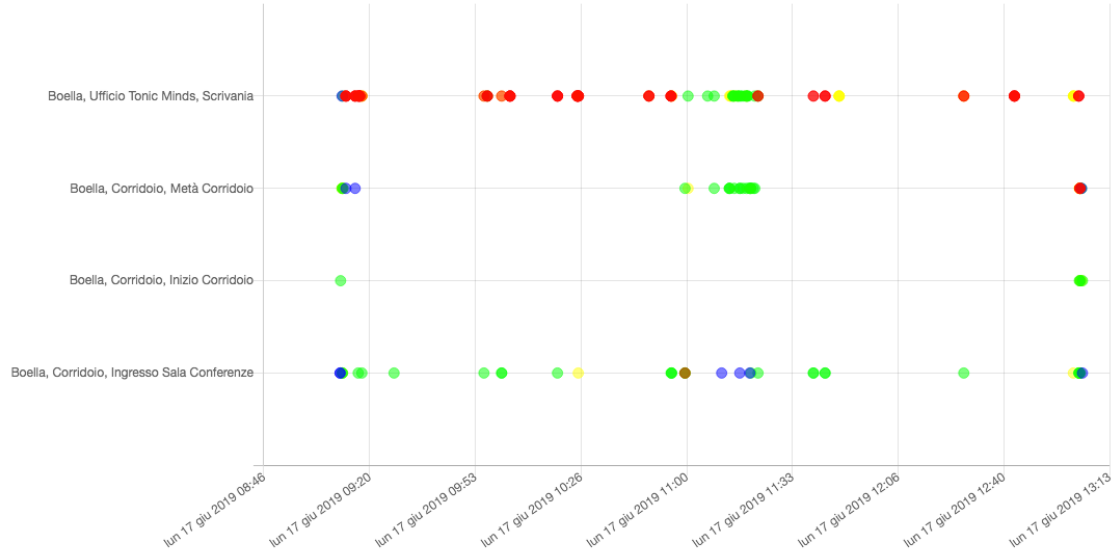


Figura 7.6: Tracciamento di un dispositivo

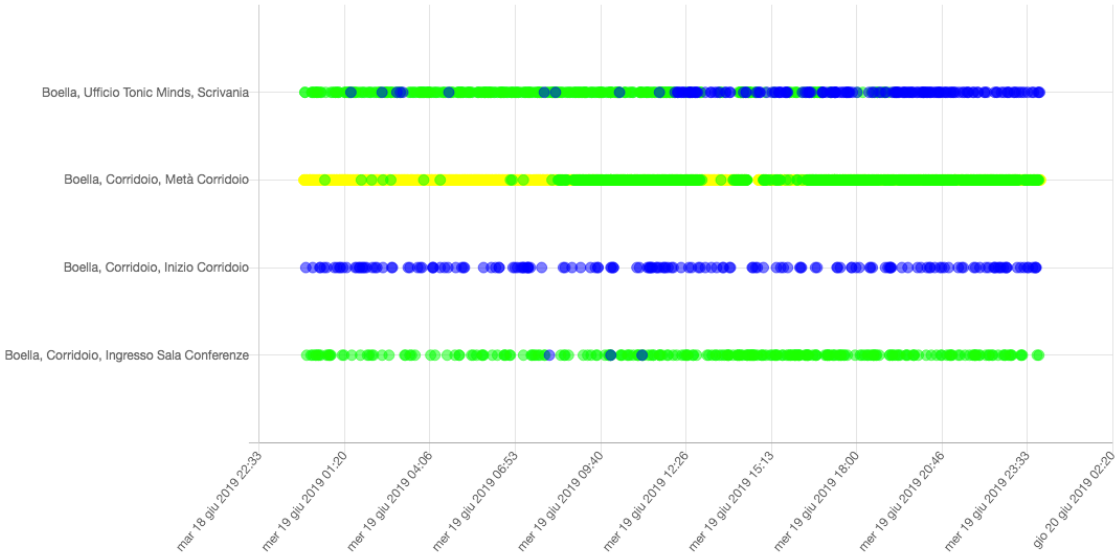
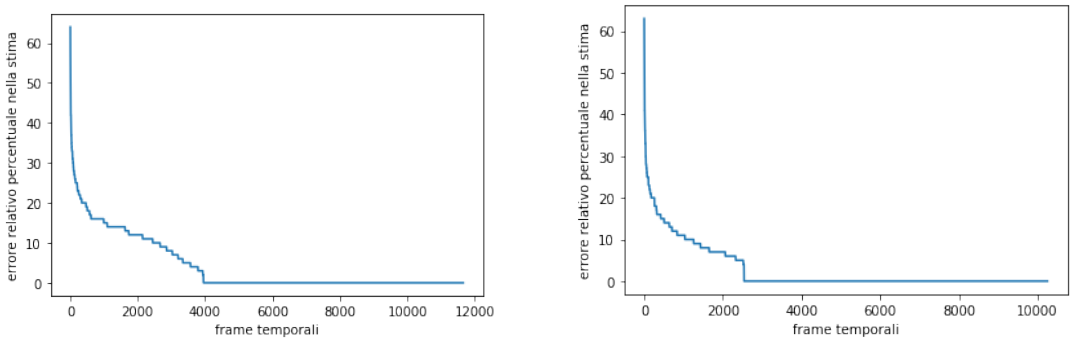


Figura 7.7: Tracciamento di un dispositivo che emette un numero anomalo di probe request

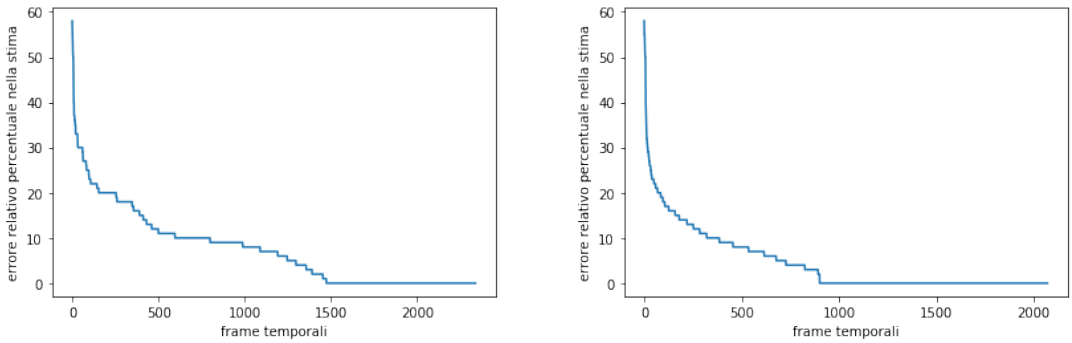
7.6 Capacità di conteggio dei dispositivi

Come affermato in precedenza, l'utilizzo di questo approccio per tracciare i dispositivi è minato dal fatto che i fingerprint vengono calcolati su dati non abbastanza univoci per permettere di distinguere modelli diversi dello stesso dispositivo o comunque dispositivi diversi che emettono pacchetti probe request contenenti le stesse informazioni. Tuttavia l'utilizzo di questo sistema per conteggiare i dispositivi presenti in un certo intervallo di tempo sembra dare risultati più soddisfacenti. Per i due dataset considerati, sono stati conteggiati i fingerprint calcolati sui pacchetti global, in modo da avere un valore di riferimento (sempre partendo dal presupposto che ad ogni device corrisponda un solo indirizzo mac globally administered) e il numero di indirizzi mac globally administered univoci in intervalli di tempo di 1, 5 e 15 minuti.
Per ogni intervallo di tempo è stato calcolato l'errore relativo percentuale tra il numero di MAC address distinti ed il numero di fingerprint distinti.
Tali errori sono stati ordinati in maniera discendente e rappresentati nei grafici per D1 e per D2. Nella tabella 7.1 è descritta, per ogni dataset, la percentuale di frame che



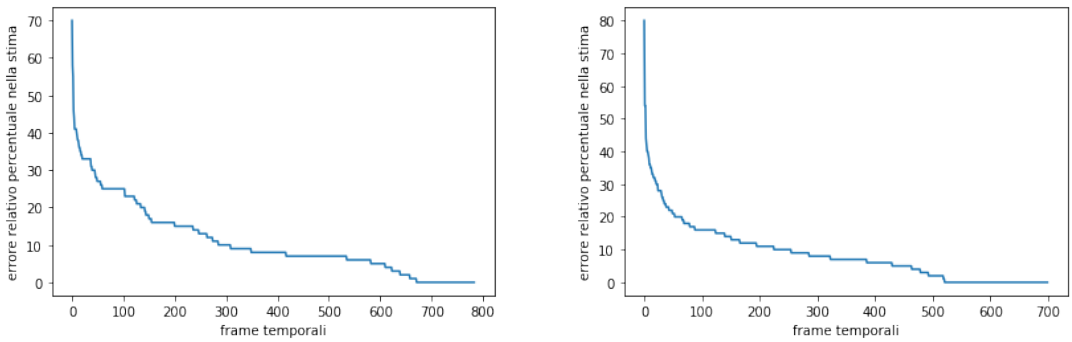
(a) D1 (b) D2

Figura 7.8: Errore relativo nella stima dei dispositivi per intervalli di 1 minuto



(a) D1 (b) D2

Figura 7.9: Errore relativo nella stima dei dispositivi per intervalli di 5 minuti



(a) D1 (b) D2

Figura 7.10: Errore relativo nella stima dei dispositivi per intervalli di 15 minuti

presentano una stima di errore inferiore al 5%, 10% e 15% per frame temporali di 1, 5 e 15 minuti. Da questi dati è possibile trarre alcune osservazioni.

Dataset	Errore relativo	1 Minuto	5 Minuti	15 Minuti
D1	<5 %	71,38 %	46,69 %	25,76 %
	<10 %	79,06 %	74,52 %	63,65 %
	<20 %	97,13 %	93,42 %	82,91%
D2	<5 %	77,33 %	67,26 %	38,57 %
	<10 %	89,92 %	84,50 %	67,86 %
	<20 %	98,31 %	96,67 %	92,43 %

Tabella 7.1: Percentuale di errore di stima per frame temporali

Innanzitutto è evidente come l'errore di stima diminuisca con il diminuire della dimensione dei frame temporali. Questo è dovuto al fatto che, considerando un intervallo di tempo più ristretto, è meno probabile che due device emettano pacchetti che generano lo stesso fingerprint. Infatti aumentando l'ampieza degli intervalli temporali di osservazione aumenta l'errore relativo percentauale.
L'intervallo temporale utilizzato fino ad ora per effettuare i conteggi in regime di funzionamento normale è infatti di 5 minuti. Questo tempo dovrebbe garantire che una buona percentuale dei device presenti nella zona di osservazione emettano almeno un Probe Request. Basandoci anche sull'affermazione di Lim et al. [14] questo intervallo di tempo è sembrato ragionevole. Tuttavia va sottolineato che l'intervallo con cui gli smartphone emettono Probe Request non è facilmente prevedibile dal momento che dipende dallo stato in cui si trova il device (in uso o no, connesso ad una rete o no), e pertanto è presente una forte variabilità nei tempi che intercorrono tra l'emissione di due burst consecutivi. Pertanto, la finestra temporale utilizzata attualmente per il conteggio, potrebbe non risultare molto significativa per ottenere conteggi del numero dei device in termini assoluti, ma piuttosto in termini relativi.
Va nuovamente sottolineato che, anche in questo caso, queste stime sono state effettuate utilizzando solo pacchetti che contengono indirizzi globally administered visto che è l'unico modo per avere un valore di riferimento.

Se avessimo voluto valutare la bontà della stima del numero di device che implementano sistemi di mascheramento dell'indirizzo MAC avremmo dovuto effettuare la raccolta in un ambiente estremamente controllato in modo da poter conoscere l'effettivo numero di dispositivi. Un esperimento del genere risulta comunque impossibile da effettuare con un numero di dispositivi ed in condizioni rilevanti pertanto dobbiamo basarci solo sulle resa delle stime effettuate sui device con indirizzi globally administered e considerare questo come caso migliore per i device locally administered.
Pertanto possiamo concludere che le stime effettuate in regime di funzioanemnto normale sul numero di device che usano indirizzi locally administered, possono al meglio approssimare per difetto il vero numero di dispositivi, con un errore che, nelle migliori delle ipotesi, dovrebbe essere simile a quello espresso nella tabella 7.1.
Va inoltre tenuto in considerazione che il seguente metodo permette di conteggiare in modo approssimato il numero di device ma non necessariamente ad un device corrisponde una persona, anche se il numero di questi sarà fortemente correlato con il numero di persone effettivamente presenti.

7.6.1 Visualizzazione dei dati

Come descritto nel capitolo precedente, il client permette di visualizzare i dati sulle stime del numero di dispositivi tramite un istogramma. Per ogni intervallo temporale considerato, in verde è riportato il numero di MAC univoci annunciati nell'intervallo di tempo e, di conseguenza, il numero di devicementre in blu il numero di device stimati conteggiando i fingerprint distinti.
L'immagine 7.11 riporta i conteggi effettuati nel corso di 24 ore. Tramite il client è possibile

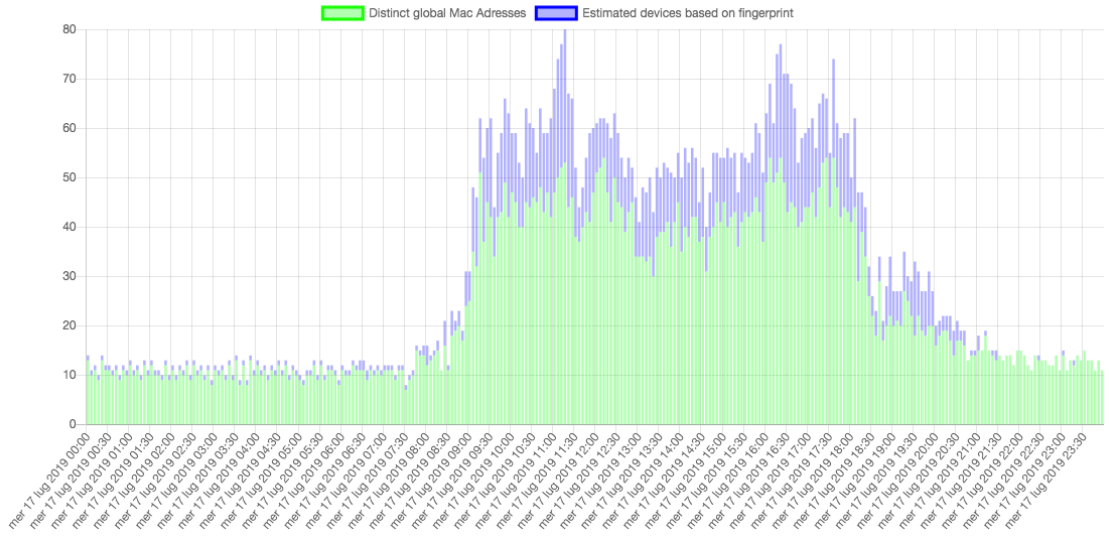


Figura 7.11: Stime del numero di dispositivi in intervalli di 5 minuti

eseguire ulteriori aggregazioni dei dati per permettere una visualizzazione migliore dei dati, aggregando per intervalli di tempo maggiore, per room o building. Effettuando queste aggregazioni, il conteggio risultante è dato dalla media dei valori considerati. Nella figura

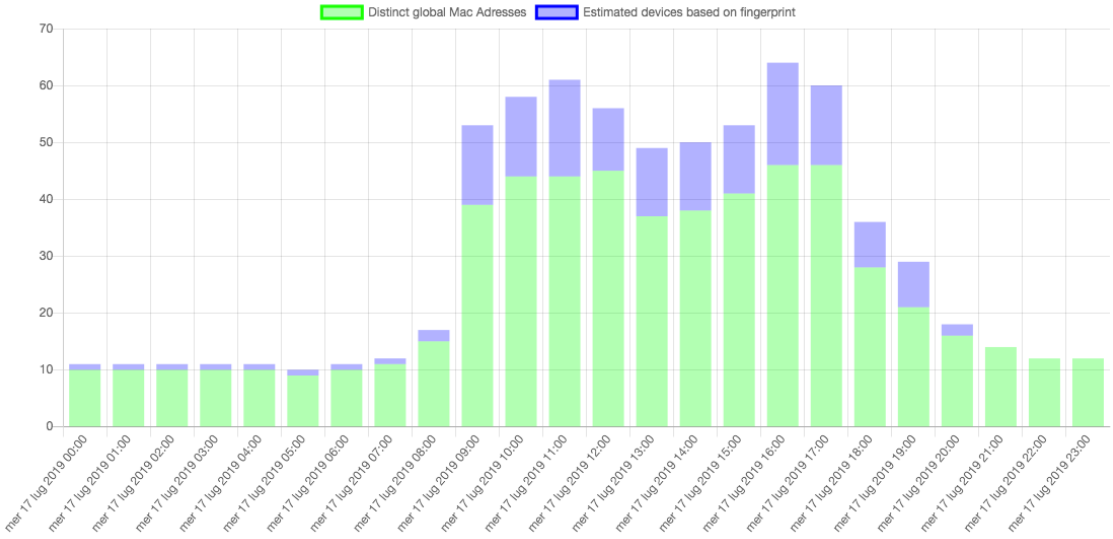


Figura 7.12: Stime del numero di dispositivi in intervalli di un'ora

7.12 sono riportati gli stessi conteggi della figura precedente ma aggregati in intervalli di un'ora. Da questi grafici emerge che i conteggi effettuati possono considerarsi coerenti con il flusso di persone all'interno dell'edificio in cui è stato svolto l'esperimento. L'orario in cui inizia ad incrementare il numero di dispositivi è coerente con l'inizio dell'orario di lavoro così come la dimunuzione durante la pausa pranzo e la diminuzione a partire dalle 18.
É possibile notare come un insieme di dispositivi viene rilevato anche di notte. Il numero di questi dispositivi è rimasto stabile all'interno di tutti i giorni osservati.
Correlare questi conteggi con il numero vero di persone presenti all'interno dell'edificio non è affatto facile. Bisogna tenere in considerazione che i dati sono stati raccolti in un ambiente in cui è molto probabile che una sola persona abbia a disposizione diversi dispositivi che di conseguenza porteranno a sovrastimare il numero di persone.
Inoltre, come sottolineato prima, il numero di dispositivi stimati con il fingerprint va considerato inferiore al vero numero di dispositivi.

7.7 Individuazione delle aree affollate tramite heatmap

Uno degli ultimi obiettivi di questa tesi è verificare la possibilità di usare questi dati per generare mappe che permettano l'individuazione delle aree più affollate di un edificio.
Tra i dati raccolti dagli sniffer, è presente l'RSSI (Received Signal Strength Indicator) che indica la potenza con cui il pacchetto è giunto allo sniffer che l'ha catturato. Questo valore è tanto più prossimo allo 0 più l'intensità del segnale è forte mentre decresce con la distanza, secondo la legge quadrata inversa [10].
In campo aperto, la relazione tra distanza e intensità del segnale è calcolabile nel seguente modo:

$$P(X) = P(X_i) - 10n \log(d)$$

dove:

- $P(X)$ è la perdita di intensità alla distanza d
- (X_i) è la perdita di segnale ad un punto di riferimento noto
- n è un parametro legato al decadimento del segnale
- d è la distanza a cui si trova il device rispetto allo sniffer

Risolvendo l'equazione per trovare d è possibile trovare la distanza in metri a cui si dovrebbe trovare il device dato il RSSI. Alcune prove effettuate hanno permesso di determinare che con $n = 2$ è possibile ottenere buone stime della distanza di un device in campo aperto. Il valore di (X_i) preso in considerazioni è stato -50 dBm, che rappresenta la perdita di segnale di un device posto a 1 metro dallo sniffer.
É necessario sottolineare che questo modello rappresenta una prima approssimazione e non tiene conto di fenomeni di riflessione, rifrazione e attenuazione del segnale, pertanto l'accuratezza del calcolo, anche in caso di campo aperto non può essere garantita.
La presenza di muri rende questo metodo di misura ancora più inefficace dato che, ad ogni muro attraversato, il segnale subisce un'attenuazione che porta a considerare il device più lontano di quanto sia in realtà.
L'approccio basato su heatmap cerca di superare questo limite basandosi sul fatto che dato un valore di RSSI, il device che ha emesso il pacchetto considerato si troverà in un'area circolare che avrà come centro lo sniffer e raggio la distanza calcolata in base al valore dell'RSSI. Sovrapponendo le circonferenze generate da altri pacchetti catturati nell'intervallo di tempo considerato potremmo individuare un insieme di aree di colorazione più intensa che dovrebbero corriposndere alle zone in cui sarà più probabile trovare un device. Questo approccio è stato applicato sia per l'individuazione di un singolo dispositivo che considerando tutti i dispositivi rilevati in un certo intervallo di tempo. É necessario sottolineare che per queste analisi sono stati usati soltanto i dati relativi a device che si annunciano con indirizzi MAC globally administered.

7.7.1 Heatmap applicata ad un singolo device

Considerando uno specifico device, sono stati considerati tutti i pacchetti rilevati nel corso di 24 ore, e aggregati in intervalli di un minuto. Per ogni minuto rilevante, cioè che presenta almeno un pacchetto raccolto, è stata applicata alla pianta dell'ambiente considerato la heatmap generata nel modo descritto in precedenza. Nel grafico 7.13 è riportato il tracciamento di un dispositivo preso come campione. Le figure 7.14 e 7.15 mostrano la heathmap risultante in due momenti in cui è evidente lo spostamento del device. In entrambi i casi è possibile osservare che nell'intervallo di un minuto considerato, 3 degli sniffer hanno rilevato almeno un pacchetto emesso dal device. Il fatto che in entrambi i casi il device venga rilevato vicino ad ogni sniffer, denotato dall'area di colore rosso intensa e con un raggio particolarmente ridotto è evidenza che il device stesse percorrendo il corridoio. Questo esempio risulta particolarmente significativo della potenzialità di questo approccio che, associando al grafico 7.13 queste ulteriori rappresentazioni, da modo di avere una visione più precisa sulla possibile posizione del device.

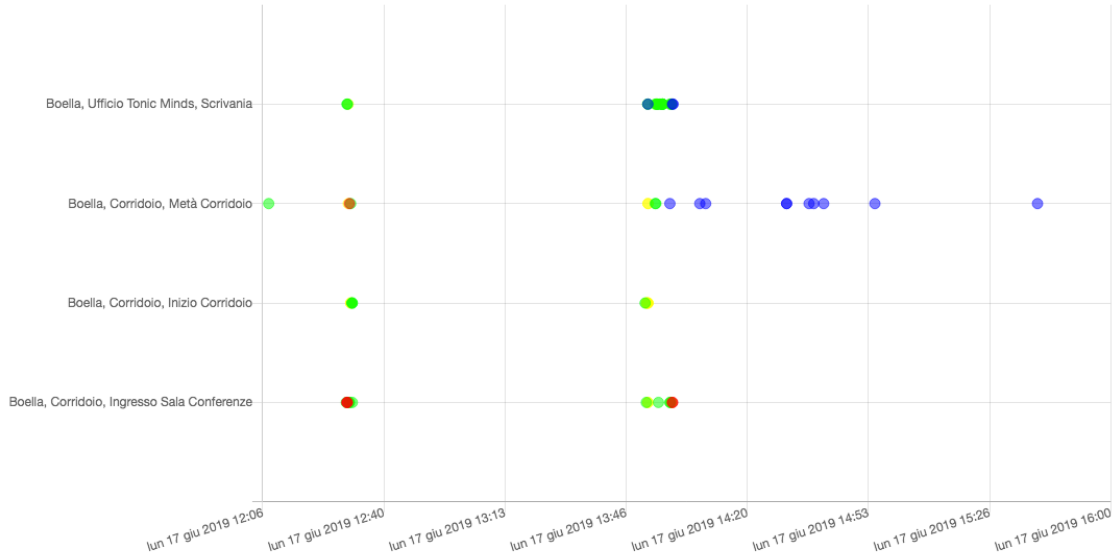


Figura 7.13: Tracciamento di un dispositivo

Figura 7.14: Heatmap

7.7.2 Heatmap applicata a dati aggregati

Applicando l'approccio descritto precedentemente a tutti i pacchetti raccolti in una certa finestra temporale da tutti gli sniffer è possibile, in linea teorica, evidenziare le zone con una concentrazione maggiore di device e quindi di persone. Per ogni indirizzo MAC (e quindi device) rilevato in un intervallo di tempo di un minuto e stata calcolata, per ogni sniffer, la media dell'RSSI dei pacchetti raccolti. Come risultato di tale aggregazione otterremo, per ogni minuto considerato, una lista che conterrà il riferimento dello sniffer e un valore di RSSI per ogni device. I dati così aggregati sono stati visualizzati tramite heatmap.
Rispetto al caso precendente, i risultati ottenuti sono difficilmente interpretabili.
In generale è possibile osservare una colorazione diffusa su tutta la zona osservata di bassa intensità con zone che presentano una tinta piu scura sporadicamente. Incrementando il tempo di aggregazione l'intensità del colore aumenta visto che aumentano il numero di device considerati ma il comportamento rimane comunque difficile da interpretare e non denota particolari pattern riconoscibili. É possibile notare una certa differenza tra gli orari in cui non sono presenti persone all'interno dei locali e quelli in cui sono presenti. Attualmente ritengo che sia necessario rivalutare la metodologia utilizzata. Per completezza vengono riportati alcuni esempi (figure 7.16, 7.17 e 7.18).

Figura 7.15: Heatmap

Figura 7.16: Heatmap corrispondente ad un'aggregazione di un minuto al mattino

Figura 7.17: Heatmap corrispondente ad un'aggregazione di 5 minuti al mattino

Figura 7.18: Heatmap corrispondente ad un'aggregazione di 5 minuti in tarda mattinata

Capitolo 8

Osservazioni finali e sviluppi futuri

I risultati ottenuti hanno dimostrato che il sistema progettato è in grado di effettuare la raccolta massiva di dati ma andrebbe messo alla prova in un contesto di utilizzo più realistico. La messa in opera all'interno dell'università permetterebbe sicuramente di verificare l'effettiva scalabilità dell'architettura. Sono ancora molte le funzionalità che possono essere sviluppate ma che necessitano di un insieme di dati più eterogenei per dare risultati apprezzabili. Attualmente lo studio dei tragitti svolti dalle persone non è stato approfondito in quanto difficilmente rilevabili nel contesto in cui sono stati raccolti i dati. Dislocando diversi sensori all'interno dei corridoi del Politecnico sarebbe possibile seguire gli spostamenti delle persone e usare questi dati in forma aggregata per ricavare informazioni sui tragitti piu seguiti e sulla provenieza delle persone rilevate in un certo intervallo di tempo all'interno di una zona. La realizzazione modulare dell'architettura e l'utilizzo del protocollo MQTT permettono di sfruttare quest'architettura come base di partenza per nuovi progetti basati sulla raccolta di dati.
Ricapitolando i risultati ottenuti per quanto riguarda il tracciamento ed il conteggio dei dispositivi, è stato evidenziato che attualmente l'utilizzo dei frame Probe Request non permette di deanonimizzare un dispositivo in modo certo e pertanto non può essere usato per il tracciamento. I risultati ottenuti per quello che riguarda il conteggio dei dispositivi sono più promettenti ma bisogna tenere in considerazione che le stime sul numero di device che utilizzano indirizzi locally administered non potranno essere validati se non in situazioni estremamente controllate, che non rappresentano un contesto di utilizzo realistico (con decine se non centinaia di device) e che comunque non è garantito che nell'intervallo di tempo considerato per effettuare i conteggi, tutti i device osservabili emettano effettivamente almeno un frame probe request. Per questo i risultati dei conteggi ottenuti possono essere considerati più validi in termini relativi piuttosto che in termini assoluti. Tuttavia i risultati dei conteggi denotano una forte correlazione tra l'andamento rilevato e l'effettiva presenza di persone nei locali.
Infine, è evidente che l'approccio basato su fingerprint individuato sarà sempre meno utilizzabile man mano che saranno di più i dispositivi che implementeranno sistemi di randomizzazione dei MAC address all'interno dei Probe Request.

Bibliografia

[1] A. Banks, R. Gupta *MQTT Version 3.1.1.*, OASIS Standard, October 2014, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, Ultima versione: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[2] M. S. Ghast, *802.11 Wireless Networks: The Definitive Guide*, O'Reilly Media Inc., 2005.

[3] J. Carnell, *Spring Microservices in Action*, Manning Publications, 2017.

[4] J. Porter, *London Underground to start tracking all phones using Wi-Fi starting in July*, The Verge, July 2019, https://www.theverge.com/2019/5/22/18635584/london-underground-tube-tfl-wi-fi-tracking-privacy-data-security-transport

[5] J. O Malley, *Here's What TfL Learned From Tracking Your Phone On the Tube* , Gizmodo, February 2017, https://www.gizmodo.co.uk/2017/02/heres-what-tfl-learned-from-tracking-your-phone-on-the-tube/

[6] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000.

[7] D. Hardt, Ed., *The OAuth 2.0 Authorization Framework*, RFC 6749, Internet Engineering Task Force, October 2012, https://tools.ietf.org/html/rfc6749

[8] M. Jones, J. Bradley, N. Sakimura, *JSON Web Token (JWT)*, RFC 7519, Internet Engineering Task Force, May 2015, https://tools.ietf.org/html/rfc7519

[9] Eclipse Project, Moquette Broker, https://projects.eclipse.org/projects/iot.moquette

[10] OnkarPathak P. P., Palkar R., Tawari, *Wi-Fi indoor positioning system based on RSSI measurements from Wi-Fi access points–a trilateration approach.* Int. J. Sci. Eng. Res, April 2014

[11] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, F. Plessens *Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms* Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. ACM, 2016

[12] J. Freudiger *How talkative is your mobile device? An experimental study of Wi-Fi probe requests*, WiSec, 2015

[13] C. Matte *Wi-Fi Tracking: Fingerprinting Attacks and Counter-Measures*, Université de Lyon, 2017

[14] R. Lim, M. Zimmerling, L. Thiele *Passive, privacy-preserving real-time counting of unmodified smartphones via zigbee interference*, Distributed Computing in Sensor Systems (DCOSS), 2015

[15] J. Martin, T. Mayberry, C. Donahue, L. Foppe, L. Brown, C. Riggins, E. C. Rye, D. Brown, *A Study of MAC Address Randomization in Mobile deices and When it Fails*, Proceedings on Privacy Enhancing Technologies, 2017
[16] D. Gentry, A. Pennarum, *Passive Taxonomy of Wifi Clients using MLME Frame Contents*, arXiv preprint arXiv:1608.01725, 2016
[17] https://docs.docker.com
[18] https://angular.io