Coppie chiave-valore distribuite con Wi-Fi Direct

Giulio Zaccaroni
giulio.zaccaroni@studio.unibo.it

Dicembre 2021

Si vuole sviluppare un modulo che permetta di replicare, all'interno di gruppi di dispositivi Android, delle coppie chiave-valore attraverso Wi-Fi Direct. I gruppi sono caratterizzati da un *master* (il creatore del gruppo) e più *slave*, questi ultimi possono entrare e uscire in qualsiasi momento dal gruppo. Oltre al modulo si vuole anche un applicativo dimostrativo che lo utilizzi, questo deve disporre di due schermate:

- *Lista Gruppi* che permette:
  - la creazione di un nuovo gruppo;
  - l'accesso ad un gruppo già esistente;
- *Visualizza Gruppo* che permette:
  - la visualizzazione delle coppie chiave-valore;
  - l'uscita dal gruppo per gli *slave*;
  - l'eliminazione del gruppo per il *master*.

Il modulo per la creazione/gestione dei gruppi deve essere creato in modo da permetterne il riutilizzo per progetti futuri, deve quindi poter essere utilizzabile anche in progetti che non prevedano l'uso di coppie chiave-valore (esempio: database SQLite).

1 Obiettivi e Requisiti

1.1 Requisiti funzionali

I requisiti funzionali, in ordine decrescente di importanza, sono:

1. ogni dispositivo deve avere in memoria i dati condivisi dal gruppo;
2. per modificare i dati deve essere inviata una richiesta al master;
3. le richieste vengono accettate dal *master* e poi propagate
4. un dispositivo deve poter entrare in un gruppo;
5. un dispositivo deve poter uscire da un gruppo;
6. deve essere possibile vedere l'elenco dei gruppi presenti nelle vicinanze.

1.2 Requisiti non funzionali

I requisiti non funzionali, in ordine decrescente di importanza, sono:

1. l'applicativo deve permettere a dispositivi con sistema Android di interagire tra loro a prescindere dal produttore di questi;
2. l'applicativo deve supportare il maggior numero di versioni Android (almeno fino a Android Lollipop);
3. non deve essere possibile avere due *master* nello stesso momento;
4. il sistema deve continuare a funzionare anche se alcuni dispositivi si disconnettono (tolleranza di partizione);
5. i membri devono poter sempre accedere ai dati (disponibilità);
6. tutti i membri devono vedere gli stessi dati (coerenza);
7. il sistema non deve essere legato a un modello particolare di persistenza dei dati;
8. l'interfaccia deve poter essere semplice da usare anche per personale non esperto.

Nota sui requisiti 1 e 2: questi vengono imposti dalla disponibilità di dispositivi per il progetto, dato che dispongo unicamente di un dispositivo Nexus con Android 5 (API Level 21) e uno Redmi con Android 11 (API Level 30).

1.3 Scenari

La libreria permette la replicazione di dati tramite Wi-Fi Direct in maniera semplice. Perché questo avvenga è necessario che il *master* crei un gruppo e che gli *slave* si colleghino ad esso. All'interno del gruppo sia il *master* sia gli *slave* possono modificare i dati: nel caso dell'applicativo DEMO è possibile aggiungere o modificare coppie chiave-valore. Se uno *slave* si disconnette o esce dall'area del *master* questi potrà solo accedere ai dati in suo possesso ma non potrà più o effettuare modifiche. I soggetti che interagiscono con la libreria sono *master* (1) e *slave* (0-n). Di seguito due scenari esemplificativi di utilizzo della libreria:

- Lorenzo e Sofia vogliono fare un'app che permetta di giocare a scacchi senza la necessità di una connessione a internet, hanno bisogno di una libreria che gli permetta di sapere di chi è il turno e dove sono posizionati i pezzi.
- Sofia e dei suoi amici vogliono fare un gioco di quiz in cui il *master* fa delle domande e gli altri devono rispondere, ogni giocatore ha un punteggio che viene incrementato in base alla correttezza della risposta e al tempo impiegato a darla. Hanno bisogno di una libreria che permetta loro di tenere traccia dell'ordine di risposta e della domanda attuale.

1.4 Politica di auto-valutazione

In Android vi sono due categorie di test:

- *Local Unit Test*: vengono eseguiti localmente nell'ambiente di sviluppo, non hanno accesso alle chiamate di API di Android a meno di utilizzo di particolari librerie (Roboelectric [14]);
- *Instrumented Unit Test* [2]: vengono eseguiti su un dispositivo reale o un emulatore e permettono di testare tutte le funzioni dell'applicazione, con questo tipo di test è possibile anche verificare il funzionamento di activity e fragment.

A meno che l'ambiente di esecuzione dei test non sia collegato a un dispositivo reale (generalmente non possibile in un ambiente di testing remoto) purtroppo nemmeno l'*Instrumented Unit Test* permette di verificare il corretto funzionamento di Wi-Fi Direct. Per questo motivo sarà importante nella parte di progettazione andare a disaccoppiare la parte che dipende da Wi-Fi Direct dal resto per poter testare la porzione più grande possibile di codice in modo automatico.

Quindi si sceglie di inserire dei test automatici per le seguenti aree di funzionamento:

- replicazione dei dati tra dispositivi;
- disponibilità dell'informazione anche se una parte dei dispositivi che compongono il gruppo non è raggiungibile;
- disponibilità dell'informazione anche se il dispositivo non riesce a connettersi al master;

- capacità degli *slave* di riconnettersi al *master* senza perdita di informazione se questo non è temporaneamente disponibile;
- capacità di alcuni *slave* di connettersi al gruppo in un secondo momento;
- disponibilità dell'informazione anche se una parte dei dispositivi che compongono il gruppo decide di non farne più parte.

Non vengono inseriti test per l'interfaccia grafica non essendo questa elemento centrale del progetto.

2 Analisi dei requisiti

Teorema di CAP La libreria alla base di questo applicativo dovrà permettere la replicazione dei dati tra più dispositivi in particolare cercando di rispettare i requisiti (4,5,6). Non è possibile rispettare tutti e tre i vincoli assieme per il teorema di CAP quindi si decide, rispettandone la priorità data, di privilegiare il funzionamento del sistema in caso di disconnessione dei dispositivi e la possibilità di accedere sempre ai dati a discapito della coerenza di questi. Inoltre si decide anche di ignorare le *Byzantine Failures* [16], non essendo richiesta capacità di resistenza contro di esse nei requisti.

Wi-Fi Direct e Android Effettuando verifiche sulle possibilità offerte dalla implementazione Android, sono emerse le seguenti considerazioni:

- l'altro dispositivo deve accettare la richiesta di connessione per permettere al primo di connettersi [15];
- non funzionano in maniera coerente tra dispositivi di OEM (Original Equipment Manufacturer) differenti [13, 5];
- è assente una documentazione completa: le API presentano comportamenti non documentati [13, 5];
- il funzionamento è instabile e non ripetibile: il tempo di ricerca di dispositivi vicini può cambiare in maniera importante tra un avvio e l'altro [13, 5];
- se utilizzato per un lungo periodo di tempo le API tendono a bloccarsi [13, 5].

Si presume che alcuni di questi problemi derivino dall'entrata in funzione di sistemi di risparmio energetico del dispositivo, dato che la ricerca di dispositivi nelle vicinanze è molto dispendiosa.

Serializzazione e deserializzazione dei dati Un ultimo aspetto da considerare per la comunicazione è la necessità di serializzare/deserializzare i dati, in merito a questo, però, non vi sono particolari requisiti.

Chiamate asincrone Il progetto non permette di utilizzare una versione recente di Java per i vincoli sulla *Framework API Level* 21 di Android che impongono l'utilizzo di Java 8. Questo requisito non è trascurabile: in Java 8 sono assenti gran parte delle funzioni dei linguaggi moderni che permettono di evitare la scrittura di codice *boilerplate* e rendono più comprensibile il codice. Inoltre, è importante ricordare che su Android non è possibile effettuare chiamate IO sul *thread* che gestisce la UI perché questo la renderebbe *non-responsive* dando all'utente l'idea che l'app sia bloccata.

3 Background

3.1 Wi-Fi Direct

> Disclaimer: le informazioni di seguito fornite sono estratte da Wi-Fi Direct® Specification (v1.9) [1] e possono cambiare in future versioni della specifica.

Wi-Fi Direct®, o precedentemente Wi-Fi Peer 2 Peer, è uno standard Wi-Fi per la creazione e gestione di reti LAN funzionalmente equivalenti a una tradizionale rete Wi-Fi senza la presenza di un intermediario (i.e. access point, router). La connessione può avvenire con qualsiasi dispositivo (indipendentemente dal produttore) a patto che almeno uno implementi il protocollo Wi-Fi Direct, viene usato il concetto di servizio per permettere ai dispositivi di segnalare che funzioni supportano (ad esempio: stampa di documenti).

Poiché non richiede alcun Access Point, il dispositivo stesso, detto P2P Device, ha la capacità di funzionare come tale.I dispositivi che si connettono formano un P2P Group: tra loro, quello che implementa la funzionalità di Access Point è indicato come P2P Group Owner, e gli altri che fungono da client sono noti come P2P Client. La rete ha una topologia 1:n dove i client sono connessi al Group Owner. Quest'ultimo è l'unico autorizzato a collegare i dispositivi del suo gruppo P2P a una rete esterna. Wi-Fi Direct non permette di trasferire il ruolo di Group Owner all'interno del gruppo. Se il Group Owner lascia il gruppo P2P, allora il gruppo viene sciolto e deve essere ristabilito.

Il proprietario del gruppo P2P agisce come un server DHCP per fornire indirizzi IP ai client P2P connessi che utilizzano IP, questi deve come minimo supportare il protocollo Internet versione 4 (IPv4) e l'assegnazione di indirizzo IP e subnet mask.

Ogni dispositivo è identificato da un indirizzo detto P2P Device Address, questo è il suo Mac Address *globally administered* o il suo MAC Address *globally administered* con il *locally administered* bit impostato.

Ogni gruppo è identificato da un P2P Group ID e può essere temporaneo, se una volta sciolto smette di esistere, o persistente, nel caso in cui si salvino le credenziali del gruppo per riformare il gruppo dopo la formazione nel caso venisse sciolto (in questo caso il P2P Group ID resta sempre lo stesso).

Funzioni di un P2P Device Un P2P Device, in particolare, deve supportare le seguenti funzioni specifiche P2P:

- P2P Discovery fornisce una serie di funzioni che permettono a un dispositivo di identificare e connettersi facilmente e rapidamente a un altro dispositivo P2P e ai suoi servizi nelle vicinanze;
- P2P Group Operation fornisce funzioni per il funzionamento operativo di un gruppo P2P;
- P2P Power Management fornisce un insieme di funzioni per ridurre il consumo energetico dei dispositivi P2P che operano al di fuori del DMG (Directional Multi-Gigabit)

P2P Discovery La funzione di P2P Discovery consiste dei seguenti componenti principali:

- Device Discovery: un dispositivo prima trasmette un messaggio broadcast di richiesta chiedendo il MAC ID a tutti i dispositivi vicini (questa fase è simile alla fase di scansione nel normale Wi-Fi). Tutti i dispositivi che sentono questo messaggio rispondono con un messaggio unicast al mittente. I dispositivi alternano periodi di invio e ascolto delle richieste di sonda, e successivamente delle loro risposte;
- Service Discovery: il mittente invia un messaggio unicast di richiesta di servizio ad ogni dispositivo. I dispositivi riceventi rispondono con un messaggio unicast con i dettagli del servizio;
- Group Formation: permette di creare un nuovo gruppo e di determinare chi sarà il Group Owner;
- P2P Invitation: permette di ricomporre un gruppo persistente o invitare un dispositivo in un gruppo esistente.

Il processo di Service Discovery, in particolare, permette ai dispositivi Wi-Fi Direct di scoprire l'un l'altro e i servizi che supportano. Per esempio, un dispositivo Wi-Fi Direct potrebbe cercare tutti i dispositivi compatibili nella zona e poi restringere l'elenco ai soli dispositivi che consentono la stampa prima di visualizzare un elenco di stampanti Wi-Fi Direct abilitate nelle vicinanze. Prima della creazione di un gruppo P2P, i dispositivi P2P possono scambiarsi query per scoprire l'insieme dei servizi disponibili e, in base a questo, decidere se continuare o meno la formazione del gruppo.

Per realizzare questa funzione viene utilizzato il Generic Advertisement Service (GAS): un protocollo di query e risposta di livello 2 implementato attraverso l'uso di action frame pubblici, che permette a due dispositivi 802.11 non associati di scambiarsi query appartenenti a un protocollo di livello superiore (per esempio un protocollo di scoperta dei servizi).

Formazione dei gruppi È di interesse per lo svolgimento del progetto anche il componente che permette la formazione dei gruppi. Per scegliere quale dispositivo sarà Group Owner viene usato un parametro detto Group Owner Intent, se il dispositivo deve essere

Group Owner viene impostato a 15. Il dispositivo con il valore più alto del parametro diventa il Group Owner, per evitare che i due dispositivi abbiano lo stesso valore viene usato un bit speciale detto Tie Breaker assegnato casualmente.

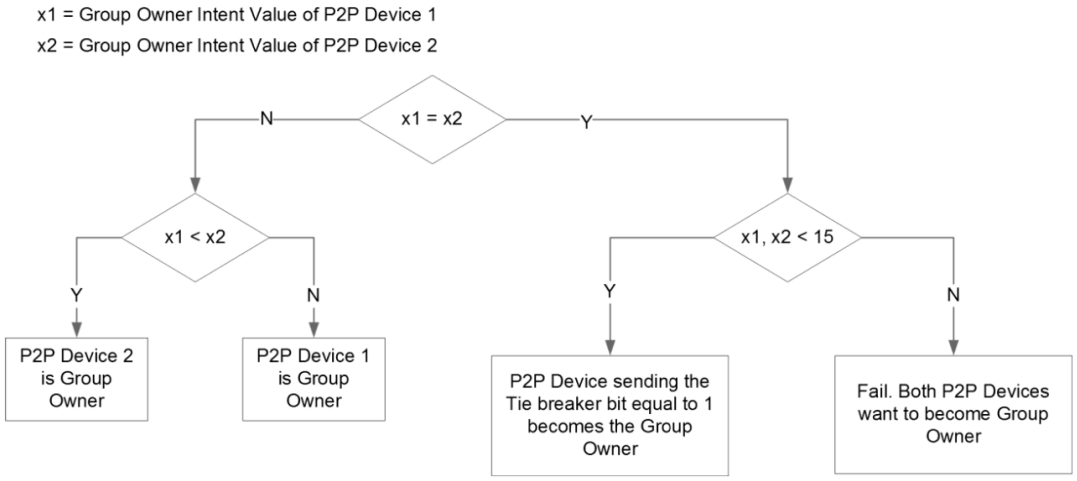

Figure 1: Processo di scelta del P2P Group Owner

Supporto Ad oggi gran parte dei dispositivi supporta Wi-Fi Direct, l'unica grande eccezione sono i dispositivi Apple che adottano invece una tecnologia proprietaria chiamata Apple Multipeer Connectivity.

4 Design

4.1 Struttura

Per progettare dei componenti riutilizzabili è fondamentale avere una struttura che lo permetta; in questo caso, secondo me, è importante distinguere tra:

- una componente riutilizzabile indipendente dalle informazioni relative alla persistenza (Modulo Widi), il cui compito è quello di permettere il dialogo e l'interazione con Wi-Fi Direct;
- un applicativo dimostrativo che si occupa dell'interazione tra libreria Widi e utente finale nel caso specifico delle coppie chiave-valore.

Modulo Widi Il Modulo Widi è composto da due interfacce di interazione con l'utilizzatore del modulo e altre due che ne gestiscono la persistenza. L'utilizzo di interfacce permette un *testing* più agile tramite *mock* e maggiore facilità nella sostituzione dei componenti. Per permettere agli *slave* che entrano in un secondo momento di raggiungere il medesimo

stato del *master* senza la necessità di scaricare tutti i dati, è stato inserito il concetto di changelog, vale a dire un registro di tutte le modifiche che vengono richieste.

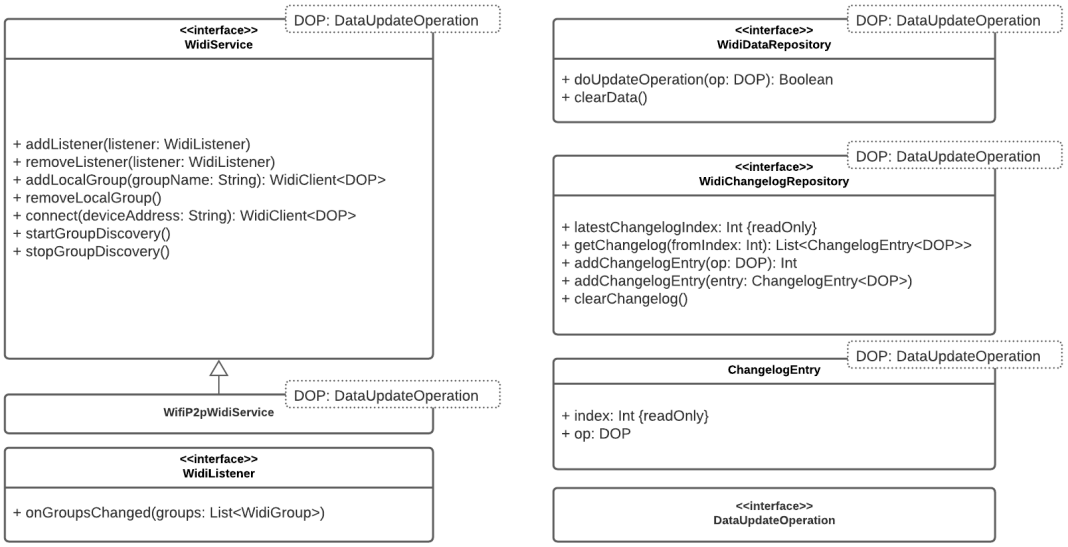

Figure 2: Diagramma delle classi di `WidiService`

`WidiService` permette:

- la creazione di un gruppo con `addLocalGroup` a cui è necessario passare il nome del gruppo, il dispositivo che chiama questo metodo diventa *master del gruppo*;
- la rimozione di un gruppo con `removeLocalGroup`;
- la ricerca di gruppi vicini attraverso `startGroupDiscovery` e `stopGroupDiscovery`: per essere notificati della presenza di gruppi prima di avviare la *discovery* è necessario registrare un listener con `addListener`;
- la connessione con un gruppo attraverso `connect` che riceve come parametro l'indirizzo MAC del *master* del gruppo;

All'interno della libreria è disponibile un'implementazione di questa interfaccia specifica per Wi-Fi Direct.

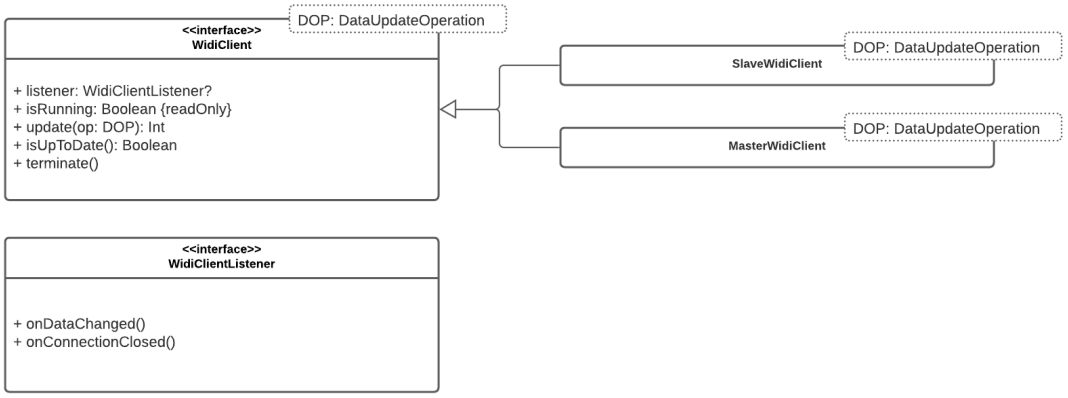

Figure 3: Diagramma delle classi di `WidiClient`

`WidiClient`: permette a *master* e *slave* di interagire tra loro. Sono disponibili due implementazioni di questa interfaccia: una per il *master* e una per gli *slave*.

Le interfacce di gestione della persistenza, invece, richiedono più lavoro da parte dell'utilizzatore della libreria, in quanto è necessario creare la propria implementazione. Queste ultime sono:

- `WidiDataRepository`: esegue le operazioni richieste dal *master* o dagli *slave*;
- `WidiChangelogRepository`: si occupa di mantenere tutti i log e ne permette il recupero con la possibilità di filtrare per indice, così da diminuire il carico sulla memoria.

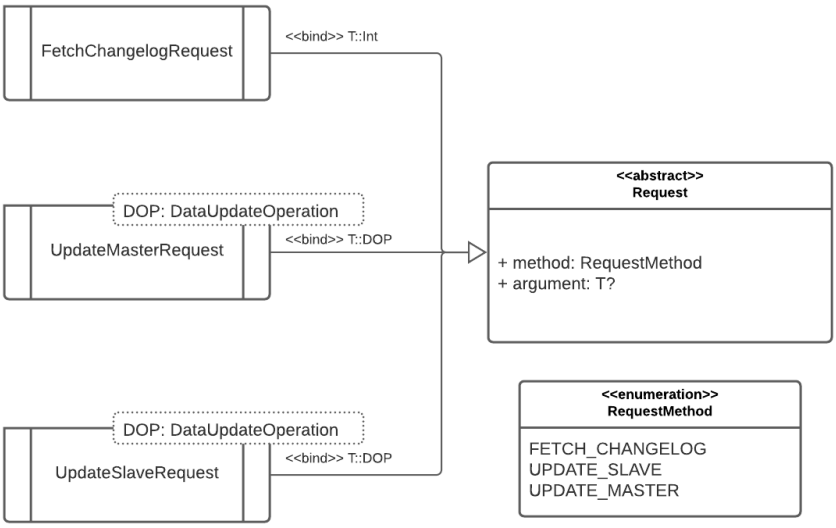

Figure 4: Diagramma delle classi di Request

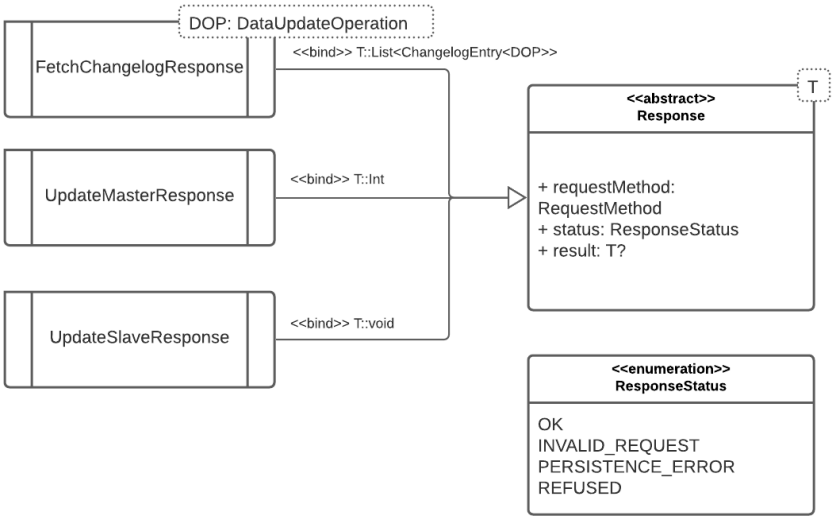

Figure 5: Diagramma delle classi di Response

Infine, sono presenti `Request` e `Response`, due classi astratte il cui compito è trasportare informazioni sulle richieste e risposte inviate. È necessario che queste classi supportino la serializzazione e la deserializzazione per consentirne l'invio tra dispositivi.

4.2 Comportamento

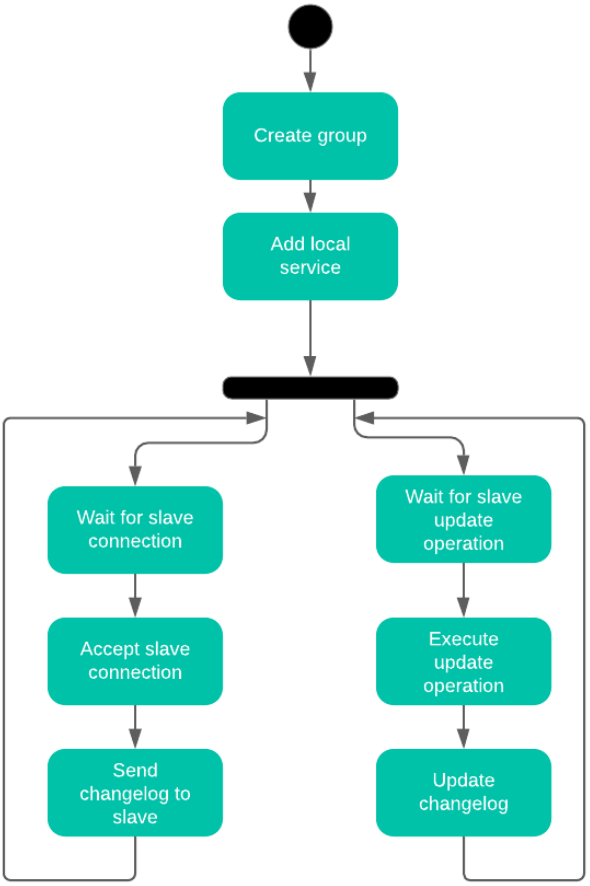

Figure 6: Diagramma di attività, creazione di un gruppo

Nella figura sopra è possibile vedere la creazione di un gruppo, l'utente che lo crea è detto *master* mentre quelli che vi accedono sono detti *slave*. Nonostante per brevità venga omesso ovviamente le fasi di ricerca degli *slave* e esecuzione delle operazioni di aggiornamento vengono eseguite fino alla chiusura del gruppo da parte del *master*.

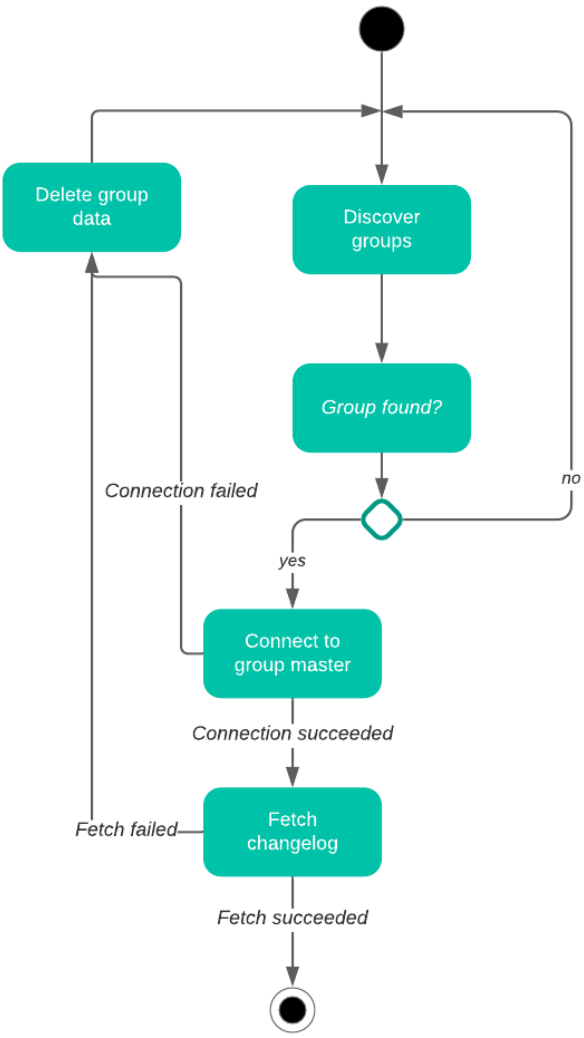

Figure 7: Diagramma di attività, accesso a un gruppo

Nel diagramma sopra è possibile vedere le varie fasi dell'entrata in un gruppo, per brevità non viene mostrata la fase di controllo sulla connessione e sulla presenza di nuovi dati mostrata invece nella figura 8. Tutti gli utenti che accedono a un gruppo sono detti *slave*, mentre il creatore del gruppo è detto (*master*).

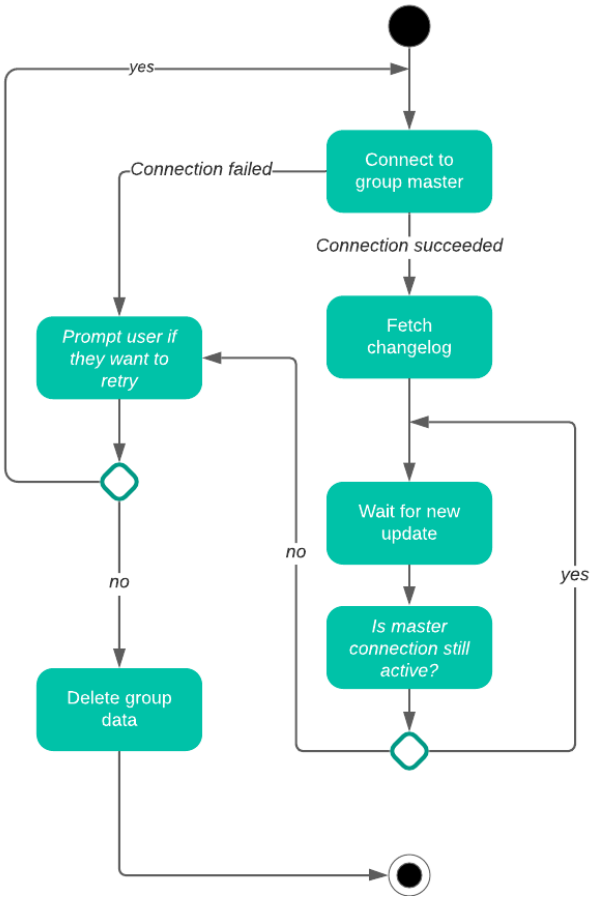

Figure 8: Diagramma di attività, riconnessione al gruppo

Nell'immagine sopra è possibile vedere le varie fasi della riconnessione a un gruppo la quale avviene ogni volta che lo slave decide (volontariamente o involontariamente) di disconnettersi, nell'immagine è anche mostrato cosa succede se è il master a scollegarsi temporaneamente.

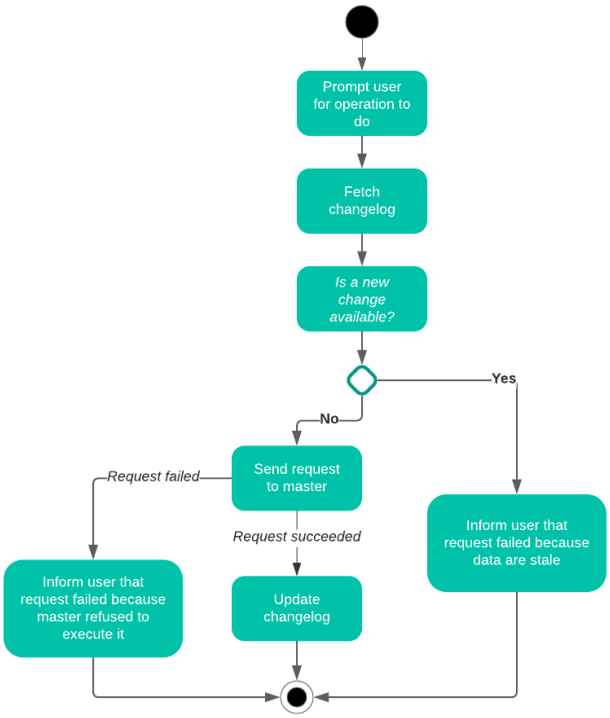

Figure 9: Diagramma di attività, operazione di update

Infine, sopra è possibile vedere i passaggi per l'aggiornamento del master da parte di uno *slave*. Si noti il fatto che, se per qualche motivo al momento della richiesta il *changelog* dello *slave* non è aggiornato, la richiesta viene rifiutata (in modo da assicurare che l'operazione venga eseguita solo se lo *slave* ha l'ultima versione del *changelog*). Questa funzione è stata inserita per evitare, ad esempio nel casi d'uso degli scacchi mostrato sopra, ad un giocatore di fare una mossa se non ha la scacchiera aggiornata. Sono al corrente che questa, nel caso in cui molti *slave* agiscano in contemporanea in casi d'uso diversi può essere non voluta e problematica, per questo può essere facilmente rimossa.

4.3 Interazione

L'interazione tra *master* e *slave* avviene sempre tramite lo scambio, in coppia, di richieste (`Request`) e risposte (`Response`).

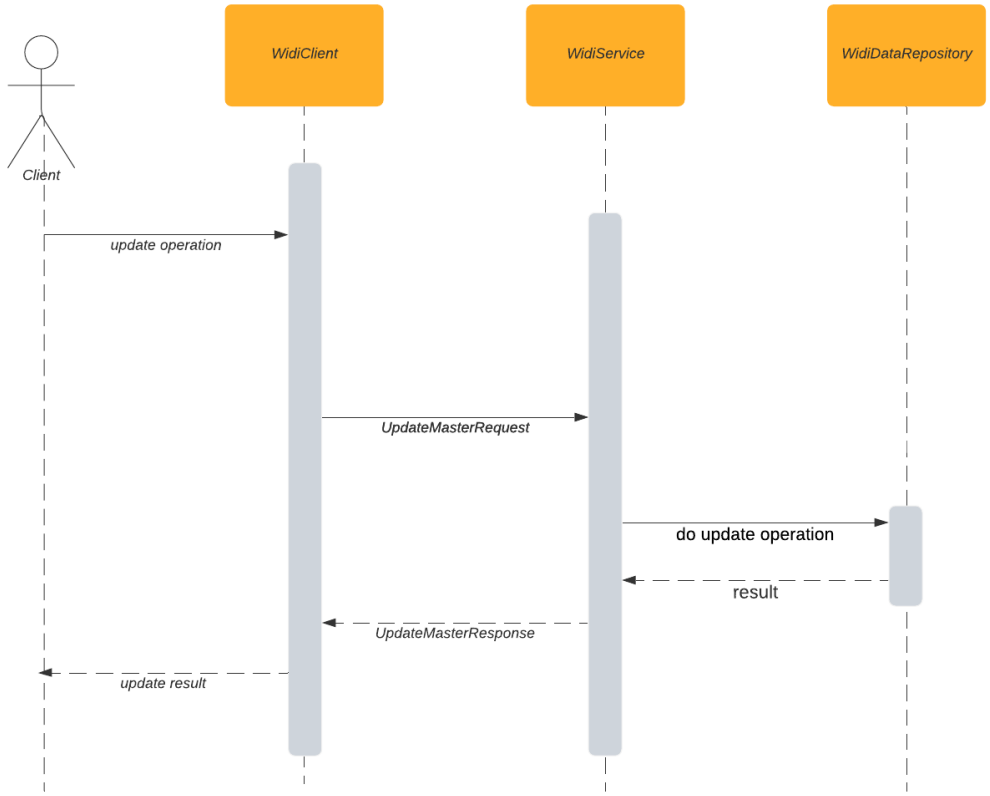

Figure 10: Diagramma di sequenza, richiesta di aggiornamento da parte del client

5 Dettagli implementativi

5.1 WifiP2pManager

In Android è possibile accedere all'API che permette l'utilizzo di Wi-Fi Direct attraverso la classe `WifiP2pManager`, la quale si occupa di molti compiti, tra i quali:

- creazione e distruzione di gruppi: è una funzione legacy che permette di creare un Access Point in modo da permettere anche a dispositivi che non supportano Wi-Fi Direct di connettersi, non viene usata in questo progetto;
- gestione di servizi (discovery e creazione/eliminazione): permette di segnalare ai dispositivi nelle vicinanze la disponibilità di un servizio;
- connessione con dispositivi specifici: permette di collegarsi direttamente a un dispositivo se si conosce il suo indirizzo.

Nella implementazione il *master* si occupa di creare un servizio tramite `WifiP2pManager .addLocalService` e, una volta individuato il servizio con `WifiP2pManager.discoverServices` del *master*, gli slave si collegano ad esso tramite `WifiP2pManager.connect`. Dato che il master è l'unico componente del gruppo senza il quale questo non funziona (perché banalmente non è possibile accettare le richieste di aggiornamento) viene scelto il *master* come *P2P Group Owner*, per farlo viene impostato il parametro `Group Owner Intent` a 0.

Nello sviluppo delle chiamate al modulo WifiP2pManager [4] di Android, che si occupa dell'interazione tramite Wi-Fi Direct, sono stati riscontrati diversi problemi che hanno portato a scelte poco ortodosse nell'utilizzo dell'API. Tra questi:

- `WifiP2pManager.discoverServices`: cerca servizi nella rete locale. Dopo 120 secondi di funzionamento, la funzione si blocca senza mandare avvisi (si presume per risparmiare batteria), quindi è necessario rifare l'invocazione poco prima del blocco per permettere un funzionamento costante;
- `WifiP2pManager.requestConnectionInfo`: richiede le informazioni della connessione attiva, se esiste. Se non viene registrato un `BroadcastReceiver` (anche senza una implementazione) restituisce sempre null;
- `WifiP2pManager.addLocalService`: si occupa di aggiungere un servizio alla rete locale. Se non viene invocato prima `WifiP2pManager.clearLocalServices` funziona in maniera non affidabile. Inoltre, se non viene invocata anche `WifiP2pManager. discoverPeers` a intervalli costanti (10 secondi), i dispositivi che cercano il servizio dopo la chiamata della `WifiP2pManager.addLocalService` potrebbero non trovarlo mai.

5.2 Chiamate asincrone

In Java 8 non sono disponibili ancora gran parte delle funzioni recenti, come i `CompletableFutures` [11]. Per ovviare a questo problema, si decide di usare le Kotlin Coroutines [8] per

gestire le chiamate asincrone, le quali permettono l'invocazione di funzioni senza bloccare il *thread* da cui vengono chiamate e il lancio di eccezioni. Inoltre, consentono una scrittura più comprensibile del codice, poiché è più simile a quello non asincrono.

Esempio:

```
manager.connect(channel, config)
val connectionInfo = manager.requestConnectionInfo(channel)
```

Queste due semplici linee di codice, senza l'utilizzo delle Kotlin Coroutines, diventano:

```
manager.connect(
    channel,
    config,
    object : WifiP2pManager.ActionListener {
        override fun onSuccess() {
            manager.requestConnectionInfo(
                channel,
                object : WifiP2pManager.
                    ConnectionInfoListener {
                    override fun onConnectionInfoAvailable(
                        p0: WifiP2pInfo?
                    ) {
                        // Rest of code
                    }
                }
            );
        }
        override fun onFailure(p0: Int) {}
    }
);
```

5.3 Serializzazione e deserializzazione dei dati

Per lo scambio dei dati sono stati valutati:

- *Json*: garantisce leggibilità dei dati scambiati in fase di *debugging*, è conosciuto da gran parte degli utenti;
- *BSON* [10]: permette di scambiare dati in maniera molto efficiente sia dal punto di vista delle performance che dal punto di vista della dimensione dei dati trasmessi, ma non è leggibile dagli utenti;
- *gRPC* [9]: permette l'invio e la ricezione di dati tramite HTTP/2 su una singola connessione TCP. Per quanto interessante per la sua capacità di gestire sia lo scambio di informazioni che la serializzazione in maniera affidabile, rischia di aggiungere molta complessità al progetto. Inoltre, attualmente non è supportato l'avvio di un server gRPC su Android [6].

È stato scelto di usare Json per lo scambio dei dati, perché al momento questo progetto è da intendersi come *proof-of-concept* e non è un sistema dove i tempi di scambio sono critici.

Per serializzare i dati si è scelto di usare Gson, ma, non trattandosi di una libreria nativa Kotlin, sono stati riscontrati problemi in fase di sviluppo. I principali sono:

- i tipi Kotlin come `List<ChangelogEntry<DataUpdateOp>>`
  che vengono convertiti in Java come: `List<out ChangelogEntry<DataUpdateOp>>`
  non permettono a Gson di vedere il tipo `ChangelogEntry`. Per evitarlo, è necessario aggiungere l'annotazione `JvmSuppressWildcards` [7]

```
List<@JvmSuppressWildcards ChangelogEntry<DataUpdateOp>>
```

- in Kotlin al posto di Void viene usato Unit, un singleton. Gson, anziché usare l'istanza del singleton, ne crea una nuova rompendo l'equals. Per risolvere questo problema è stato aggiunto un deserializzatore per Unit.

5.4 Esecuzione dei test

Per velocizzare l'esecuzione dei test, ho deciso di usare Roboelectric [14], che ne permette l'esecuzione senza necessità di un collegamento con l'emulatore di Android. Ovviamente non tutte le API funzionano e, in certi casi, ho notato comportamenti diversi tra le chiamate effettuate sul dispositivo reale e quelle effettuate su Roboelectric. Tuttavia, la differenza è minima rispetto al vantaggio in termini di tempi di esecuzione. È stato usato JUnit 4 e non 5 perché è la versione più recente ufficialmente supportata su Android. Sono stati implementati test per le seguenti funzioni:

- serializzazione di richieste e risposte di *master* e *slave*;
- persistenza dei dati;
- modifica dei dati da parte del *master* e aggiornamento degli *slave*;
- modifica dei dati da parte degli *slave* e aggiornamento del *master*;
- chiusura effettiva della connessione e controllo che dopo la disconnessione uno slave non riceva più dati;
- capacità di alcuni *slave* di connettersi al gruppo in un secondo momento;
- disponibilità dell'informazione anche se una parte dei dispositivi che compongono il gruppo decide di non farne più parte.
- corretto funzionamento del *master* anche nel caso in cui non ci siano *slave* collegati.
- corretto recupero dei dati persi nel caso in cui lo *slave* si disconnetta temporaneamente.

È possibile trovare la documentazione del modulo (prodotta con Dokka) qui.

6 Auto-valutazione

Alla fine del lavoro tutti i test automatici hanno avuto esito positivo, nonostante il codice non risulti interamente coperto da essi. I test effettuati direttamente sui dispositivi reali, invece, hanno presentato le seguenti criticità:

- a volte il servizio di ricerca dei gruppi può impiegare diversi minuti a trovare un gruppo;
- non è stato possibile fare prove con più di due dispositivi.

In generale il progetto può considerarsi svolto con successo poiché i test effettuati funzionano.

7 Istruzioni per il deployment

Se si vuole generare un nuovo manufatto software vi sono due metodi.Il primo è più semplice e prevede unicamente il push del codice aggiornato verso la repository Gitlab, in questo modo la libreria sarà prodotta automaticamente. In alternativa, è possibile seguire i seguenti passaggi sul proprio computer personale:

1. installare Android Studio versione 2020.3.1;
2. aprire il progetto con Android Studio;
3. a questo punto, vi sono due opzioni:
   - per installare il progetto su un dispositivo collegato selezionare il dispositivo dal menù a tendina in alto e premere "Run";
   - per generare un pacchetto apk:
     a) aprire il menù a tendina "Build";
     b) selezionare "Build Bundle(s) / APK(s);
     c) selezionare "Build APK(s)".

È possibile reperire informazioni sull'utilizzo del modulo qui.

8 Esempi di utilizzo

Figure 11: Creazione del gruppo

Figure 12: Visualizzazione del gruppo

Creazione di un gruppo In queste viste è possibile vedere i passaggi per la creazione di un gruppo: è sufficiente premere "Create Group": ad ogni gruppo viene assegnato automaticamente un nome unico (con ragionevole probabilità) casuale (UUID v4).

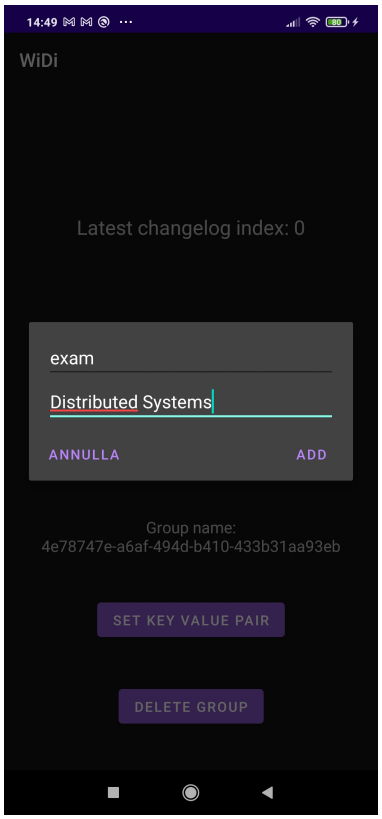

Figure 13: Aggiunta coppia chiave-valore

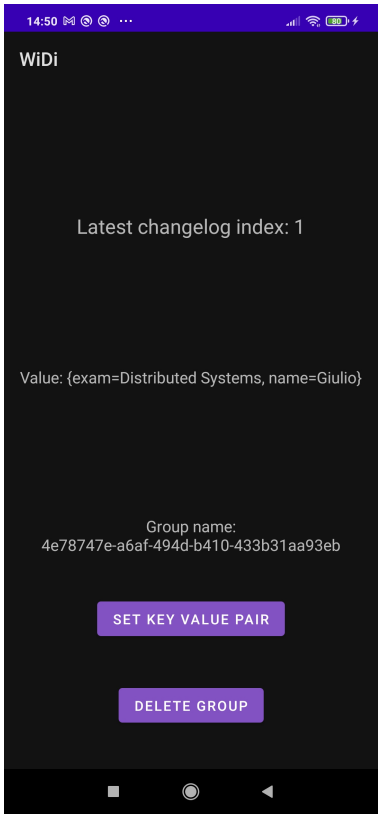

Figure 14: Gruppo con coppie chiave-valore

Impostazione di una coppia chiave-valore In queste viste è possibile vedere la modalità di assegnazione di un valore a una chiave: è necessario premere "Set key value pair", inserire il nome della chiave "Key" e del valore che si vuole le venga assegnato "Value".

Figure 15: Lista dei gruppi

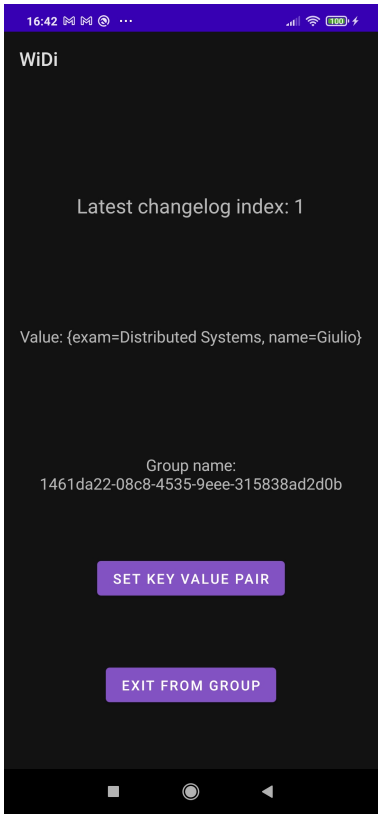

Figure 16: Gruppo

Accesso a un gruppo

- La vista a sinistra mostra la schermata con l'elenco gruppi, tramite la quale è possibile accedervi premendo sul tasto;
- La vista a destra mostra la schermata del gruppo vista dal *client*.

9 Conclusioni

Ho iniziato studiando diversi algoritmi di consenso e valutandone le differenze, per poi puntare a una semplice replicazione di dati con logica *master-slave*, data la difficoltà inerente nel partire subito con un implementazione complessa. Questo mi ha permesso di avere più tempo per approfondire i problemi di Wi-Fi Direct in Android e per trovare delle soluzioni agli stessi. Cercare di arrivare direttamente a un prodotto con un algoritmo di consenso funzionante avrebbe aumentato esponenzialmente la complessità del progetto, rischiando di portare a un fallimento. Ho quindi studiato per comprendere quale potesse essere l'architettura più intuitiva per la libreria, in modo da permettere un testing semplice. Successivamente, ho implementato e testato le varie funzioni. Data la difficoltà a testare l'app con i dispositivi reali, inizialmente i test sono stati eseguiti soprattutto in locale sfruttando JUnit. Solo in ultima fase ho sperimentato direttamente sull'app. Questo approccio diverso al codice mi ha permesso di risparmiare molto tempo, superando diverse criticità che erano troppo difficili da debuggare sui dispositivi.

9.1 Sviluppi futuri

Ho ipotizzato più sviluppi futuri per il progetto:

- il codice è strettamente legato ad avere un unico master: se questi esce non è più possibile fare operazioni sui dati condivisi, bisognerebbe rimuovere questo requisito permettendo agli *slave* di eleggere un nuovo master;
- attualmente, per aggiornare i nuovi *slave* con i dati contenuti nel *master* è necessario inviare tutto il changelog: per permettere l'eliminazione di parte del changelog e velocizzare l'aggiornamento degli *slave*, si potrebbe inviare direttamente la struttura dati;
- attualmente viene usata una struttura dati a mappa per mantenere i dati, la quale si potrebbe sostituire con uno spazio di tuple;
- si potrebbe sperimentare la sostituzione delle socket in favore di gRPC (attenzione: al momento della scrittura non è possibile avviare un server gRPC su Android [6]);
- al posto dell'uso diretto delle socket si potrebbe usare un protocollo standard come HTTP (esistono librerie che permettono di avviare server HTTP su Android [12]);
- si potrebbero scambiare i dati tramite BSON anziché Json per migliorare le performance.

9.2 Cosa ho imparato

Questo progetto mi ha permesso innanzitutto di conoscere Wi-Fi Direct e di approfondirne il funzionamento, ma ho anche compreso che una documentazione non esaustiva non permette all'utente finale di usare in maniera agevole il prodotto, rendendolo scarsamente utilizzato. La ricerca di accorgimenti per far funzionare il sistema è stata molto

interessante, dato che anche utilizzatori importanti di Android come Microsoft (nella sua implementazione di Tether per Cordova) hanno riscontrato i miei stessi problemi. Inoltre, la stessa ricerca mi ha permesso di capire che non è detto che ogni classe abbia le stesse implementazioni in tutte le versioni di Java; in particolare l'ho notato con Socket, che in Android ha avuto un comportamento notevolmente diverso. Il progetto, più in generale, mi ha permesso di estendere le mie competenze per quanto riguarda la programmazione mobile, Kotlin e i sistemi distribuiti, che sono tutti elementi fondamentali per consentire l'interazione a cui siamo abituati ogni giorno con i dispositivi mobili.

References

[1] Wi-Fi Alliance. Wi-fi direct specification. https://www.wi-fi.org/file/wi-fi-direct-specification.

[2] Android. *Instrumented Unit Tests*. https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests.

[3] Android. *Local Unit Tests*. https://developer.android.com/training/testing/unit-testing/local-unit-tests.

[4] Android. *WifiP2pManager*. https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.

[5] Crash-Test-Buddies. Wi-fi buddy. https://github.com/Crash-Test-Buddies/WiFi-Buddy/blob/master/README.md.

[6] Brandon Dutra. feature request: support grpc servers on android. https://github.com/grpc/grpc-java/issues/4407.

[7] Kotlin Foundation. *JvmSuppressWildcards - Kotlin Programming Language*. https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.jvm/-jvm-suppress-wildcards/.

[8] Kotlin Foundation. *Kotlin Coroutines*. https://kotlinlang.org/docs/coroutines-overview.html.

[9] Dropbox Inc., Google Inc., Skyscanner Ltd., and WeWork Companies Inc. *gRPC*. https://grpc.io/.

[10] MongoDb. *BSON*. https://bsonspec.org/.

[11] Oracle. *CompletableFutures*. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html.

[12] Piotr Polak. Android http server. https://github.com/piotrpolak/android-http-server.

[13] Thali Project. Android wireless issues. http://thaliproject.org/androidWirelessIssues/.

[14] Roboelectric. *Roboelectric*. http://robolectric.org/.

[15] Unknown. Wi-fi direct api for connection acceptance - issue tracker, 2012. https://issuetracker.google.com/issues/36946915.

[16] Wikipedia. Byzantine fault - wikipedia, 2021. https://en.wikipedia.org/wiki/Byzantine_fault.